TOWARDS A GEOMETRIC UNIFICATION OF EVOLUTIONARY ALGORITHMS

Alberto Moraglio

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ESSEX

NOVEMBER 2007

UNIVERSITY OF ESSEX

Date: **November 2007**

Author: **Alberto Moraglio**

Title: **Towards a Geometric Unification of Evolutionary Algorithms**

Department: **Computer Science**

Degree: **Ph.D.** Convocation: **November** Year: **2007**

Table of Contents

Abstract

Evolutionary algorithms are successful and widespread general problem solving methods that mimic in a simplified manner biological evolution. Whereas most of evolutionary algorithms share the same basic algorithmic structure, they differ in the solution representation – the genotype – and in the search operators employed – mutation and crossover – that are representation-specific. In the research community there is a strong feeling that the evolutionary computation field needs unification and systematization in a rational framework to survive its own exceptional growth of the last two decades. The lack of a common formal framework encompassing all solution representations have prevented EC researchers to build a truly general theory of evolutionary algorithms, as well as to develop a formal theory of search operators design for new representations and problems.

In this thesis, we propose a general geometric framework that addresses these two important problems. The unification is made possible by surprisingly simple representation-independent geometric definitions of crossover and mutation using the notion of distance associated with the search space. This novel way of looking at genetic operators allows us to rethink various familiar aspects of evolutionary algorithms in a very general setting, simplifying and clarifying their relations. We show that many important genetic operators for the most frequently used representations fit this framework. This makes this framework highly relevant because it unifies pre-existing evolutionary algorithms. The abstract definitions of mutation and crossover can be used as formal recipes to build new mutations and crossovers for virtually any new solution representations and problems. We designed and tested new operators on a number of problems obtaining very good experimental results. The same abstract definitions of mutation and crossover can be used to build a truly general representation-independent theory of evolutionary algorithms. We started building such a theory and showed that all evolutionary algorithms with geometric crossover does the same type of search, convex search. This is a general and important result because it shows that a non-trivial representation-independent theory of evolutionary algorithms is possible.

Acknowledgements

Looking back at my PhD journey as a whole, a rollercoaster of emotions runs through me. Many are those who made this journey a memorable one, and my gratefulness to them extends far beyond the boundaries of this page!

I am deeply indebted with Riccardo Poli, my supervisor, for shaping me as a researcher and for being an example of commitment to excellence and pure passion for research.

This thesis would not have been possible without the great help of various collaborators of mine, with whom I have one or more publications. They gave a valuable contribution to this research and supported me very much. In particular, I would like to express my sincere gratitude to: Yong-Hyuk Kim, Yourim Yoon and Byung-Ro Moon for their hard and meticulous work and exquisite politeness; Julian Togelius and Simon Lucas who showed me the fun part of research; Cecilia di Chio for being so humane and always willing to help; Rolv Seehuus for being the first who collaborated with me. Also, I wish to thank Yossi Borenstein for being always available to discuss new ideas, for his insightful remarks and for his constant support.

There would be so many people to thank for making the department of Computer Science such a friendly and pleasant place to study and work in. My first office mates deserve a special mention for their kindness and friendship, and for tutoring me when I first arrived at the university, especially Carlos Acosta and Li Hui. Also, I would like to thank Serafin Martinez and Biliana Alexandrova for their delightful hospitality in Mexico City on occasion of a conference.

I am very grateful to the department of Computer Science that has financially supported my entire PhD and for providing me with funds to attend many international conferences.

I would like to thank also all those researchers in the Evolutionary Computation community who believed in the theory presented in this thesis from its early stages, encouraged me and gave me important suggestions.

I wish to express my sincere gratitude to my family for their understanding, for their loving support and for being always close to me. Finally, I am most grateful to Fumiyo for her patience, for her ceaseless support and for making my existence a happier one.

Chapter 1

Introduction

Darwinian evolution – one of the most intriguing and powerful mechanism of nature – has been a constant source of inspiration for researchers interested in search and optimisation for over four decades, leading to the formation of a large research community and a huge body of literature which go under the name of Evolutionary Computation.

Evolutionary algorithms[1] (EAs) rely on surprisingly few ingredients of Darwinian evolution, namely:

Inheritance: individuals have a genetic representation (in nature, the chromosomes and the DNA) such that it is possible for the offspring of an individual to inherit some of the features of its parents.

Variation: the offspring are not exact copies of the parents, but instead reproduction involves mechanisms that create innovation, as new generations are born. Typically, variation is produced both through *mutations* of the genome and through the effect of sexually recombining the genetic material coming from the parents to obtain offspring chromosomes (*crossover*).

Natural selection: individuals best adapted to the environment have longer life and higher chances of mating and spreading their genetic material.

[1]Here, we use this term to denote those methods that are inspired to biological evolution. In particular, it does not encompasses Ant Colony Optimisation and Particle Swarm Optimisation.

The notion of individual in nature corresponds in EAs to a tentative solution to a problem of interest. The fitness of natural individuals corresponds to the objective function used to evaluate the quality of the tentative solutions in the computer. The genetic variation processes of mutation and recombination are seen as mechanisms (search operators) to generate new tentative solutions to the problem. Finally, natural selection is seen as a mechanism to promote the diffusion of the genetic material of individuals representing good quality solutions, and, thus, having the potential to create even fitter individuals (better solutions).

Typically EAs have the following form:

1. Initialise population and evaluate the fitness of each population member
2. Repeat until stopping condition is satisfied
   (a) Select sub-population for reproduction on the basis of fitness (*Selection*)
   (b) Copy some of the selected individuals (*Reproduction*), perform *Crossover* or *Mutation* on the remaining selected individuals
   (c) Evaluate the fitness of the new population

Although a lot more goes on in natural evolution, these ingredients are in fact sufficient to obtain artificial systems with the ability to find high quality individuals.

For all the phases in a EA there exist many different realisations. For example, there are many alternative genetic representations for solutions, the most important being binary strings, real-valued vectors, permutations, and trees. Also particularly striking is the case of the genetic operators (crossover and mutation), where not only there are completely different classes of operators for different representations but also, even focusing on one representation, there are at least as many alternative realisations of crossover and mutation in the literature as researchers working in evolutionary computation.

This is certainly a healthy situation, but it makes it difficult to understand the difference and similarities between different evolutionary methods. This has led us to start a research programme attempting to unify alternative operators and representations under a single more general theory.

1.1 Motivations

There are various flavours of evolutionary algorithms, the best known of which are genetic algorithms, evolution strategies and genetic programming. However, the current classification originates from historical reasons and is based on rather superficial features rather than on fundamental differences in operation.

Since they are easy to implement, robust and relatively simple to apply to new problems, EAs are extensively used in practice. Their scope covers combinatorial and continuous optimisation, program and structure design, search, machine learning, game theory, art and more.

For many real-world applications, EAs produce relatively good solutions in reasonable time (most of the times) requiring only some prior knowledge on the problem, embedded in the form of 'natural' solution representation and genetic operators. Their fitness function can be even noisy, dynamically changing, or inconsistent.

However, EAs perform less well than specialised algorithms for specific problems and tend to produce good solutions rather than global optima (on NP-hard problems). To overcome these limitations, they are easily and fruitfully hybridised with specific heuristics for the problem at hand. They can also be modified (biased) to embed in the search process extra knowledge on the problem, when available. EAs can be pushed toward optimisation rather than approximation by combining them with local search.

EAs theoretical foundation is still far from being satisfactory [150]. The existing theories lack (to different extents) of generality and explanatory power; they lack also of real utility for the practitioner in guiding the design of representations, operators, fitness functions, hybridisation, parameter settings, etc. Also, in sharp contrast to evolutionary computation practice, theory is far from simple.

In the research community there is a strong feeling that the evolutionary computation field needs unification and systematization in a rational framework to survive its own success (De Jong [27]).

Many evolutionary algorithms look very similar when cleared of algorithmically irrelevant differences, such as authorship, historical origin, domain of application, phenotype interpretation and representation-independent algorithmic characteristics that, in effect can be freely exchanged between algorithms such as for example selection scheme. Ultimately, the origin of the differences of the various flavors of evolutionary algorithms is rooted in the solution representation and relative genetic operators.

Are these differences only superficial? Is there a deeper unity encompassing all mutation and crossover operators, hence all evolutionary algorithms beyond the specific representation? Formally, is a general mathematical framework that unifies search operators for all solution representations possible at all? Would such a general framework be able to capture essential properties encompassing all EAs or would it be too abstract to say anything useful?

A number of researchers have been pursuing EC unification along this line. Although, so far, no one has been able to build a fully-fledged theory of representations. For example, Radcliffe pioneered a unified theory of representations [125], although he never used the word "unification"; Poli unified the schema theorem for traditional genetic algorithms and genetic programming [79]; Stephens suggested that all evolutionary algorithms can be unified using the language of dynamical systems and coarse graining [149]; while Rothlauf initiated a less formal theory of representations [137]; Rowe et al., building upon Radcliffe's work, has devised a theory of representation based on group theory [138]; Stadler et al. built a theory of fitness landscapes that connects with representations and search operators [135].

The lack of a unified way to deal with different solution representations has an impact on both theory and practice of evolutionary algorithms and it seriously limits them.

This is one of the reasons why evolutionary computation theory is fragmented, and it has led to the development of significantly different theories for different representations. This fragmentation is symptomatic of the fact that the very fundamental working principles underlying all evolutionary algorithms are not yet well understood. More fundamentally, the lack of a general framework for representations prevents us from investigating whether such common principles exist at all.

Evolutionary algorithms have been shown to be powerful tools to address a vast array of optimization, search, learning and design problems. However designing a good evolutionary algorithm for a new problem is more an art than a science. Preexisting theories give little or no guidance for the choice of solution representation and the design of search operators for new problems. The only theory that could in fact do this is a theory in which representations and operators are not fixed *a priori* but are treated as variable entities – in other words a theory that encompasses all representations.

It is, therefore, important, for both theory and practice of evolutionary algorithms, to devise a general mathematical framework encompassing all solution representations. The aim of this thesis is to set up such a framework and unveil its significance in terms of the consequences and insights brought about by seeing evolutionary algorithms from a new and more general viewpoint.

1.2 Research questions

General research questions are:

- *Is the unification of evolutionary algorithms within a general mathematical framework possible?*
- *What are the advantages and consequences of the unification?*

More specifically these can be broken down into the following questions:

1. *Representation-independent operators:* how can we define representation-independent genetic operators?
2. *Representation unification*: do abstract genetic operators generalise the main genetic operators for most frequently used representations?

3. *Evolutionary and neighbourhood search*: are neighbourhood search and evolutionary search dual and equivalent?

4. *Crossover principled design*: how can we do principled design of crossovers for new representations?

5. *Problem knowledge*: what is problem knowledge and how can we formalise and exploit it?

6. *Abstract evolutionary search*: is it possible to describe the search of a representation-independent evolutionary algorithm?

7. *Fitness landscape conditions*: on what class of fitness landscapes the formal evolutionary algorithm works well and why?

8. *Biology*: do abstract genetic operators generalise real biological operators?

1.3 Overview of the achievements

In this section, a short overview of the major research achievements is reported.

1. *Possibility of unification*: mathematical unification is possible. We have developed an axiomatic geometric framework for unification. The definition of search operators is axiomatic and intentionally does not involve the notion of representation (unification by abstraction of the representation[2]).

2. *Scope of unification*: the significance of unification lies on how many interesting cases it encompasses. We have shown that many interesting pre-existing operators for the most-used representations fit the requirements of the unification. The representations we have studied are: binary strings, real vectors, permutations, sets, syntactic trees, variable-length sequences and a number of extensions and combinations of them.

[2]This does not mean we regard solution representation as an unimportant "implementation detail". It simply means that the relationship between *representation and search operators*, which is ultimately what is relevant to the search, can be expressed in a geometric language that is representation-independent.

3. *Clarification and simplification*: the change in perspective coming with the unification clarifies and simplifies a number of fundamental notions: mutation and crossover are seen as searching the same landscape; crossover is now seen associated with the classic, simple notion of fitness landscape; neighborhood search and evolutionary search are shown to be dual and equivalent; evolutionary algorithms are seen as problem-independent and representation-independent formal algorithms; problem knowledge is understood as smoothness of the fitness landscape; the role of the EA designer has been clarified.

4. *Crossover principled design*: by applying the abstract definition of crossover to a distance firmly rooted in a specific representation one obtains a formal recipe to build new crossovers for any representation. When the distance chosen as basis for the new crossover is "meaningful" in terms of the problem addressed, the new operator is likely to perform well. We have shown how to do crossover principle design and tested it on a number of problems obtaining very good results.

5. *Significance of a general theory*: A fundamental result is that all evolutionary algorithms with any solution representation and with search operators that fit the definition of abstract crossover do the same search: convex search. This is a very general and deep result arising from the axiomatic definition of crossover only. The importance of the convex search result stems from the fact that knowing how all evolutionary algorithms with crossover search the space is preliminary to understand under what formal, general, representation-independent condition on the fitness landscape they all perform well.

1.4 Organisation of the thesis

The thesis is organised as follows.

In chapter 2 we present a literature survey. In chapter 3 we introduce the geometric framework and show how it generalises traditional mutation and crossover for binary strings.

In chapters from 4 to 8, we show how the most important solution representations fit the geometric framework. We have considered the following representations: real vectors, permutations, sets, trees and sequences. Each representation is also used to illustrate a different aspect of the geometric framework: we show important differences on how continuous spaces and discrete spaces fit the framework (chapter 4); show how to do crossover principled design for a number of problems (experiment for TSP, N-queens problem) and how the geometric framework unifies together representation-based and neighbourhood-based meta-heuristics (chapter 5); how different representations can give rise to equivalent search spaces (chapter 6); how some solution representation cannot be associated with a neighbourhood structure (chapter 7); and how the geometric framework encompasses naturally both artificial and biological evolution (chapter 8).

In chapter 9 we introduce the product geometric crossover that allows to build new geometric crossovers by combination of known geometric crossover. We illustrate its use on the Sudoku puzzle.

In chapter 10, we introduce the quotient geometric crossover that is cental to understand the effect of crossover at a phenotypical level. As an application of this, in chapter 11 we present a case study: the multi-way graph partitioning problem. The geometric framework here is used to deal with redundant solution and hard constraints.

In chapter 12 we generalise geometric crossover to the multi-parent case and show how this allows us to generalise Particle Swarm Optimisation (PSO) to any solution representations. We design new PSOs, for the Euclidean, Manhattan, Hamming and permutation swap spaces.

In chapter 13 we show that surprisingly one-point crossover has a geometric interpretation and show that a number of operators across representations are one-point geometric.

In chapter 14 we show the subtle ways in which geometric crossover relates with the notion of distance, prove the existence of non-geometric crossovers, hence the non-tautological nature of general theory of geometric crossover.

In chapter 15 we show that all geometric crossovers perform the same type of search: convex evolutionary search. This is a deep result.

In chapter 16 we draw conclusions and present future work.

1.5 Publications

This thesis is based on a number of coauthored publications. These publications are mostly work of the author of this thesis. The contributions of the coauthors is as follows. Most of the publications are coauthored with Riccardo Poli whose role was supervising and providing feedback. The other co-authors helped substantially with or carried out the experimental part in the co-authored publications, except for Yong-Hyuk Kim, Yourim Yoon and Byung-Ro Moon, who also helped to develop part of the theory (related to quotient spaces) in the co-authored publications.

Chapter 1, is partially based on the paper "Geometric unification of evolutionary algorithms"[87]. The literature survey, chapter 2, is unpublished material. Chapter 3, on the geometric interpretation of crossover, is based on the paper "Topological interpretation of crossover"[95] coauthored with Riccardo Poli. The work on real-code representation in chapter 4 is an extension of the paper "Geometric crossover for real-code vectors" [169] coauthored with with Yourim Yoon, Yong-Hyuk Kim, Byung-Ro Moon. Chapter 5 on permutations is partly based on the paper "Topological crossover for the permutation representation" [106] coauthored with Riccardo Poli and additional unpublished material. Chapter 6 on sets is based on the paper "Geometric crossover for sets, multisets and partitions" [101] coauthored with Riccardo Poli. Chapter 7 on syntactic trees is based on the paper "Geometric landscape of homologous crossover for syntactic trees" [97] coauthored with Riccardo Poli. Chapter 8 on sequences is based on the paper "Geometric crossover for biological sequences" [107] coauthored with Riccardo Poli and Rolv Seehuus (the experimental part carried out by Rolv Seehuus is not included in this thesis). Chapter 9 on product crossover is based on two papers, "Product geometric crossover", [103] coauthored with Riccardo Poli and "Product geometric crossover for the Sudoku puzzle" [110] coauthored with Julian Togelius and Simon Lucas. Chapter 10 on quotient crossover is unpublished material (partly based on common work in progress with Yong-Hyuk Kim, Yourim Yoon and Byung-Ro Moon). Chapter 11 on graph partitioning is based on the paper "Geometric crossovers for multyway graph partitioning" [89] coauthored with Yong-Hyuk

Kim, Yourim Yoon, Byung-Ro Moon. Chapter 12 on geometric PSO is based on three papers "Geometric Particle Swarm Optimization" [88] and "Geometric particle swarm optimization on binary and real spaces: from theory to practice" [31] coauthored with Cecilia di Chio and Riccardo Poli and "Geometric PSO for the Sudoku Puzzle" [109] coauthored with Julian Togelius. Chapter 13 on one-point geometric crossover is unpublished material. Chapter 14 on non-geometric crossover is based on the paper "Inbreeding properties of geometric crossover and non-geometric recombinations" [105] coauthored with Riccardo Poli. Chapter 15 on convex search is unpublished material. Chapter 16 with the conclusions and future work is unpublished material.

The author of this thesis has also coauthored a number of published papers on the geometric framework that are not listed above [90, 92, 108, 72, 143, 86, 102, 99, 100]. Additionally he has also a number of technical reports on the geometric framework [91, 93, 104, 98, 96, 94, 169, 170].

Chapter 2

Literature Survey

The individual chapters of this thesis are relatively self-contained and include references to relevant existing work. In this chapter a literature survey is presented in the attempt to depict the state of the art of evolutionary computation theory. It is intentionally broad and touches upon all the areas of knowledge related to the research focus of this thesis.

2.1 Evolutionary algorithms

There are various flavours of evolutionary algorithms [164]. The usual classification is done mainly on an historical basis rather than on other functional similarities (e.g. encoding, operators used, selection/replacement scheme and solution interpretation). A collection of seminal papers covering the origins of evolutionary computation is presented by Fogel [40].

Genetic algorithms (GAs), introduced by Holland [59] and made widely known by Goldberg [51], work on fixed binary encodings, use both crossover and mutation, but give strong emphasis to crossover. Also, they traditionally use fitness proportional selection and solutions are decoded from binary strings (genotypes) to actual solutions (phenotypes) via a genotype-phenotype map. GAs can be used in various tasks: search, optimisation, machine learning. Evolution Strategies (ES), developed by Rechenberg and Schwefel [132], use a direct real encoding, often do not use recombination and tend to use very small populations. Their main purpose is continuous function optimisation. Evolutionary Programming (EP), introduced by Fogel [41], is similar to evolution strategies, the main difference being that the original purpose was to evolve finite state machines instead of vectors of reals. Genetic programming, made well-known by Koza [77], works on variable-size trees, use crossovers and mutations tailored to this representation and interprets solutions as computer programs.

In an optimisation perspective, evolutionary algorithms are not ready-made algorithms (problem solvers). Rather they consist in a design method (a structured framework or a meta-heuristic) inspired by natural evolution which guides the designer in building an approximation algorithm which can find good solutions to the specific problem at hand. Solutions are represented using a representation that best suits the problem domain and genetic operators (mutation and recombination) are developed for the specific representation.

A more rational classification of evolutionary algorithms would be based on the properties of the solution space being actually explored as suggested in the following section.

2.2 Optimisation problems

Continuous optimisation problems are defined over a set of continuous (real) variables. The goal is to search this multidimensional Euclidean space to find the optimum. There are specific techniques (e.g., gradient-based methods) for this specific domain for some classes of functions (e.g., continuous, convex). There are also EAs for continuous optimisation working on real-coded solutions [132].

Combinatorial optimisation problems are defined over a finite collection of discrete solutions. The goal is to search this collection to find the optimum. Many combinatorial optimisation problems are difficult (NP-Hard). For certain problems there are approximation algorithms [6]. However, in practice these problems are normally attacked using local search and other meta-heuristics techniques (EAs are a subclass of such techniques). Meta-heuristics are quite easy to apply and give good practical (average) results but there are very few computational complexity results for them.

EAs applied to combinatorial optimisation problems use problem-specific representations and genetic operators [122]. Domain-specific representations and operators speed up the search by using knowledge of the problem at hand. The choice of the representation and related genetic operators is crucial to get a good genetic algorithm (reaching good quality solutions in acceptable time) for a specific problem. A lot of work has been done, but a theoretically principled design methodology for domain-specific representations and associated genetic operators is still lacking.

Genetic algorithms are stochastic optimisers. They are relatively good at identifying promising search regions but are perhaps not as effective at exploiting local optima. For this reason they are sometimes profitably coupled with local search (which has complementary characteristics: it is good at finding local optima but gets stuck there) to improve their performance.

In the most general case, combinatorial design problems [136] are optimisation problems defined over an infinite set of discrete solutions (structures) of variable size. However, sometimes combinatorial design problems are restricted to a finite set of discrete solutions by giving a limit on the maximum solution size. The goal of a combinatorial design problem is to find a structure that maximizes a given objective function. This can be viewed interchangeably as a design or as a search problem. A design problem cast as an optimisation problem looks similar to a combinatorial optimisation problem with two major differences: the interpretation of the solution of the problem addressed (design rather than optimisation) and the additional phenomena affecting the search process due to variable size of solutions (e.g., bloat).

Genetic programming [77] (GP) (where the structure of a solution is a tree and the solution is interpreted as a program or a function), together with all the other flavours of EAs working with variable-length solutions, can be seen as combinatorial design techniques instead of program induction techniques. This is not the familiar way of looking at GP. However, when GP is seen in this way, it becomes possible to think of a formal general optimisation framework encompassing continuous optimisation, combinatorial optimisation and design problems. Also in GP the choice of the representation and genetic operators is crucial to get a good EAs. A unified theoretical framework would then be applicable to GP as well, and more generally to all variable-length representations, benefitting these representations in terms of principled design of genetic operators and analytical results on expected performance.

2.3 Evolutionary computation theories

The main objectives of evolutionary computation theories are (or should be) explaining why and how evolutionary algorithms work, how well they work for a specific problem (landscape), tell us about their convergence properties, guiding evolutionary algorithm design (representation and genetic operators), recommending specific parameter setting for a given time vs. quality trade off. Ideally, they should do all this for any problem, any representation, and all genetic operators in a straightforward and computationally inexpensive way. The state of the art of evolutionary computation theory is, however, far from this ideal scenario. The main results to date are summarised below (see the book of Reeves and Rowe [134] for a good introduction on this subject).

An initial explanation of the qualitative behaviour of genetic algorithms relied on the famous (and controversial) Holland's Schema Theorem [40] and on the Building Block Hypothesis (Goldberg) [51]. This theorem was initially stated only for binary representations and then believed to hold for virtually any other representation.

The building block hypothesis is not generally accepted as a proper explanation (Grefenstette) [54]. Intuitively, the schema theorem tacitely assumes some form of correlation among the fitness of the solutions matching the same schema[4]. The difficult part is to understand and capture the essence of this relationship in a formal way and integrate it explicitly in the theory. Inspired by the schema theorem and abstracting from it Radcliffe [125] proposed a principled design technique for genetic operators defined over domain-specific solution representations that tries to make explicit use of the correlation between fitness of the solutions belonging to the same schema.

The schema theorem has been improved (Stephens) [148] (made exact, thereby making it possible to study the dynamic of genetic algorithms) and generalized for variable-length genetic algorithms and genetic programming (Poli) [79]. Its explanation power resides in the concept of schema (partial solution) which is what a genetic algorithm with crossover operator actually processes. Initially Holland [59] suggested that good solutions are built by combining highly fit and short schemata (building blocks). However, these more recent investigations have shown that the building blocks used by crossover are neither necessarily short nor highly fit. As long as there are many of them, crossover will recombine them with high probability. The schema theorem in its exact form indeed describes accurately the behaviour (dynamics) of GAs, but it does not say for what general class of fitness landscapes they perform well and why. In fact, because of the NFL theorem[165] (see Section 2.6) GAs cannot be expected to perform better than random search on average over all fitness landscapes.

Alternative models of GAs have also been proposed. Vose [158] models a simple genetic algorithm (defined over binary strings) using Markov Chains describing the stochastic evolution of the population undergoing the application of genetic operators. It is a very interesting approach. Unfortunately it has various limitations (at least in its original form): it makes quite restrictive assumptions on population sizes (infinite or very small[1]), it works only for binary representations and standard operators, it has been applied only to toy problems like one-max and it increases exponentially in computational space and time required with the size of the strings. However, recently there have been efforts to applying this model to more general representations [79] [146].

Other models of genetic algorithm operate at a different level of granularity. Rather than focusing on the dynamics of population of solutions (like Vose) or of specially (syntactically) related groups of solutions (schemata), they work in the fitness domain. Coarse-grained variables describing the state of the system (distribution of the fitness in the population, syntactic correlation of solutions in the population) are identified and equations derived to model the system dynamic (due the applications of genetic operators) through those macro-variables. These models are really powerful and computationally tractable. They have been successfully applied to analyse the dynamic of genetic algorithms for less simple problems (but still a caricature of real problems) arising in real world combinatorial optimisation problems (Rogers, Prugel-Bennett) [124]. However, a major drawback of these models is that they are very problem-dependent and

[1]In principle this model does not have any restriction on population size. However, if one wants to execute the model on a computer only very small populations can be used.

their application to a new problem is far from being standard (you need the help of a talented physicist!).

Others prefer a collection of simple approximate models to describe particular (and important) aspects of genetic algorithms' dynamics (and outcome) without aiming to describe the whole complexity of the evolving system (Goldberg) [52].

Concerning models for suggesting optimal parameter settings, general models are still unavailable. For evolutionary algorithms using only mutation on real-coded solutions there are models to suggest the best value for the mutation parameter [132]. There are also results on adaptive parameter setting for EAs [142].

The trend of evolutionary computation theories is toward integration and unification into a common framework [150]. There is a need for a theory able to cover general solution representations (like graphs) and any (generic) operators. It is unclear, however, if such an ideal theory would be able to say something useful about any problem addressed with EAs. A clarification of basic (and sometimes implicit) assumptions, theoretical limits and explanation power (and therefore utility) of evolutionary computation theories is needed.

2.4 Fitness landscapes

Fitness landscapes (see [135] for a good overview) are of central importance to any flavour of search algorithm (and in other fields too: e.g., evolutionary biology and physics). A fitness landscape is the fitness function of a given problem instance together with a neighbourhood structure connecting solutions. The neighbourhood structure is not directly defined in a search algorithm, are the specific search operators used by the search algorithm at hand that give rise to the neighbourhoods. In fact, one of the contribution of this thesis is to clarify and simplify the relationship between search operators and neighbourhood structure. In chapter 3, we give a formal definition of fitness landscape and present a novel way to relate search operators and neighbourhood structure. In the following, we present an intuitive picture of various aspects of fitness landscapes.

The neighbourhood structure induced by the search operators affects enormously the effectiveness of the search, much more than other features of the search algorithm. The same instance of the same problem under one neighbourhood structure can be much easier to search than with another. Nevertheless, perhaps surprisingly, choosing a good neighbourhood structure cannot transform the computational complexity of a problem because this is a measure of inherent difficulty of the problem (defined in terms of scalability over all the instances of a problem in the worst case scenario). So, unless $P = NP$, an NP-hard problem will remain still difficult (NP-hard) because the computational complexity class of a problem does not depend on what algorithm one uses to solve it. Hence, ultimately it does not depend on the specific neighbourhood structure induced by the search operators of the search algorithm used to solve it. However, the problem may become easier to search in practice when the aim is for good solutions (approximations), and not for the optimum, and when the instances of the problem occurring in practice are not the worst possible. Conversely, a bad choice of neighbourhood for a relatively easy problem may render the fitness landscape very hard to search.

The use of the word "landscape" is to suggest a way to interpret the behaviour of search algorithms and visualise the difficulties arising in finding the optimum solution. Supposing the search algorithm has to maximise the fitness (for a minimisation problem similar considerations can be made) the optimum solution can be seen as the highest peak in the landscape. Local optima are any other lower peaks in the landscape.

Local search algorithms start from a given (random or created somehow) point in the landscape and climb up until they reach the top of a local peak (normally the nearest). This is not guaranteed to be the optimum solution (global optimum). Indeed, it may just be a local optimum and the global optimum can be far away and of much better quality.

The "metaphor" of landscape suggests what characteristics may make it hard or easy to search at an intuitive level. If there is only one peak, the landscape is said to be unimodal, if there are many then is called multimodal. The landscape can be smooth or rugged. The landscape can also be flat or steep. In general, multimodal landscapes are harder to search than unimodal, but not always (see Needle-in-Haystack for a counterexample). Ruggedness is an unpleasant fitness landscape characteristic because introduces very many local peaks making pure local search algorithms completely ineffective. A smooth landscape is preferable but not a flat one since it gives no information at all about the direction to follow. A landscape can be a combination of the characteristics mentioned above.

In contrast to local search, evolutionary algorithms, and more in general population-based algorithms, perform a parallel search on the entire landscape throwing a net and then concentrating the search on the most promising regions, on average.

Above we introduced some of intuitive characteristics of a fitness landscape that may make it hard or easy to search. However, there are various methods to measure more precisely the hardness of fitness landscapes or classifying them in classes of difficulty [65][9][62]. None of them, however, is a perfect measure for all landscapes. There are landscapes that are assessed to be easy (or hard) by a given measure but they turn out to be hard (easy) in practice.

The correlation function method[160] consists in running a number of random walks of a given length and then comparing the fitness of the starting solution and the final solution. If these fitness on average differ a lot then the landscape is understood to be difficult to search. A different method [82], similar in the spirit, is running a simple local search algorithm various times and see on average the difference in fitness among the starting solution and the fitness of the local optimum reached. Bigger difference translates into an easier search, hence a easier landscape.

A different class of methods rely on the concept of epitasis [57] (that is in fact a measure of inner decomposability of the problem). Problems that present higher level of epitasis are thought to be inherently more complex because in order to be solved the right conjunction of a greater number of elements has to be identified.

The Walsh transform [130] of a fitness landscape is useful to represent a landscape defined over binary strings. From the Walsh coefficients (which in general are exponentially many in the length of the encoding) various properties of the landscape can be derived. Intuitively, simple landscapes are more decomposable and have only few non-zero Walsh coefficients (e.g., in One-Max only the order 1 Walsh coefficients are non-zero). The Walsh transform has also an interesting connection with schemata, allowing calculating statistics of a given schema (mean, variance and all the higher order statistics) in a straightforward way [49] [50]. By using the Walsh transform and its natural affinity with schemata it is possible to create deceptive (misleading) landscapes for genetic algorithms based on the schema theorem [162].

Besides measuring the difficulty of a given landscape, researchers wanted to build artificial fitness landscapes belonging to a controlled class of difficulty to test and understand theoretical aspects of evolutionary algorithms (and other meta-heuristics) more easily. NK-landscapes [9], Needle-in-a-Haystack, One-Max, [84] and many other artificial landscapes [65] [9] are all built to test or to emphasise specific aspects of evolutionary algorithms.

Whereas local search algorithms have a natural interpretation in the fitness landscape, evolutionary algorithms are more problematic. Local search algorithms can be seen as following a path in the landscape from the initial solution to a peak close to the initial point. Variations of local search algorithms which employ strategies to avoid getting stuck in local optima (for example, Taboo Search [48]) have a natural interpretation in the landscape as well. What makes the interpretation of evolutionary algorithms in the fitness landscape problematic is the presence of the crossover operator. A (simple) mutation can be thought as a random step of local search. Recombination, differently from mutation, is an operator working on (normally) two parent solutions and producing one or two offspring solutions. Recombination seems not to have an intuitive interpretation in the fitness landscape. The concept of hyper-neighbourhood has been introduced [135] (a multidimensional neighbourhood) with the purpose of explaining the effects of recombination. This interpretation, however, does not put in the same framework different operators. Hence, it does not make it possible to directly compare them nor to understand their interactions. An effort to comprehend the interconnection between manipulating syntactic representation of solutions (by recombination, mutation and other operators) and their topological interpretation in the classic neighbourhood structure of the solution space can result in a deeper understanding of the mutual effect of operators, a clarification of what recombination in general is and a cross-fertilization of evolutionary algorithms and local search.

It is worth trying to understand how the computational complexity of a problem at hand affects the shape of a landscape. The landscape is not given with the problem; it is rather part of the design of search algorithm. However, hard combinatorial optimisation problems (NP-hard) are inherently hard and this hardness must be reflected in the fitness landscape. Understanding how complexity of problem and shape of landscape are linked is extremely important to design effective search algorithms. (Section 2.5 reports an overview on computational complexity.)

The fitness landscape idea is strictly connected with black-box optimisation. On the one hand, when looking at a problem in terms of fitness landscape, the original problem disappears (its definition and complexity class are "lost") and it is replaced by a topology over the solution space. On the other hand, in the black-box optimisation model, the original problem is thought to be in a black box, the only pieces of information known about it are obtained by feeding the box with a solution and getting feedback on its quality (fitness). In practice, the interface with this black-box requires a given encoding (representation) for input solutions (like binary encoding). Such an encoding together with a metric defined over it (like Hamming distance for binary strings) induces a neighbourhood relationship among solutions (hypercube structure for binary strings). This neighbourhood structure over the solution space together with the whole of solution fitness is a fitness landscape. So, fitness landscape and black box optimisation are really strictly related concepts.

When transforming the problem at hand from its original form (its definition by a mathematical formula) into a fitness landscape some information is lost. In a sense all information about the problem is in the landscape, since all the problem solutions are there. In another sense, some potentially important information is lost, which is the explicit definition of the problem. Metaphorically speaking, the explicit definition of the problem can be thought as a "map" of the landscape: even though one can retrace the map from the landscape itself, searching a landscape using a map can be far easier! Considerations about the role of problem knowledge in search are presented in Section 2.6.

2.5 Computational complexity

Computational complexity concerns the analysis of resources (time and space) needed by algorithms to solve problems [120]. The computational complexity class of a problem defines the inherent complexity of the problem (for example, the sorting problem has time complexity $O(n \log n)$). The computational complexity of an algorithm devised to solve a problem is the analysis of resources used by the algorithms to solve the problem (continuing the example, Bubble Sort is a sort algorithm and its complexity in time is $O(n^2)$).

Most combinatorial optimisation problems belong to the NP-Hard complexity class and are believed not to be solvable in polynomial time (there is very strong evidence but not yet a proof). Solvable means that the supposedly polynomial solver algorithm has to find the optimal solution to every instance of the problem in polynomial time.

There are various ways to relax the conditions required of a solution to make a hard problem efficiently solvable [6]. The first one is devising algorithms running in polynomial time for a subset of the instances of the problem. For example, the general travelling salesman problem (TSP) is NP-hard, but imposing some restriction on the distance between cities makes it more tractable and solvable in polynomial time [6]. Also, there is interesting work on the so-called phase-transitions in the complexity of problems, trying to delimit where the easy (tractable) instances end and the hard ones begin [45].

The second way to make a hard problem more tractable is to relax the condition of optimality. One can relax it by requiring the algorithm to solve the problem within certain bounds of time and approximation to the optimum. This approach leads to a hierarchy of approximation complexity classes on the basis of the tightness of the bound [6]. However, there are problems for which negative results about those classes have been proven. For example, for the general TSP there are no approximation algorithms that can guarantee performance in polynomial time (if $P \neq NP$)[6].

Since for interesting combinatorial optimisation problems it is not possible to find algorithms with guaranteed performance in the above sense, there is an increasing interest in randomised algorithms which can be run many times on the same problem instance producing different results [111]. Since the solution found by randomised algorithms is a random variable, the performance bound to characterize the complexity of these algorithms is on the *expected* approximation rate to the optimum. A randomised algorithm with good expected performance is not guaranteed to find a good solution in all executions, but it does so on average. Evolutionary algorithms are randomised algorithms and could fit well in this complexity context.

Even when theoretically-sound approximation algorithms are available, they may be too inefficient or hard to implement. Practitioners are not interested in performance guarantees; they just want to generate good solutions to a problem in a reasonable amount of time. This is usually done by using heuristics specifically tailored to the problem at hand as a starting point, followed by an improvement phase using one of the many meta-heuristic techniques (e.g., local search or evolutionary algorithms just to mention two). Meta-heuristics are relatively easy to apply to a vast arrays of problems, can easily accommodate and exploit problem knowledge and, when well-designed, they are quick and work well on many problem instances occurring in practice. However, they do not have any performance guarantee and so they could deliver arbitrarily bad solutions for though instances of the problem.

At present, the only way to tell whether a meta-heuristic will be effective for a given problem is to implement it and run it. There is no adequate theoretical understanding of the design of search neighbourhoods (representations and genetic operators for evolutionary algorithms) and search strategies. Building a theory able to characterise some form of performance guarantee for meta-heuristic is another open challenge.

2.6 Problem knowledge

The use of problem knowledge in search makes the difference between a poor and a successful search algorithm [1]. However, problem knowledge (or lack of it) does not only have a practical use, it can lead to very strong theoretical results too (e.g., no free lunch theorem [165]). Problem knowledge is an intuitive notion, not a formal one. In the following, we present a categorisation in the attempt to clarify its role and use in search algorithms.

There are two types of problem knowledge used in search algorithms. The first is the knowledge of the problem in general (for example knowing that the problem is a TSP). The second is the knowledge of the specific problem instance addressed (continuing the example, knowing the number of cities and the exact distance among all cities).

The first kind of knowledge is generally embedded in the search algorithm at a design time through the choice of the neighbourhood structure for local search algorithms and the choice of the representation and genetic operators for evolutionary algorithms. If the objective function is not already given, it can be designed as well using this knowledge to guide the search.

The knowledge on the specific problem instance can be expressed in two different forms: by the mathematical expression of the objective function (or some other equivalent concise form) or by the whole solution space (enumeration of all solutions). This distinction gives rise to two different (complementary) approaches to solve problems by search. The first category comprises all the techniques that use the definition of the problem (mathematical expression) to deduce incrementally the optimal solution while minimising the actual search effort. They do so by exploiting general properties of the solution space (for example branch and bound algorithms or linear programs solvers). They can be seen as top-down approaches, working from general to particular. The second category encompasses all the techniques based on the black-box computational model. These techniques try to induce the optimal solutions by guessing solutions (or identifying them following a strategy), test them in the black-box and using the feedback to guide the following guesses. These methods do not use concise knowledge (mathematical expression) to guide the search (nevertheless, in fact they do use knowledge that is just expressed in a different form). They can be seen as bottom-up approaches because they work on specific solutions and generate and test (trial and error) hypotheses on the solution space (the location of the optimum). In this classification, evolutionary algorithms are black-box methods.

The No Free Lunch (NFL) [165][166] theorem proves that when the performance of a (black-box-model-based) search algorithm is averaged over all possible search spaces (over all fitness functions) its performance will be the same as that of random search. Put another way, if search algorithm A outperforms algorithm B on a set of problems then there must exist problems on which algorithm B outperforms algorithm A, so that when averaged over all possible functions their performance is the same. This theorem has serious implications for anyone trying to search a search space or equivalently optimise a fitness function. Most serious of all is that the theorem proves that there cannot be a single algorithm that is best on all problems. There is no holy grail.

In practice, however, it seems that some search algorithms (say EAs) are much better than others (pure random search). How can the NFL implications be reconciled with practical experience?

The first candidate explanation is based on the class of Real World Problems [144], and it goes as follows. The NFL holds for performance averaged over the entire set of possible problems[2]. However, it is not clear whether it holds on the class of Real World Problems that is a particular subset of the entire set of problems. If it does not, than it is not true that all algorithms will perform equally in practice, and it is this set that people are really interested in. The challenge, therefore, becomes understanding what is peculiar about real world problems and, consequently, which specific classes of search algorithms could be advantageous.

A second explanation relies on the notion of problem knowledge. Evolutionary algorithms, local search and in general all meta-heuristics are not ready-made search algorithm, but rather they are algorithmic templates that are customized by the designer to the specific problem by choosing suitable neighbourhood structures, solution representations and search operators. It is this problem knowledge embedded in the search algorithm that makes EAs ultimately perform better than random search. The NFL does not applies here because the designer adapts the search algorithm to the problem. If the problem changes, the designer will adapt the same

[2]The NFL holds also for smaller sets, e.g., sets of functions closed under permutation.

algorithmic template to the new problem. The challenge is, therefore, understanding how to model formally this type of problem knowledge, to study how it affects search performance and build a theory that guides the design of neighbourhoods, representations and search operators that make explicit use of it.

2.7 Open questions and challenges

From the previous survey various big open questions arise:

- *Evolutionary Algorithms for Optimisation Problems*: How can evolutionary algorithms be best applied to optimisation problems?
- *Evolutionary Computation Theory*: Why and how evolutionary algorithms work? How can evolutionary computation theory be unified and generalized? What are the roles and the limits of evolutionary computation theory?
- *Fitness Landscape*: What really makes a landscape difficult to search? How are evolutionary algorithms and fitness landscapes related?
- *Problem Knowledge*: How does problem knowledge affect search algorithms' performance in practice and in theory?
- *Computational Complexity*: How could computational complexity theory be extended to evolutionary algorithms?

These questions unveil three major challenges for the theory of evolutionary algorithms:

1. *Principled Design of Evolutionary Algorithms*: The first one is about the need for a principled design, with a strong theoretical basis, of evolutionary algorithms for generic combinatorial optimisation and combinatorial design problems. Ideally the principled design guide should be able of suggesting good representations together with genetic operators to any problem.

2. *Unification of Evolutionary Computation Theories*: The second challenge is about a grand unification and generalization of the existing evolutionary computation theories. This unification may pass through the extension of the theory to very general structures (like graphs) or, abstracting from the structure, to virtually any structure. The resulting theory should also be able to say useful things about the application of evolutionary algorithms to hard optimisation problems rather than to toy problems.

3. *Computational Complexity of Evolutionary Algorithms*: The third challenge is about a critical review of the existing corpus of evolutionary computation theory adopting standard theoretical tools from computer science. The field of computational complexity and a careful analysis of the hidden assumptions about problem knowledge can lead to a comprehension of theoretical limits and define more precisely role and explanatory power of evolutionary computation models.

These three research avenues are not completely independent. Indeed, a principled design of evolutionary algorithms needs a strong evolutionary computation theory supporting it. The latter, in turn, needs to be firmly understood in the broader realm of computational complexity.

How do these challenges relate with the focus of the thesis? As stated in the introduction, the aim of this thesis is to construct a formal framework that allows us to handle in a unified way search operators for all solution representations. This framework lays the foundation for a general theory of evolutionary algorithms (challenge 2) and at the same time it is the foundations of a formal theory of search operators design (challenge 1). Provided that a general computational complexity result for evolutionary algorithms is truly possible (challenge 3), this framework may be the starting point upon which to build it.

Chapter 3

Geometric Interpretation of Crossover

In this chapter we give a representation-independent geometric definition of crossover that links it tightly to the notion of fitness landscape. Building around this definition, a geometric framework for evolutionary algorithms is introduced that clarifies the connection between representation, genetic operators, neighbourhood structure and distance in the landscape. Traditional genetic operators for binary strings are shown to fit the framework. The advantages of this interpretation are discussed.

3.1 Introduction

Fitness landscapes and genetic operators have been studied for considerable time in connection with evolutionary algorithms. However, a unifying theory of the two is missing and many questions about their relationship remain unanswered. Below we will briefly analyze the current situation in this respect.

Fitness landscapes and genetic operators are undoubtedly connected. Mutation is intuitively associated with the neighbourhood structure of the search space. However, the connection between landscape and crossover is less clear. Complicated topological structures, hyper-neighbourhoods, have been proposed [24] [64] [46] [135] to formally link crossover to fitness landscapes. However, even when using these ideas, effectively one is left with a different landscape for each operator [24], which is deeply unsatisfactory. Important questions then are: is there an easier way of interpreting crossover in connection to fitness landscapes? Are crossover and mutation really unrelated?

An established way of defining a fitness landscape for search spaces where a natural notion of distance exists is to imagine that the neighbourhood of each point includes the points that are at minimum distance from that point [9]. Once a landscape is defined, typically the notion of distance is not used further. Couldn't distance play a much more important role in explaining the relationship between landscapes and crossover?

Local search and many other meta-heuristics are naturally defined over the neighbourhood structure of the search space [48]. However, a peculiarity of evolutionary algorithms (seen as meta-heuristics) is that the neighbourhood structure over the search space is specified by the way genetic operators act on the representation for solutions. One may wonder whether it is possible to naturally reconcile these two ways of defining structure over the search space.

In yet another sense, solution representation and neighbourhood structure are just two different perspectives on the solution space. An example is the classical binary string representation and its geometric dual, a hypercube, which has been extremely useful in explaining genetic algorithms [161]. Can solution representation and neighbourhood structure be two sides of the same coin for other representations, like permutation lists or syntax trees?

The traditional mutation and crossover operators defined for binary strings have been extended to other representations [79]. Also, there are general guidelines for the design of such operators for representations other than binary [129] [151]. Is there a way to rigorously *define*, rather than *design* or *extend*, mutation and crossover in general, independently of the representation adopted?

Except for solution representations, many evolutionary algorithms are very similar which suggests that unification might be possible [150]. Are all evolutionary algorithms really the same algorithm in some sense?

In this chapter we clarify the connection between representation, genetic operators, neighbourhood structure and distance and we propose a new answer to the previous questions. The results of our work are surprising: all the previous questions are connected, and the central question to address is really only one: *what is crossover?*

This chapter is organized as follows. In section 3.2, we introduce some necessary definitions. Geometric definitions of crossover and mutation are given in section 3.3, where we also prove some properties of these operators. As an example, in section 3.4, we show how traditional mutation and crossover defined over binary strings fit our general geometric definitions for mutation and crossover. In section 3.5, we give an overview of properties and implications of our geometric interpretation of crossover. In the following sections, we consider some of them in greater detail. In section 3.6, we focus on the notion of distance duality that arises when considering geometric operators together with edit distances. In section 3.7, we explain how the geometric interpretation of crossover greatly simplifies the relation between crossover and fitness landscape. In section 3.8, we show how the geometric definition of crossover can be used to design new crossovers for new representations. In section 3.9, we explain why geometric crossover works well on smooth landscapes by showing that evolvability and heritability of geometric crossover are linked with landscape smoothness. In section 3.10, we explain how problem knowledge and fitness landscape are related and how to embed problem knowledge in the search done by geometric operators.

3.2 Preliminary definitions

3.2.1 Search problem

Let S denote the *solution set*[1] comprising all the candidate solutions to a given *search problem* P. We assume S to be finite[2]. The members of this set must be seen as *formal solutions* not relaying on any specific underlying representation.

The goal of a search problem P is to find specific solutions in the search space that maximize (minimize) an *objective function*:

[1]We distinguish between solution set and solution space. The first refers to a collection of elements, while the second implies a structure over the elements.

[2]With due precautions, most of the concepts presented in this chapter can be extended to countably and uncountably infinite solution sets.

$$g : S \rightarrow R$$

Let us assume, without loss of generality, that the goal of the search problem P is to maximize g. The *global optima* x^* are points in S for which g is a maximum. We collect them in the set S^* where

$$x^* \in S^* \Leftrightarrow g(x^*) = \max_{x \in S} g(x).$$

Notice that global optima are well-defined when the objective function is well-defined and are independent of any structure defined on S. On the contrary, *local optima* are definable only when a structure over S is defined. A search problem in itself does not come with any predefined structure over the solution set.

3.2.2 Search space

Formally, a *metric space* (M, d) is a set M provided with a metric or distance d that is a real-valued map on $M \times M$ which fulfils the following axioms for all $s_1, s_2 \in M$:

1. $d(s_1, s_2) \geq 0$ and $d(s_1, s_2) = 0$ if and only is $s_1 = s_2$;

2. $d(s_1, s_2) = d(s_2, s_1)$, i.e., d is symmetric; and

3. $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$, i.e., d satisfies the triangular inequality.

A simple, connected graph (undirected and non-weighted) is naturally associated with a metric space on the set of its vertices. The distance between any two vertices is the length of the shortest path (number of edges) connecting these vertices. Distances arising from simple, connected graphs are known as *graphic distances* and are metrics [154]. They have additional special features besides respecting the metric axioms.

Similarly, a metric space can arise from a connected weighted graph [30] as follows: if $G = (V, E)$ is a graph and $w = (w_e)_{e \in E}$ are strictly positive weights assigned to its edges, one can define the path metric $d_{G,w}(i,j)$ of the weighted graph (G,w). Namely, for two nodes $i,j \in V$, $d_{G,w}(i,j)$ denotes the smallest value of $\sum_{e \in P} w_e$ where P is a path from i to j in G. In general, a metric space induced by a weighted graph is *non-graphic* and the same metric can arise from different weighted graphs.

Conversely, any finite metric space can be represented by means of a weighted graph, where the vertices of the graph represent the elements of the metric space and the weights on the edges denote the distance between elements connected by those edges. In general, there is more than one weighted-graph representation for the same metric space. Two of them are the *nearest-neighbors graph*, in which there is an edge between two nodes only if they are nearest neighbours according to the metric considered, and the all-pairs graph, which is a fully-connected graph explicitly reporting the distance between any pair of vertices. There are, however, many intermediate weighted-graph representations.

A simple, connected graph is naturally associated with a neighbourhood structure, as follows. Let $W = (V, E)$ be a simple, connected graph, where V is the set of vertices and E is the set of edges. The neighbourhood function $Nhd : V \to 2^V$ associated with W maps every vertex in V to the set of all its direct neighbour vertices in V directly linked to it by an edge. Since the graph W is undirected, the neighbourhood function Nhd is symmetric: $y \in Nhd(x) \Leftrightarrow x \in Nhd(y)$. Also, since the graph W is connected, a finite number of repeated applications of Nhd to any vertex x returns the set of all vertices V: $\exists n \forall x \in V : Nhd^n(x) = V$ where n is a finite positive integer and the application of Nhd to a set of vertices is interpreted as to return the union of the sets of direct neighbour vertices to all input vertices.

A *search space* is the solution set S endowed with a given metric d which defines a distance between solutions. We say that d is the structure of the search space. When d is a graphic distance, the search space endowed with d is naturally associated with a simple, connected graph in which each vertex represents a solution. In this case, the search space is also naturally associated with a neighbourhood function Nhd.

Notice that d is an arbitrary distance and need not have any particular connection or affinity with the search problem at hand. The same solution set can be associated with more than one distance giving rise to different search spaces for the same search problem.

3.2.3 Solution representation

The distance associated with a solution space is closely related to the solution representation chosen. In the following, we elicit this relation.

The solution set S is the set of formal solutions of a problem. In order to implement an algorithm to search S, we need to represent the formal solutions by means of syntactic configurations (syntactic objects or genotypes) which can actually be manipulated by the search operators to obtain offspring solutions from parent solutions.

A *representation mapping* is a function $r : C \to S$ associating any syntactic configuration in C with a formal solution in S. When this mapping is bijective, we refer to it as a *direct encoding*. In this case, C and S are only formally different. When the representation mapping is not injective, we refer to it as a *redundant encoding* in which case the size of the configurations set C is larger than the size of the solutions set S.

The configuration set C can be endowed with one or more distances that measure the syntactic dissimilarity between any two configurations. Edit distances are an important class of syntactic distances. The following general, formal definition of edit distance was presented by Cormode in his PhD thesis [22], in which he also showed that any edit distance is a metric.

Definition 3.2.1. An edit distance is a distance defined by a set of edit operations. Formally, the edit operations are defined by a symmetric relation R on the set of configurations C. The edit distance between two configurations is the minimum number of edit operations needed to transform one into the other. That is, $d(a, b)$ is the minimum n such that $R^n(a, b)$, and 0 if $a = b$.

Notice that for an edit distance to be a metric, the edit operations must be reversible and for any two configurations there must be a (finite) sequence of edit operations that transforms one configuration into the other. A configuration set can be associated with different sets of edit operations that give rise to different notions of edit distance for the same set.

Edit distances are graphic distances since these distances are associated with the simple, connected graph associated with the neighbourhood function defined as $\forall x \in C : Nhd(x) = \{y|R(x, y)\}$.

In this thesis, except for chapters 10 and 11, in which we consider redundant encodings, we will assume that the representation mapping between C and S is bijective, and that the distance defined on C extends naturally to S. We will, therefore, use interchangeably the terms solution space and configuration space.

3.2.4 Fitness landscape

A *fitness landscape* F is a triple (C, d, f) where (C, d) is a configuration space and $f : C \to R$ is a *fitness function* mapping a syntactic configuration to its *fitness value*. The fitness value is a real number. It may or may not coincide with the objective function value of the solution represented by the input genotype. For the sake of simplicity, we assume that it is. Therefore, the fitness function is the composition of the representation mapping r, which associates configurations to a formal solutions, with the objective function g, which associates formal solutions to their objective function values: $f = g \circ r$.

We can now define *local optima* of the landscape (C, d, f) as those configurations in C that have fitness equal or larger than those of all their nearest neighbours with respect with the distance d.

3.2.5 Balls and segments

In classical Euclidean geometry, the measure of the distance between two points in the plane, say A and B, is calculated using the well known formula: $d(A, B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$. This is certainly a very intuitive notion of distance. By redefining the distance function between two points one obtains a new geometry for each distance redefinition. One simple example is the so-called 1st order Minkowski distance: $d(A, B) = |x_A - x_B| + |y_A - y_B|$. This definition of distance is fairly natural: it is the minimum distance that a taxicab would need to travel to reach point B from point A, if all streets are only oriented vertically and horizontally. For this reason, this metric is often referred to as the Manhattan metric. Many geometric figures, like circles, ellipses, parabolas, are defined in terms of distance. For instance, a circle is just the set of points with a fixed distance to the centre. These, of course, look quite different if we use a non-Euclidean measure of distance.

If we go further and say that a shape corresponds to a particular definition independently from the specific notion of metric used, we are then dealing with *abstract shapes* that are defined axiomatically and present abstract geometric properties that are shape-specific but not distance-specific. These abstract shapes are studied in *metric geometry.* Two of them, balls and segments, turn out to be very useful to define abstractly mutation and crossover and in the following we consider them in more detail.

In a metric space (S, d) a *closed ball* is the set of the form

$$B_d(x; r) = \{y \in S | d(x, y) \leq r\},$$

where r is a positive real number called the *radius* of the ball. A *line segment* (or closed interval) is the set of the form

$$[x; y]_d = \{z \in S | d(x, z) + d(z, y) = d(x, y)\},$$

where $x, y \in S$ are called *extremes* of the segment. Note that $[x; y]_d = [y; x]_d$. The *length* l of the segment $[x; y]_d$ is the distance between a pair of extremes $l([x; y]_d) = d(x, y)$. Let H be a segment and $x \in H$ an extreme of H, there exists only one point $y \in H$, its conjugate extreme, such that $[x; y]_d = H$. Examples of balls and segments for different spaces are shown in Figure 3.1. Note how the same set can have different geometries (see Euclidean and Manhattan spaces) and how segments can have more than one pair of extremes. For instance, in the Hamming space, a segment coincides with a hypercube and the number of extremes varies with the length of the segment, while in the Manhattan space, a segment is a rectangle and it has two pairs of extremes. Also, a segment is not necessarily "slim", that is, it may include points that are not on the boundaries. Finally, a segment does not coincide with a shortest path connecting its extremes (*geodesic*). In general, there may be more than one geodesic connecting two extremes.

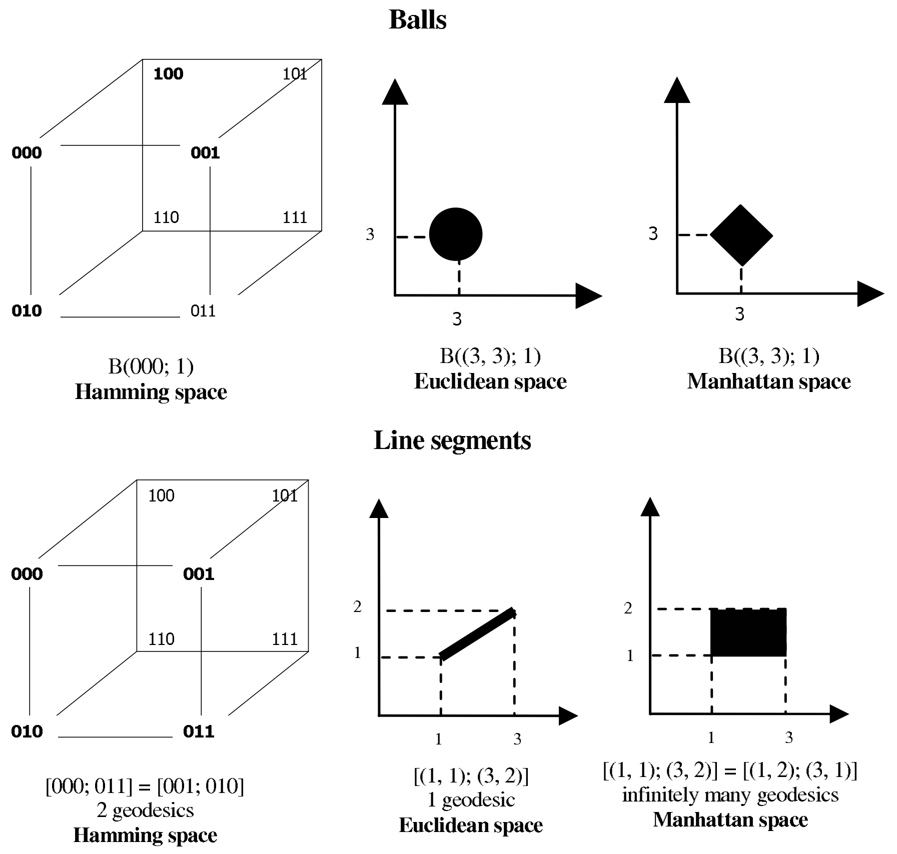


Figure 3.1: Examples of balls and segments

3.3 Geometric genetic operators

In this section, we *define*, postponing the justifications of these definitions to the following sections, two *classes* of operators in the landscape (i.e., using the notion of *distance* coming with the landscape): geometric mutation and geometric crossover. Within these classes, we identify two specific operators: geometric uniform mutation and geometric uniform crossover. *These definitions are representation-independent and, therefore, the operators are well-defined for any representation.*

A k-ary genetic operator OP takes k parents $p_1, p_2, \ldots, p_k \in C$ and produces one offspring $c \in C$ with probability

$$Pr\{OP(p_1, \dots, p_k) = c\} = f_{OP}(c|p_1, \dots, p_k).$$

Mutation is a unary operator while *crossover* is typically a binary operator.

Definition 3.3.1. The *image set* of a genetic operator OP is the set of all possible offspring produced with non-zero probability by OP when the parents are $p_1, p_2, \dots, p_k \in C$:

$$Im[OP(p_1, p_2, \dots, p_k)] = \{c \in S | f_{OP}(c|p_1, p_2, \dots, p_k) > 0\}$$

Notice that the image set is a *mapping* from a vector of parents to a set of offspring.

Definition 3.3.2. A unary operator M is a *geometric ε-mutation* operator on the search space (C, d), if $Im[M(p)] \subseteq B_d(p; \varepsilon)$ where ε is the smallest real for which this condition holds true.

In other words, in a geometric ε-mutation all offspring are at most ε *away* from their parent. Notice that all mutations are geometric on any search space for some ε.

Definition 3.3.3. A binary operator CX is a *geometric crossover* on the search space (C, d) if $Im[CX(p_1, p_2)] \subseteq [p_1; p_2]_d$.

This simply means that in a geometric crossover offspring lay *between* parents. We use the term *recombination* as a synonym of any binary genetic operator.

We now introduce two *specific* operators belonging to the *families* defined above.

Definition 3.3.4. *Geometric uniform ε-mutation UM* is a geometric ε-mutation on the search space (C, d) where all z at most ε away from parent x have the same probability of being the offspring:

$$f_{UM}(z|x) = \frac{\delta(z \in B_d(x, \varepsilon))}{|B_d(x, \varepsilon)|}$$

$$Im[UM(x)] = \{z \in S | f_{UM}(z|x) > 0\} = B_d(x, \varepsilon)$$

where δ is a function which returns 1 if the argument is true, 0 otherwise.

When ε is not specified, we mean $\varepsilon = 1$.

Definition 3.3.5. *Geometric uniform crossover UX* is a geometric crossover on the search space (C, d) where all z laying between parents x and y have the same probability of being the offspring:

$$f_{UX}(z|x, y) = \frac{\delta(z \in [x; y]_d)}{|[x; y]_d|}$$

$$Im[UX(x, y)] = \{z \in S | f_{UX}(z|x, y) > 0\} = [x; y]_d.$$

Theorem 3.3.1. *When d is a graphic distance function, the configuration space (C, d) determines uniquely, and is uniquely determined by any of the following:*

1. *The neighborhood function Nhd associated with the space (C, d),*
2. *The neighborhood graph $W = (V, E)$ associated with the space (C, d),*
3. *The probability distribution function $f_{UM}(z|x)$ of the offspring of the uniform geometric mutation UM on the space (C, d),*
4. *The probability distribution function $f_{UX}(z|x, y)$ of the offspring of the uniform geometric crossover UX on the space (C, d),*
5. *The set of all balls $\boldsymbol{B}$ on the space (C, d),*
6. *The set of all segments $\boldsymbol{H}$ on the space (C, d).*

Proof. Equivalences 1 and 2 are trivial consequences of the fitness landscape definition.

Equivalence 3: given UM one has its conditional density function $f_{UM}(z|x)$ and, consequently, the image set mapping $Im[UM(x)]$, i.e., the mapping $x \mapsto B(x, 1)$. The structure of the space is, therefore, given by $Nhd : x \mapsto (B(x, 1) \setminus \{x\})$.

Equivalence 4: analogously, given UX, one has the mapping $(x, y) \mapsto [x; y]$. By restricting this mapping through its co-domain considering only segments of size 2, the corresponding restricted domain coincides with the set of edges E of the neighborhood graph. Hence, the structure of the space is determined.

Equivalence 5: the relation of inclusion between sets $\subseteq$ induces a partial order on $\mathbf{B}$. The set of all balls of radius 1, $\mathbf{B_1}$, can be determined by considering all those balls in $\mathbf{B}$ that have, as only predecessors, balls of size 1 (i.e., balls of radius zero). Given a ball $b \in \mathbf{B_1}$ a point $x \in b$ is the center of the ball if and only if $\forall x' \in (b \setminus x) \exists b' \in \mathbf{B_1} : b \neq b' \wedge x, x' \in b'$ [3]. Knowing the center of each ball of radius 1, it is possible to form the map $x \mapsto B(x, 1)$ and proceed as in equivalence 4.

Equivalence 6: by considering only segments in $\mathbf{H}$ of size 2, one can form the set E of the edges of the neighborhood graph. Hence, the structure of the space is determined. □

A direct and important consequence of theorem 3.3.1 is that, for geometric operators based on graphic metric spaces, the set of all uniform geometric mutations and the set of all uniform geometric crossovers are in 1-to-1 correspondence.

In chapter 7, we will present the more complex relationship between search spaces and geometric operators based on non-graphic metric spaces.

[3]Given two different points in the same ball of radius 1, $x, x' \in b$, they are either at distance 1 or distance 2. If they are at a distance 2, b is the only ball in $\mathbf{B_1}$ satisfying this condition since the two points are extremes of a diameter of the ball b and identify the ball univocally. If they are at a distance 1, there must exist at least two balls in $\mathbf{B_1}$ containing x, x' one in which x is the center and x' is not, and another one in which the roles are reversed. This symmetry holds because the neighborhood is symmetric.

3.4 Generalization of binary string crossover

Given two binary strings $s_1 = (x_1, \dots, x_n)$ and $s_2 = (y_1, \dots, y_n)$ of length n, the Hamming distance $d_H(s_1, s_2)$ is the number of places in which the two strings differ:

$$d_H(s_1, s_2) = \sum_{i=1}^{n} \delta(x_i \neq y_i).$$

A *property* of the Hamming distance is that a binary string $s_3 = (z_1, \dots, z_n)$ lays between s_1 and s_2 if and only if every bit of s_3 equals at least one of the corresponding bits of s_1 and s_2, i.e., $\forall i : z_i \in \{x_i, y_i\} \Leftrightarrow s_3 \in [s_1; s_2]_{d_H}$.

Traditional (one-point, two-point, uniform, etc.) crossovers for binary strings belong to the class of mask-based crossover operators [153]. A crossover operator is a probabilistic mapping $cx_m : S \times S \rightarrow_m S$ where the mask m is a random variable with different probability distributions for different crossover operators. The mask m takes the form of a binary string of length n that specifies for each position from which parent to copy the corresponding bit to the offspring, i.e., $cx_m(s_1, s_2) = s_3$ and $m = (m_1, \dots, m_n)$ then $z_i = x_i \cdot \delta(m_i = 0) + y_i \cdot \delta(m_i = 1)$.

Theorem 3.4.1. *All mask-based crossover operators for binary strings are geometric crossovers.*

Proof. We need to show that for any probability distribution over m $Im[cx_m(s_1, s_2)] \subseteq [s_1, s_2]$ holds. Out of all possible mask-based crossovers, those with a non-zero probability of using all the 2^n masks produce the biggest image set for any given pair of parents. Formally, this is given by $Im[cx(s_1, s_2)] = \{cx_m(s_1, s_2) | m \in \{0,1\}^n\}$. So, it is sufficient to prove that $Im[cx(s_1, s_2)] \subseteq [s_1, s_2]$ for this image set. This is equivalent to prove that $\forall m \in \{0,1\}^n : s_3 = cx_m(s_1, s_2) \rightarrow s_3 \in [s_1, s_2]$.

Given $s_1 = (x_1, \dots, x_n), s_2 = (y_1, \dots, y_n)$, and any mask m, there exists a unique $s_3 = (z_1, \dots, z_n)$. From the definition of mask-based crossover it follows that $\forall i : z_i \in \{x_i, y_i\}$. Then, from the Hamming distance property mentioned above, it follows that $\forall m : s_1 \in [s_1, s_2]$, which completes the proof of the first part of the theorem.

□

Definition 3.4.1. The *zero-or-one-bit mutation* is an operator where a string of size n is either mutated by flipping one bit or is not mutated. For each bit, the probability of being mutated is $\frac{1}{n+1}$, so is the probability of not mutating any bit.

Theorem 3.4.2. *The geometric uniform crossover for the configuration space of binary strings endowed with Hamming distance is the traditional uniform crossover. The geometric uniform 1-mutation for the configuration space of binary strings endowed with Hamming distance is equivalent to a zero-or-one-bit mutation.*

Proof. Let us start by proving that the image sets of traditional uniform crossover and geometric uniform crossover coincide. We need to show that $Im[cx(s_1, s_2)] = [s_1, s_2]$, where $Im[cx(s_1, s_2)]$ was defined in the proof of theorem 2, from which we know that $Im[cx(s_1, s_2)] \subseteq [s_1, s_2]$. Consequently, all we need to prove is that $\forall s_3 \in [s_1, s_2] \Rightarrow \exists m \in B^n : cx_m(s_1, s_2) = s_3$. For the Hamming distance property this is equivalent to say $\forall s_3 \forall i : z_i \in \{x_i, y_i\} \Rightarrow \exists m \in B^n : cx_m(s_1, s_2) = s_3$, where z_i are the bits of s_3. From the definition of crossover this is equivalent to proving that $\forall s_3 \forall i : z_i \in \{x_i, y_i\} \Rightarrow \exists m \in B^n : z_i = x_i \cdot \delta(m_i = 0) + y_i \cdot \delta(m_i = 1)$. This is true because, when the bits in the parents differ, it always exists a mask that specifies the parent in which the bit equals the offspring bit. If the bits do not differ, the mask indifferently specifies one parent or the other for that bit. This shows that the image sets of traditional uniform crossover and geometric uniform crossover coincide.

Every element of the image set of the traditional uniform crossover has identical probability of being the offspring [161] and the same is true for the elements of the image set of the geometric uniform crossover (by definition). This completes the proof of the first part of this theorem.

The image sets of zero-or-one-bit mutation and geometric 1-mutation coincide as it is trivial to see by noting that the Hamming ball of radius 1, which is the image set of geometric 1-mutation, coincides with the image set of the zero-or-one-bit mutation. Every element of the image set of zero-or-one-bit mutation has identical probability of being the offspring and the same is true for the elements of the image set of the geometric uniform 1-mutation (by definition). □

3.5 Properties of geometric crossover

In the following, we highlight the properties of the class of geometric crossovers:

1. *Generalization*: geometric crossover is a generalization of the family of crossovers based on masks for binary representation (section 3.4) and it captures and generalizes the distinction between mask-based crossover for binary strings and other forms of recombination (see chapter 14).

2. *Unification*: as we will show in the next chapters a variety of operators developed for other important representations, such as real-valued vectors, permutations and syntax trees, fit our geometric definitions given suitable notions of distance (naturally not *all* pre-existing operators do this, but many do). Hence, geometric crossover has the potential to lead to a unification of the different evolutionary algorithms.

3. *Representation independence*: as we discussed in chapter 2, evolutionary computation theory is fragmented. One reason for this is that there is not a unified way to deal with

different solution representation (although steps forward in this direction have recently been made [79] [150]), which has led to the development of significantly different theories for different representations. In this context, one important theoretical implication of our geometric definitions is that the genetic operators are fully defined without any direct reference to the representation. This may pave the way to a unified treatment of evolutionary theory (see chapter 15).

4. *Clarification*: the connections between operators, representation, distance, neighborhood and fitness landscape are completely clear when using geometric operators. We discuss this in section 3.7.

5. *Geometric interpretation*: an evolutionary algorithm using geometric operators does a form of geometric search based on segments (crossover) and balls (mutation). This suggests looking at the solution space not as a graph or hyper-graph, as normally done, but rather as a geometric space. The notion of distance arising from the syntax of configurations reveals, therefore, a natural *dual* interpretation:[4] (i) it is a measure of similarity (or dissimilarity) between two syntactic objects; (ii) and it is a measure of spatial remoteness between points in a geometric space. We explain this duality in section 3.6.

6. *Principled design*: One important practical implication of the geometric definition of crossover is the possibility of doing crossover principled design. When applying evolutionary algorithms to optimization problems, a domain specific solution representation is often the most effective [26] [128]. However, for any non-standard representation, it is not always clear how to define a good crossover operator. Given a good neighborhood structure for a problem, all meta-heuristics defined over such a structure tend to be good. Indeed, the most important step in using a meta-heuristic is the definition of a good neighborhood structure for the problem at hand [48]. With geometric crossover, given a good neighborhood structure or a good mutation operator, a crossover operator that respects such a

[4]Any mathematical object/property that admits a definition only based on the concept of distance possesses a dual nature: a syntactic one and a geometric one.

structure is automatically defined. We give an example of crossover design in section 3.8.

7. *Landscape and knowledge*: the landscape structure is relevant to a search method only when the move operators used in that search method are strongly related to those which induce the neighborhood structure used to define the landscape [64]. This is certainly the case for the geometric operators. The problem knowledge used by an evolutionary algorithm that uses geometric operators is embedded in the connectivity structure of the landscape. The landscape is, therefore, a *knowledge interface* between a *formal problem* and a *formal search algorithm* that has no knowledge of the problem whatsoever. In order for the knowledge to be transmissible from the problem to the search algorithm *through the landscape*, there are two requirements: (i) the search operators have to be defined over the connectivity structure of the landscape (i.e., using a distance function); (ii) the landscape has to be *designed* around the specific definitions of the operators employed in such a way as to bias the search towards good areas of the search space so as to perform better than random search. We explain the relation between problem knowledge and fitness landscape in section 3.10.

8. *Landscape conditions*: for the no free lunch theorem [165], over all the problems, on average any search algorithm performs the same as random search. So, in itself, a given search algorithm (any meta-heuristics) is not inherently superior to any other. A search algorithm, therefore, to be of use, has to specify the class of problems for which it works better than random search. The geometric definition of mutation (connected with the concept of ball) and the geometric definition of crossover (connected with the concept of segment) are simple and general and give an insight on what types of general conditions over the landscape may make them superior to random search: some forms of *continuity* and *convexity*, respectively. What the exact and most general conditions are is an open question that we will start to address in section 3.9 and discuss further in the concluding chapter of this thesis. Continuity and convexity, in various guises, are well-known desirable conditions of fitness landscapes for the most common neighbourhood-based meta-heuristics [160][48].

They are also important to guarantee good performance in many other traditional optimisation techniques [122]. Hence, ensuring them should guide the landscape design for the geometric operators.

3.6 Distance duality

Descartes [28] introduced the idea of *duality* in geometry by drawing a one-to-one correspondence between (vectors of) real numbers to points of the Euclidean space through the notion of coordinates. This duality taught at school to everyone, the Cartesian coordinate system, seems so obvious and natural nowadays that its importance is somehow invisible to the most. The remarkable thing of the Cartesian duality is that it connects two different worlds, numbers and geometry, allowing us to manipulate algebraically geometric shapes, on one hand, and solving equations graphically, on the other. This duality is the basis of analytic geometry and its successor, calculus.

We introduce a generalization of Cartesian duality drawing a correspondence between syntactic objects, that generalize the notion of real vectors, and nodes of the neighbourhood structure, that generalize the notion of points in the Euclidean space to that of points in discrete spaces. Traditional geometric shapes, trajectories and geometric transformations of shapes in the Euclidean space, that are defined solely in terms of distances, can be readily generalized, by distance redefinition, to discrete spaces associated with edit distances. We refer to these geometries as *edit distance geometries.*

The edit distance is a measure of syntactic dissimilarity between two syntactic objects. Every syntactic object uniquely represents a point in the space induced by the syntax and the notion of edit move considered. The induced space naturally inherits the edit distance from the syntax. However, in the transition the distance loses its meaning of dissimilarity measure and becomes a measure of spatial remoteness between points of the space. When one uses this distance as base for metric geometry, one actually uses a measure of syntactic similarity as if it were a notion of spatial remoteness. This creates the dual bond between syntax and space. So, any geometric definition based on edit distance has also a syntactic meaning.

For example, the definition of a ball of radius one corresponds to all points that are in space within the radius (geometric interpretation) and also correspond to all syntactically related configurations that differ in their syntax of an edit unit from a given one (syntactic interpretation). This natural duality of edit distance geometries is important because it allows us to use instruments and ideas from geometry to reason about syntactic objects and *vice versa*. We call this property *distance duality*. Distance duality is a never-ending source of surprises because for each choice of syntax and edit move the syntactic interpretation of the same geometric definition is different and peculiar of the specific syntax and move considered. This is the same type of surprise that arises when one draws a ball using alternative notions of distance for real vectors, say Euclidean and Manhattan, obtaining respectively a rounded-ball and a diamond-ball.

This has powerful consequences for evolutionary algorithms. One important consequence of the distance duality is that the geometric definition of crossover, being also a syntactic definition, tells exactly what crossover is for any choice of syntax and edit move (crossover principled design). Another important consequence of the duality is that recombination operators for different representations, admit the same geometric definition (unification). In the following chapters, we will study closely the effects of the distance duality for the geometric definitions of ball and segment for a few solution representations.

3.7 Crossover landscape

Whereas local search algorithms have a natural interpretation in the fitness landscape, evolutionary algorithms are more problematic. Local search algorithms can be seen as following a path in the landscape from the initial solution to a peak close to the initial point. One of the characteristics that makes the interpretation of evolutionary algorithms in the fitness landscape problematic is the presence of the crossover operator. A point mutation can be thought as a random step of local search. Hence, mutation is intuitively associated with the neighbourhood structure of the search space. Recombination, differently from mutation, is an operator working on two parent solutions and producing one or two offspring solutions. Recombination seems not to be associable with a simple fitness landscape because of its more complex relationship between parents and offspring.

The notion of fitness landscape is useful if the search operators employed are connected or matched with the landscape: the greater the connection the more landscape properties mirror search properties[64]. *A fitness landscape can be seen as a function of the search operator* employed which induces the neighbourhood structure of the associated fitness landscape[135]. A mutation operator simply assigns a set of "accessible neighbours" (offspring) to each configuration (parent). So, the neighbourhood structure is a (possibly directed) graph with weighted arcs indicating the transition probabilities from the parent to a specific offspring. The most immediate consequence of the fact that recombination acts on two arguments is that the recombination induced configuration space cannot be represented as a simple graph with the set of genotypes representing the set of vertices. This leaves two alternatives: one can change the nature of the vertex set and have pairs of types as vertices of a directed graph [24] [64]. The alternative is to leave the vertices to represent individual genotypes and to make edges more complex, leading to hyper-graphs [46]. Both these approaches lead to complex structures of the search space associated with recombination operators that, unlike for the case of mutation, do not allow us to develop an understanding of how recombination operators relate with its fitness landscape. Another important drawback of both approaches is that different operators are associated with different fitness landscapes, hence making it impossible to directly compare different search operators and understand their interactions.

Geometric crossover and geometric mutation are based on the distance associated with the search space, seen as a metric space, and on the geometric notions of ball and line segment. This approach is the dual of the traditional approach: *we see genetic operators as functions of the search space*[5]. So, mutation and crossover share the same neighbourhood structure. Seeing

[5]This may appear paradoxical because a search space arises from the search operator that induces it. So, we need a search operator to induce a structure on the search space in the first place: how can we then define a search operator on its own search space? We answer this question in chapter 14.

search operators as functions of the search space greatly simplifies the relationship between search operators and fitness landscape and allows different search operators to share the same search space so clarifying their roles. In chapter 15, we give a general characterization of how the search of evolutionary algorithms with geometric crossover relates with its fitness landscape.

3.8 Crossover principled design

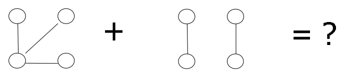
In this section, we give an example of principled design of crossover. Then, we discuss commonalities and differences between geometric crossover and path-relinking, which is a meta-heuristic that combines solutions in the neighbourhood structure.

3.8.1 Example of crossover principled design

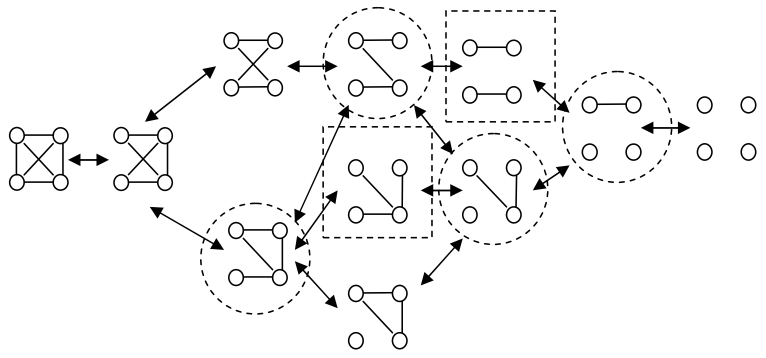
An example of principled design of crossover is shown in Figure 3.2, where we assume that we want to evolve graphs with four nodes and we are given a mutation operator for such graphs that either adds or removes exactly one edge. We want to define a good crossover operator that would, for example, produce meaningful offspring when applied to the parent graphs in Figure 3.2(a). The configuration space for this problem is shown in Figure 3.2(b). The parent graphs are boxed while the graphs belonging to the segment defined by the parents are encircled. With our definition of geometric crossover these are all possible successors, as shown in Figure 3.2(c).

In order to derive an operational definition of crossover for graphs that tells us how to manipulate the syntax of the parents to obtain the offspring in the segment between them, we rely on the duality of the edit distance for graphs based on the edge insertion/deletion move. For this edit distance, the geometric notion of moving along a geodesic between points a and b (shortest path) corresponds, syntactically, to transforming the graph that represents point a into the graph that represents point b by a shortest sequence of edge insertion/deletion moves. The offspring of a and b are, therefore, all the intermediate graphs obtained by transforming parent b into parent a adding and removing the minimum amount of edges. Consequently, the offspring of a and b can be also obtained by (i) matching the vertices of the parent graphs a and

(a)



(b)



(c)

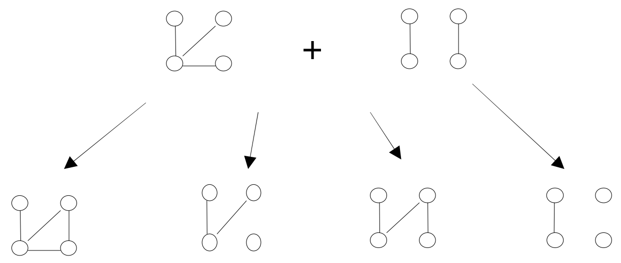


Figure 3.2: Inducing crossover from mutation (see text).

b such that to minimize the number of differences of corresponding edges (best inexact graph matching), and then, (ii) recombining the structures of the graphs so aligned (for example, using a randomly generated crossover mask covering all pairs of vertices). Figure 3.3 shows, on the left, the best inexact graph matching of the two parent graphs, and on the right, the alignment of the parents where solid edges denote edges common to both parent graphs, and broken edges denote edges that belong to a single parent graph. So, the offspring graphs can be obtained by recombining edges between the aligned parent graphs at the locations identified by the broken edges.

3.8.2 Connection with path-relinking

Geometric crossover presents strong connections with the meta-heuristics path-relinking. In the following we briefly describe path-relinking, then we outline the main commonalities and

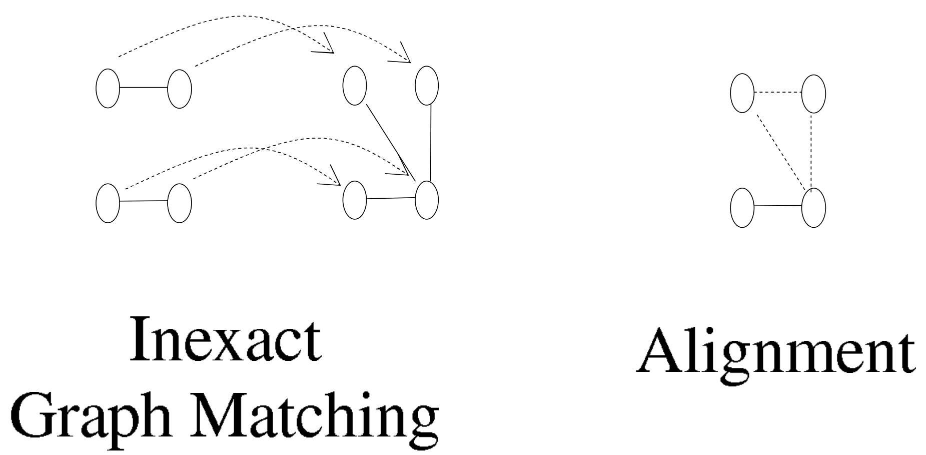


Figure 3.3: Geometric crossover for graphs.

differences between path-relinking and geometric crossover.

Path-relinking

Path-relinking [47] is an evolutionary algorithm which differs from other evolutionary procedures by providing unifying principles for joining solutions based on path constructions in both Euclidean and neighborhood spaces, and by utilizing deterministic strategies where other approaches resort to randomization. Path-relinking is more abstract than other evolutionary algorithms in that it does not deal directly with solution representations, and how to manipulate them, which is left as implementation detail.

For path-relinking based on solutions in the Euclidean space, new solutions are created by building linear combinations of parent solutions using both positive and negative weights. This means that trial solutions can be both, inside and outside the convex region spanned by the parent solutions. The concept of combining solutions by making linear combinations of parent solutions is generalized to neighbourhood spaces. Linear combination of points in the Euclidean space can be re-interpreted as path between and beyond solutions in a neighbourhood space. To generate the desired paths, it is necessary to select moves that satisfy the following condition: upon starting from an initiating solution, the moves must progressively introduce "attributes" contributed by a guiding solution. Notice that the path between two parent solutions is not normally a shortest path between them, but simply a path from a parent towards the other parent. Also, the solutions on this path are generally not feasible and require a repairing mechanism to be applied.

Differences between path-relinking and geometric crossover

Like path-relinking, geometric crossover connects parents and offspring via the search space rather than directly specifying how to manipulate the syntax of the specific solution representations. Also, in both recombination operators offspring are in paths connecting the parents in the search space. However there are important differences between path-relinking and geometric crossover: *geometric crossover can be understood as a formalized generalization (to metric spaces) of path-relinking that elicits the dual relationship between distance and solution representation so giving a formal recipe to design new crossover operators, on one hand, and making it possible to build a general, representation-independent theory of evolutionary algorithms, on the other.* We consider the differences between geometric crossover and path-relinking more into detail in the following.

- *Purpose*: path-relinking was developed, by researchers guided by a practical interest, as a general heuristic to combine solutions in the neighbourhood structure. Geometric crossover arises from the desire of giving a formal definition of crossover that encompasses all solution representations, hence leading towards a theoretical unified framework encompassing all evolutionary algorithms.

- *Theory*: whereas path-relinking is a heuristic method for which no theory is available, geometric crossover is formally defined and it is the very foundation for a general theory.

- *Generalization*: whereas path-relinking is linked with the neighbourhood structure of the search space, geometric crossover is defined for general *metric spaces*. Geometric crossover is, therefore, linked with the structure of the search space via the distance associated with the search space. This allows us for a natural generalization of geometric crossover from distances arising from neighbourhood graphs (graphic distances) to general distances. This is important because natural distances associated with variable-length solution representations, such as trees and sequences, are non-graphic (see chapters 7 and 8).

- *Duality elicitation*: On one hand, path-relinking which combines solutions in the neighbourhood structure is understood to be a recombination operator very different from recombination operators defined directly on the solution representations [47]. On the other hand, recombination operators operating directly on solutions representations were believed not to be associable with simple neighbourhood structures (see section 3.7). Geometric crossover shows us that both these beliefs are wrong. In fact there is no difference between the recombination operators defined directly on the solution representations and the combination of solutions in the neighbourhood structure. This equivalence is rooted in the duality between the syntax of solution representation and the geometry of the search space (section 3.6). As we have seen in the example of geometric crossover for graphs (section 3.8.1), picking offspring in the segments between parents graphs is equivalent to obtaining offspring graphs by recombining structurally aligned parent graphs.

- *Formal crossover design*: whereas path-relinking *suggests* how to navigate the neighbourhood structure to locate the offspring solutions of parent solutions in the neighbourhood structure, geometric crossover can be used to actually design crossover operators that manipulate the syntax of the parent solutions to obtain the offspring solutions for any solution representation. These are complementary ways of performing the same search. Importantly, the definition of geometric crossover can be used as a *formal* recipe to build new crossovers because it defines *exactly* what crossover is for any representations without the need of arbitrary adaptation by the crossover designer to the specific representation.

3.9 Why geometric crossover works well on smooth landscapes

In this section, we give an *initial explanation* of the reason why geometric crossover works well on smooth fitness landscapes, so justifying both the definition of geometric crossover and the requirement of smoothness of the landscape. In this thesis, we do not intend to make the following argument more rigorous or formal, preferring to concentrate on representation and geometry issues, and leaving as important future work to define exactly what "smooth landscape" is and derive a formal relation linking it with the performance of an evolutionary algorithm with geometric crossover. Nevertheless, in this thesis, we will use this argument as a rule-of-thumb to choose (or design) "good" geometric crossovers for specific problems (see, for example, chapter 5 on permutations), or to tell whether (analyze) a given recombination operator, which is known to be geometric under a certain distance, is a "good" operator for the specific problem at hand (see, for example, chapter 8 on sequences). In chapter 15, when the search performed by an evolutionary algorithm with geometric crossover will be presented, we will be able to tell something more about the required conditions of the fitness landscape that make geometric crossover to perform well.

Intuitively, on completely random landscapes, for which the distance between solutions bear no information about the relationship between their fitness values, an evolutionary algorithm with geometric operators, on average, does not outperform random search. This is because the geometric operators, that relate parents and offspring via the distance of the landscape only, cannot link, indirectly via distance, the fitness of the offspring with the fitness of their parents. Therefore, the fitness of parents and the fitness of their offspring are independent random variables, so turning the evolutionary algorithm into a sequence of independent random trials (with respect to the fitness), or random search.

In the following, we show how an evolutionary algorithm with geometric operators can work better than random search on a smooth landscape. Or in other words, we show how the smoothness of the landscape is turned into performance by an evolutionary algorithm with geometric operators. To do this, we need to explain the relation between evolvability and heritability of a search operator, first, and then the relation between heritability and smoothness of the fitness landscape for geometric search operators. This will allow us to link the smoothness of the landscape with the evolvability of the geometric operators.

3.9.1 Evolvability and heritability

The performance of pure random search does not depend on the connectivity structure of the fitness landscape (on the distance associated with it). It depends only on the shape of the fitness distribution associated with the fitness function of the problem. The performance of random search is, therefore, independent from the particular choice of fitness landscape done by the designer and it is a measure of inherent difficulty of the problem (instance) at hand.

Interesting real-world problems are characterized by the fact that increasingly better solutions are rarer and rarer. So, a typical fitness distribution of an interesting real-world problem is bell-shaped. The performance of random search on this type of problems is poor because getting better and better solutions by random sampling becomes harder and harder: the probability of hitting a solution with fitness larger than a given fitness threshold t is the area of the right tail of the fitness distribution after the value t. So, starting from the average fitness, as t increases, the area of the tail after t decreases more and more quickly. In the following, we will assume that the fitness distribution of the problem under consideration is Gaussian with mean zero and variance one.

Evolvability of a genetic operator was defined by Altenberg [3] as the probability of producing offspring with increased fitness with respect to their parents' fitness. Evolvability is a fundamental concept because it is what ultimately allows a genetic operator to contribute to the performance of an evolutionary algorithm: genetic operators with higher evolvability produce better performance.

The evolvability of a search operator OP, which takes a parent x in input and produces an offspring y according to the probability distribution $g_{OP}(y|x)$, is the probability that the fitness f_y of the offspring is larger than the fitness f_x of the parent. Notice that, in general, the evolvability of a search operator is a conditional probability and it is a function of the parent x. We will assume, in the following, that evolvability does not directly depend on x, but it depends only on its fitness f_x. So, if two parents have the same fitness they also have the same evolvability.

Random search can be seen as a genetic operator in which offspring are drawn at random in the solution set independently from their parents. The evolvability of the random search operator is the probability that the fitness f_y of the offspring y, drawn at random in the solution set independently from its parent x, is larger than the fitness f_x of the parent x. The evolvability of y is, therefore, the area of the right tail of the fitness distribution of the problem after the value f_x because the fitness distribution of f_y corresponds, in this case, with the fitness distribution of the problem. So, the evolvability of random search is function of the fitness of the parents: the higher the fitness of the parents, the smaller and smaller the evolvability of random search.

Evolvability is connected with the notion of heritability of a genetic operator. Heritability is a central concept in quantitative genetics that was introduced to quantify the level of predictability of passage of a biologically interesting phenotypic trait, such as fitness, from parent to offspring. It was introduced to the field of evolutionary computation under the name of parent-offspring correlation[160]. Heritability allows to estimate the fitness of the offspring given the fitness of the parents. In quantitative genetics, this estimation is known as response to selection and it was introduced to the field of evolutionary computation as theory of the breeder genetic algorithm[114].

In the following, we illustrate the connection between evolvability and heritability of a genetic operator. The heritability h of a genetic operator is defined as the correlation between the fitness of the parents and the fitness of their offspring obtained by applying the genetic operator. Let us consider a search operator whose joint fitness distribution of the parent and the offspring is jointly Gaussian. The marginal fitness distributions of parents and offspring equal the fitness distribution of the problem, which we assumed earlier to be Gaussian with mean zero and variance one. Under these conditions, we can estimate the fitness distribution of the offspring for any given parental fitness by conditioning on the fitness of the parent. The fitness distribution of the offspring is a gaussian with mean $E\{f_y|f_x\} = f_x \cdot h$ and variance $Var\{f_y|f_x\} = 1 - h^2$. In fact, this is equivalent to using a simple linear regression model to do the estimate of the fitness (distribution) of the offspring. The fitness distribution of the offspring is illustrated in figure

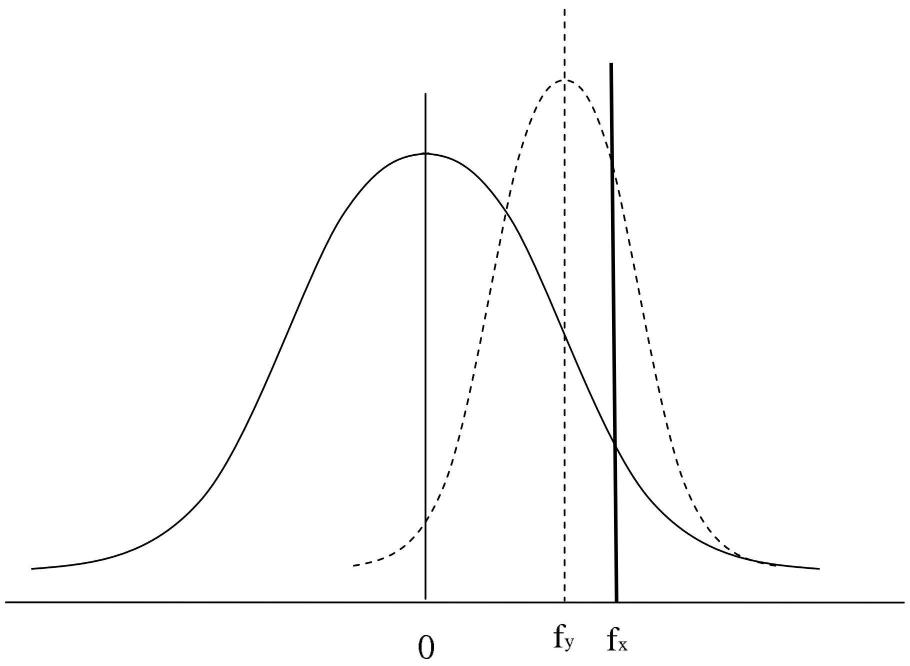


Figure 3.4: Fitness distribution of the offspring conditional to the fitness of the parent.

3.4. The solid gaussian is the fitness distribution of the problem. The solid thin vertical line indicates the mean fitness of the problem (0). The dashed gaussian is the fitness distribution of the offspring. The dashed vertical line indicates the expected fitness f_y of the offspring given the fitness f_x of the parent. The solid thick vertical line indicates the above-average fitness f_x of the parent.

There are a number of interesting facts about the fitness distribution of the offspring:

- the expected fitness of the offspring is tied to the fitness of the parents: the higher the parental fitness, the higher the expected fitness of the offspring (for parental fitness above average). Also, the higher the heritability, the closer the expected fitness of the offspring to the parental fitness.

- the expected fitness of the offspring is always lower than the parental fitness (for any positive heritability and for parental fitness above average). This is the well-known regression-toward-the-mean effect: above average parents produce, on average, above average offspring but closer to the average.

- the variability of fitness distribution of the offspring (variance) does not depend on the parental fitness, but it depends only on the heritability: the higher the heritability, the smaller the variability.
- the variability of the fitness distribution of the offspring is always smaller than the variability of the fitness distribution associated with the problem.

So, an increasing, positive heritability (i) keeps the expected fitness of the offspring of above average parents, above average and closer to the parental fitness, and (ii) reduces the variability of the fitness distribution of the offspring.

Heritability and evolvability relate as follows. Evolvability is the probability of the fitness of the offspring being higher than the fitness of the parents. So, the evolvability of the genetic operator is the area of the right tail of the fitness distribution of the offspring after the fitness of the parent. If we normalize the fitness of the parent with respect to the fitness distribution of the offspring we have that the evolvability of the operator is the area of the tail of the normal distribution after the normalized fitness of the parent.

The normalized fitness of the parent is

$$\hat{f}_x = \frac{f_x - E\{f_y|f_x\}}{\sqrt{Var\{f_y|f_x\}}} = f_x \cdot \sqrt{\frac{1-h}{1+h}}$$

for $h \neq 1$. The term $\sqrt{\frac{1-h}{1+h}}$ is a decreasing function for $h \in [0,1]$ and returns 1 when $h = 0$ and 0 when $h = 1$. So, for $h \in [0,1]$ this function returns always values in $[0,1]$.

Evolvability is a decreasing function of $\hat{f}_x$ because the area of the right tail of the normal distribution after $\hat{f}_x$ reduces for increasing $\hat{f}_x$.

The normalized fitness $\hat{f}_x$ is an increasing function of f_x. So, for h constant, the evolvability of the genetic operator considered decreases as the parental fitness increases. This is analogous to the case of the evolvability for the random search operator.

The normalized fitness $\hat{f}_x$ is a decreasing function of h. So, for f_x constant, *the evolvability of the genetic operator considered increases as the heritability of the genetic operator increases.* This is a simple and important relation between evolvability and heritability because one can improve evolvability, hence performance, by increasing heritability of a genetic operator. This cannot be done directly. However, how we well see in the following two sections, the heritability can be controlled indirectly by choosing search operators that give rise to a smooth fitness landscape.

For the extreme case $h = 0$, we obtain a search operator without heritability, for which $\hat{f}_x = f_x$. This is the case for the random search operator, in which the fitness of the offspring are independent from the fitness of the parents, hence its parent-offspring fitness correlation is 0, that is, its heritability is $h = 0$.

For the extreme case $h = 1$, we obtain a search operator with maximal heritability and no variability that produces a clone of the parent as offspring, for which $\hat{f}_x$ is not defined. For this operator, the evolvability is 0 because the probability of obtaining offspring with fitness *strictly* better than its parent's fitness is 0. However, for the limit $h \to 1^-$ the evolvability tends to its maximum value.

There is a very important difference, in terms of impact on evolvability, between a search operator, such as random search, with no heritability and a search operator with a positive heritability: the evolvability of the former is always worse than the evolvability of the latter. This is because, as we mentioned above, $\hat{f}_x$ is a strictly decreasing function of h. To have a tangible understanding on the impact of the heritability h on the evolvability, we plot the ratio of the evolvability of an operator with positive heritability over the evolvability of random search in function of the parental fitness (see Figure 3.5). This plot tells us how many times more likely it is to improve on a parent, with a given above average fitness, by applying an operator with positive heritability than by applying random search (or equivalently, an operator with no heritability). On the abscissa, we have the normalized fitness of the parent (so for example, 5 means a fitness 5 standard deviations above the average fitness of the fitness of the problem). On the ordinate, we have the ratio of the evolvability of operators with positive heritability over the evolvability of random search in logarithmic scale. We have plotted three curves with heritability values of 0.1, 0.5 and 0.9 corresponding in the graph, respectively, to the lower curve, to the

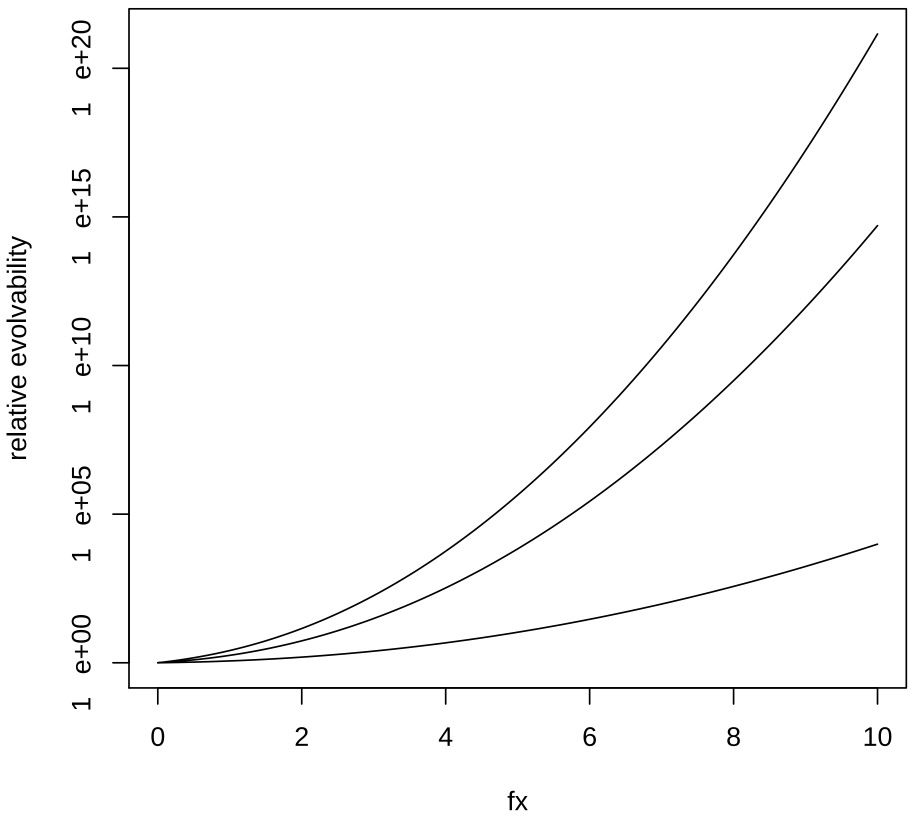


Figure 3.5: Relative evolvability of operators with positive heritability as function of parental fitness.

middle curve and to the upper curve. Firstly, we can see from the graph that the higher the heritability value, the higher the evolvability ratio for any choice of parental fitness. Secondly, as the parental fitness increases the evolvability ratio grows (super-)exponentially for all three curves. This means that *operators with positive heritability have exponentially better chances than random search to improve over the parental fitness as the parental fitness increases.* At the right end of the graph, for $f_x = 10$, the values of the evolvability ratio of the three curves are in the order of magnitude of 10^4, 10^{14} and 10^{21}. So, even an operator with very small heritability can perform orders of magnitude better than random search when the fitness to improve upon is well-above the average.

To summarize, we want to stress this simple, counter-intuitive and fundamental fact: *under the simplifying assumptions made, even something so ordinary as a loose correlation between the fitness of the parents and the fitness of the offspring gives to the evolutionary algorithms many* *orders of magnitude more chance than to random search to find better and better solutions.* We believe that much of the power of stochastic search algorithms seen in practice resides in this simple yet amazing fact.

3.9.2 Fitness landscape and heritability

In this section, we link the smoothness of a fitness landscape with the heritability of geometric mutation and geometric crossover. We will show that smoother fitness landscapes give rise to higher heritability.

Statistical model of smooth fitness landscape

Informally, we have defined smooth fitness landscape as the class of landscapes for which closer solutions tend to have closer (more correlated) fitness values. This notion of smoothness is not opposed to the notion of ruggedness: a fitness landscape may be “bumpy” yet still being smooth in the sense above. Instead, this notion of smoothness is opposed to the notion of completely random fitness landscape, for which the distance between solutions bear no information about the relationship (correlation) between their fitness values.

We can formalize the notion of smooth fitness landscape in the above sense using a statistical model. A statistical model is a probability distribution constructed to enable inferences to be drawn or decisions made from the data. We will consider fitness landscapes as realizations of an underlying statistical model. The statistical model of fitness landscape is, therefore, an abstraction of fitness landscape that specifies a set of statistical properties that all its realizations, the specific instances of fitness landscape, have in common. The type of statistical model selected shapes what is essential in its realizations and what instead is to be considered as random noise.

The type of statistical model that we consider is a isotropic random Gaussian field. This choice is based (i) on the fact that interesting real-world problems have Gaussian-like fitness distributions (as noted is section 3.9.1); (ii) on the assumption that the most important statistical feature of the fitness landscape that affects the performance of an evolutionary algorithm is the correlation structure (correlation function) of the fitness landscape; (iii) on that Gaussian random fields are relatively simple to use and well-studied; (iv) on that isotropic Gaussian random fields are naturally defined on metric spaces.

An isotropic Gaussian random field is a multivariate normal distribution in which every random variable is associated with a location in space and expresses the probability distribution of some measurable entity (e.g., height) at that specific location. In an isotropic Gaussian random field, all random variables (points in space) are identically normally distributed (mean=0, variance=1). Furthermore, the correlation between any two random variables is a function only of their distance. So, an isotropic Gaussian random landscape is completely defined by its correlation function $r(d)$, which tells for each distance d the correlation r between any two random variables at that distance.

An isotropic Gaussian random field can be easily seen as a statistical model of fitness landscape of a problem with Gaussian fitness distribution: every candidate solution is associated with a random variable (location); the measurable entity at each location is the fitness of the candidate solution; the random variable at a location describes the fitness distribution of the candidate solution associated with it (this distribution depends on what other solutions's fitness we already know); the distance between random variables is the distance associated with the landscape between candidate solutions.

We define a statistical model of *smooth* fitness landscape as: (i) having a *decreasing correlation function*, so that closer solutions have stronger positive correlation or lower negative correlation; (ii) *having positive correlation from distance zero until the average distance between solution in the search space, or further* (so, two randomly picked solutions, whose expected distance is the average distance, are not expected to have negative correlation). We say that a fitness landscape is *smoother* than another fitness landscape, if the former is associated with a correlation function that lies completely above the correlation function associated with the latter landscape.

If we know that a particular landscape under consideration is a realization of the statistical model of smooth fitness landscape, we call this landscape a smooth landscape. In general, *knowing that a particular landscape under consideration is a realization of a gaussian random field allows us to do spatial inference*: given that we know the fitness of some solutions at some locations, we can infer the fitness distributions of solutions at other locations for which the fitness is unknown using the correlation relationship between the fitness at known and unknown locations (using the correlation function coming with the statistical model of the landscape). This is, in fact, a form of multiple regression called gaussian process regression[131].

Heritability of geometric mutation

In the following, we link the heritability of a mutation operator defined on the search space only in terms of distances between parent and offspring, and the correlation function of the statistical model of smooth fitness landscape.

Let us first consider a simpler mutation operator, the circle mutation, that picks offspring uniformly at random on a circle at a given fixed distance d_c from the parent. *By definition of correlation function r, the correlation of the fitness of the parent and the fitness of the offspring, or in other words the heritability, of the circle mutation operator is $r(d_c)$.* Let us now consider a mutation operator that picks offspring uniformly at random at different distances from the parent according to a probability distribution P_d on the distances between offspring and their parent. In first approximation, the heritability of this mutation operator is the average of the heritabilities of the circle mutation operators at all distances weighted by the probability distribution on the distances: $h = \sum_{i=0,d_{max}} P_i \cdot r(i)$. Notice that, if the probability distribution on the distances is rapidly decreasing as the distance increases, as it is normally the case in mutations used in practice, on smooth fitness landscapes (for which $r(i)$ is a decreasing function) the heritability of the mutation operator is relatively high. This is because, in the weighted average, higher values of the correlation functions have larger weights. Also notice that, *the smoother the fitness landscape, the higher the heritability of this mutation operator.* This relationship is very important because one can indirectly control heritability and, in turn, evolvability and ultimately the performance of the evolutionary algorithm endowed with this mutation operator, by acting on the smoothness of the fitness landscape.

Above we have only given an estimate of the correlation between the fitness distribution of the parents and of the offspring. However, it is possible to determine completely the conditional distribution of the fitness of the offspring given the fitness of the parent. Since the underlying model of fitness landscape is a Gaussian random field, the fitness distribution of the offspring is Gaussian because it can be obtained by a linear combination of conditional fitness distributions at various distances from the parent, which are Gaussian. In fact, all geometric operators defined only in terms of distances in a Gaussian random field have Gaussian fitness distributions of the offspring because they can be obtained by linear combinations and conditioning of Gaussian fitness distributions.

Heritability of geometric crossover

In the following, we show how smoothness of the fitness landscape is translated into heritability for recombination operators with two parents.

Heritability of a genetic operator is the correlation between two variables, the fitness of the parent and the fitness of the offspring. Since in recombination operators we have 3 variables, 2 for the parents and 1 for the offspring, we define heritability as the correlation between the fitness of the mid-parent, which is a variable consisting of the average fitness of the parents, and the fitness of the offspring. This practice is routinely done in quantitative genetics to compute the response to selection. Given two parents x_1 and x_2, and an offspring y, the correlations among their fitness values depend only on their distances using the correlation function: the fitness correlation between x_1 and x_2 is $r(d(x_1, x_2))$, the fitness correlation between x_1 and y is $r(d(x_1, y))$, and the fitness correlation between x_2 and y is $r(d(x_2, y))$. The fitness correlation between y and the mid-parent, that is, the correlation between $f(y)$ and $\frac{f(x_1)+f(x_2)}{2}$, can be computed using a linear transformation of the covariance matrix of x_1, x_2 and y, and it is

$$\frac{\frac{r(d(x_1,y))+r(d(x_2,y))}{2}}{\sqrt{\frac{1+r(d(x_1,x_2))}{2}}}. \tag{3.9.1}$$

A *distance-based recombination operator* is a 2-parental operator completely defined as a function of the distances between parents and offspring, which are $d(x_1, x_2)$, $d(x_1, y)$ and $d(x_2, y)$. So, a distance-based recombination operator specifies, for every distance $d(x_1, x_2)$ between parents, and for each possible pair of distances between parents and offspring $d(x_1, y)$ and $d(x_2, y)$, the probability of picking offspring uniformly at random on the set of points identified by these distances. This definition encompasses a wide range of recombination operators. In fact it allows us to define recombination operators with probability distributions over all geometric figures that can be fully characterized in terms of distances from two fixed points (the parents), such as segments between two points, ellipses with the two points as foci, extension rays originating in a point and passing through a second point, extended segments passing through two points, hyperbola with the two points as foci, and many more.

Among all distance-based recombination operators, geometric crossover, which picks offspring in the segment between the two parents, is special in that all its offspring maximize the mid-parent fitness correlation on a smooth landscape, hence maximizing the overall heritability of the operator. We illustrate this in the following.

For any distance between parents $d(x_1, x_2)$, and for any distance from one parent and the offspring $d(x_1, y)$, the points y in the segments between x_1 and x_2 are those at a minimum distance $d(x_2, y)$ to the other parent. So, when $d(x_1, x_2)$ and $d(x_1, y)$ are fixed, the points $y \in [x_1, x_2]$ minimize $d(x_1, y)$. In terms of correlation, since the correlation function of a smooth landscape is a decreasing function of the distance, we have that when $r(d(x_1, x_2))$ and $r(d(x_1, y))$ are fixed, the points $y \in [x_1, x_2]$ maximize $r(d(x_1, y))$. So, the points $y \in [x_1, x_2]$ are those which also maximize the mid-parent fitness correlation (see equation 3.9.1) for every x_1 and x_2, and for every $d(x_1, y) \leq d(x_1, x_2)$. In other words, for each distance from a parent, on a smooth landscape, the mid-parent fitness correlation is higher for points in the segment between parents than for points outside it. Notice that the mid-parent fitness correlation for the points in the segment at some distance from the parents can be higher than that for points in the segment at some other distance from the parents. Which distance from the parents results in higher mid-parent fitness correlation depends on the details of the correlation function of the fitness landscape.

Analogously to the case of mutation, we can compute the heritability of a crossover operator, in first approximation, by averaging the heritabilities (mid-parent fitness correlations) across the distances from parents and offspring weighted by the probability P_d of picking offspring at that distance. The heritability h of the crossover operator is a function of the distance $d(x_1, x_2)$ between parents. So, we have $h(d(x_1, x_2)) = \sum_{i=0}^{d(x_1,x_2)} P_{d(x_1,x_2),i} \cdot r_{d(x_1,x_2)}(i)$ where $P_{d(x_1,x_2),i}$ is the probability of picking an offspring at a distance i from parent x_1 in a segment of length $d(x_1, x_2)$. The term $r_{d(x_1,x_2)}(i)$ is the mid-parent fitness correlation (between the fitness of the parents x_1 and x_2 and the fitness of the offspring y) with parents at a distance $d(x_1, x_2)$, and offspring y in the segment between x_1 and x_2 at a distance i from parent x_1. Using equation 3.9.1 we obtain:

$$r_{d(x_1,x_2)}(i) = \frac{\frac{r(i)+r(d(x_1,x_2)-i)}{2}}{\sqrt{\frac{1+r(d(x_1,x_2))}{2}}}. \tag{3.9.2}$$

As a first approximation, assuming the correlation function to be a linear function of the distance, the overall heritability of the crossover operator can be calculated exactly and it does not depend on the specific probability distribution of the distances:

$$h(d(x_1, x_2)) = \frac{r(d(x_1, x_2))}{\sqrt{\frac{1+r(d(x_1,x_2))}{2}}}.$$

The heritability of crossover is an increasing function of $r(d(x_1, x_2))$ and, therefore, it is a decreasing function of $d(x_1, x_2)$ on smooth landscapes because, by definition, their correlation function is a decreasing function. *This means that the heritability of crossover with closer parents (smaller $d(x_1, x_2)$) is higher.* Analogously for the case of mutation, *the smoother the landscape the higher the heritability of crossover.* This is because for the same distance between parents, on a smoother landscape the corresponding correlation function value is larger, hence the heritability of crossover for that same distance between parents is also higher.

Finally, we compare the heritability of crossover and mutation on the same landscape. We compare a circle mutation operator which picks offspring at a distance d from its parent with a crossover operator with parents at a distance $2d$ that picks offspring always halfway between parents. The comparison is fair because the offspring are at the same distance from their parents in both operators. Then, we ask when the heritability of crossover (using equation 3.9.1 with $d(x_1, y) = d(x_2, y) = d$ and $d(x_1, x_2) = 2d$) is larger that the heritability of mutation (which is $r(d)$):

$$\frac{r(d)}{\sqrt{\frac{1+r(2d)}{2}}} \geq r(d). \tag{3.9.3}$$

Interestingly, it turns out that the heritability of crossover is always higher than the heritability of mutation.

Wrong counter-examples

In this section, we have presented results linking smoothness of the fitness landscape and heritability of geometric operators. In order to anticipate a number of subtly wrong counter-examples one may rise, that seem to contradict the previous analysis, it is important to discuss the scope of the validity of our model. The results obtained in this section are based on a statistical model of fitness landscape, hence *they are valid for all the realizations of fitness landscape that meet the underlying hypothesis behind the statistical model.*

A fundamental underlying hypothesis inherent to all statistical models is that they model both *regularity* and *variability* of their realizations. Whereas the former aspect is obvious to the most, the latter aspect is often overlooked. We illustrate these two aspects for the simple case of a gaussian distribution (statistical model). If one draws a random sample with gaussian distribution (realization of the model), one may reasonably expect that the sampled value is close to the mean of the distribution (regularity), but also need to expect that almost never the sampled value is exactly coinciding with the mean of the distribution (variability). In fact, it is possible to test whether some empirical data fit the theoretical distribution they supposedly come from using a goodness-of-fit test. The possible outcomes are three: fitting, mis-fitting, over-fitting. Mis-fitting happens when the regularity in the data does not match the regularity of the model. Over-fitting corresponds to a fit which is too good to happen very often and indicates an anomalous lack of variability in the empirical data (one possible reason for this happening is that the data were fabricated to match the model). In many applications of statistical models, data mis-fitting is considered very bad because this indicates the choice of a wrong statistical model, whereas over-fitting is considered tolerable as long as the reasons behind it are understood. However, *in our model, variability between the fitness of parents and offspring (that allows offspring to be sometime better than their parents) is as essential as its regularity (predictability/heritability of the fitness of the offspring from the fitness of the parents). Since both these aspects contribute equally importantly to the performance of an evolutionary algorithm, both mis-fitting and over-fitting of a specific realization are equally bad-fit to the model.*

The hypotheses behind our statistical model of smooth landscape a valid realization should meet are:

1. *gaussian:* the frequency distribution of the fitness values of the solutions of the landscape must be gaussian-like

2. *smoothness:* the empirical correlogram, which is the correlation function estimated from sampled solution in the landscape, must be decreasing or with a decreasing trend, and positive for distances smaller than the average distance

3. *variability:* the landscape must not over-fit the statistical model according to some test of goodness-of-fit

4. *stationary(isotropic):* the landscape must not present strong “trends”

In the field of geostatistics (also known as statistics for spatial data), they have statistical tools to check whether there are trends in the landscape (see for example [23]).

A simple, tempting and wrong counter-example it may be risen is the following. Let us consider a fitness landscape whose search space is the line of real number, and whose fitness is a linearly increasing function. Clearly this fitness landscape is very smooth because closer points on the real line have more correlated fitness values. However, the geometric crossover, which on the real line picks offspring in the interval between the two parental points, has clearly evolvability zero because no offspring can have fitness larger than the largest fitness of the two parents (for the linearity of the function, the fitness of the points in the interval between the parents are in the interval between the fitness of the parents). So, one may conclude, this must be a counter-example of that, on a smooth landscape, the crossover has a high evolvability.

This example fails to be a valid counter-example because some of the assumptions behind our statistical model are not met. Let us consider, for this example, the four requirements stated above. Firstly, the fitness distribution of the fitness is not gaussian: since in a linear function every point in the real line has different fitness value, the fitness distribution is uniformly distributed. However, this is not the crucial point because a variation of the counter-example with gaussian distribution of the fitness can be built. The second requirement, smoothness, is met by the example. The third requirement, variability, fails by far: the fitness landscape does not present enough variation in the fitness values of neighboring points to be thought as a fitting realization of a gaussian random field. Figure 3.6 shows typical realizations of stationary gaussian processes (gaussian random field on the real line). Also the forth requirement, stationarity, fails because the landscape has a strongly definite upward trend.

So, the reason geometric crossover has no evolvability in the considered example is due to a combination of the lack of variability and the particular trend of the landscape. In terms of impact on evolvability, mutation is less sensitive than crossover to global trends of the landscape. However, this characteristic of crossover is not necessarily bad. In conjunction with geometric crossover, some special trends in the fitness landscape can be highly beneficial. We illustrate this for random field with a convex trend in chapter 15.

3.9.3 Smooth landscape and Lipschitz continuity

In this section, we explain the relationship between Lipschitz continuity, which is a well-known measure of smoothness of a function (landscape), and the notion of statistical smoothness introduced in section 3.9.2.

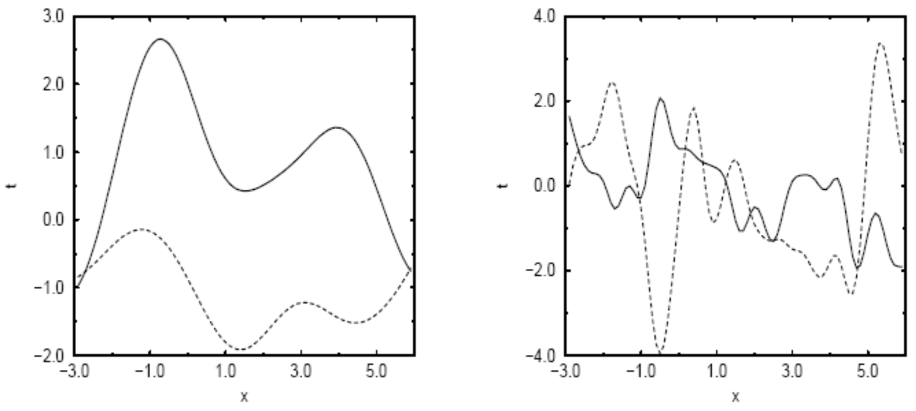


Figure 3.6: Examples of realizations of gaussian processes. On the left, realizations of a smooth landscape. On the right, realizations of a less smooth landscape.

Lipschitz continuity and normalized Lipschitz continuity

Lipschitz continuity [152] is a smoothness condition for functions which is stronger than regular continuity. Intuitively, a Lipschitz continuous function is limited in how fast it can change. A line joining any two points on the graph of this function will never have a slope steeper than a certain number called the Lipschitz constant of the function. The concept of Lipschitz continuity can be defined for functions on metric spaces (fitness landscapes).

Definition 3.9.1. Given a metric space $M = (S, d)$, a map $f : M \to \mathbf{R}$ is Lipschitz continuous with constant k, if k is the smallest constant such that $\forall x_1, x_2 \in S : \frac{|f(x_1)-f(x_2)|}{d(x_1,x_2)} \leq k$.

Notice that all functions on a finite set are Lipschitz continuous for some k.

A Lipschitz continuous function f with constant k has the two equivalent properties:

- Given two fixed values f_1 and f_2 of the function f, any two locations x_1 and x_2 with these values of fitness must be at least at a distance $d(x_1, x_2) \geq \frac{|f_1-f_2|}{k}$.
- Given two fixed locations x_1 and x_2 at a distance $d(x_1, x_2)$, their fitness values f_1 and f_2 cannot differ more than $|f_1 - f_2| \leq k \cdot d(x_1, x_2)$.

The smaller the Lipschitz constant k, the smoother the fitness landscape in this sense. From an optimization viewpoint, however, the Lipschitz constant is not a meaningful measure of smoothness of fitness landscapes. In fact, a simple linear re-scaling of the fitness landscape, say

$g = a \cdot f$, re-scales also the Lipschitz constant accordingly. So, if f has Lipschitz constant k, g has Lipschitz constant $a \cdot k$. Since a can be chosen arbitrarily, the Lipschitz constants of f and g can be arbitrarily different. However, for most meta-heuristics f and g have exactly the same hardness (e.g., evolutionary algorithms with tournament selection are not affected by fitness re-scaling). We can easily eliminate the re-scalability problem by considering, as a measure of smoothness of the landscape, instead of the Lipschitz constant itself, a version of it normalized by the difference between the maximum fitness and the minimum fitness (maximum global variation of fitness). We call this the *normalized Lipschitz constant* of the fitness landscape. Clearly, unlike the Lipschitz constant, the normalized Lipschitz constant is invariant with respect to linear fitness re-scaling.

Definition 3.9.2. Given a metric space $M = (S, d)$, a map $f : M \to \mathbf{R}$ is normally Lipschitz continuous with normal constant $\bar{k} = \frac{k}{f_{max} - f_{min}}$, where k is the Lipschitz constant of f, and f_{max} and f_{min}, are, respectively, the maximum and minimum fitness values of f.

Notice that the previous definition is a meaningful measure of smoothness for bounded functions. For unbounded functions, the normalized Lipschitz constant is 0.

Theorem 3.9.1. *The normalized Lipschitz constant for fitness landscapes whose underlying metric space is graphic is always in the range* $[0, 1]$. *The normalized Lipschitz constant for fitness landscapes based on generic metric spaces can be any arbitrarily large positive real.*

Proof. In graphic metric spaces, the minimum distance between two distinct points is 1 which corresponds to the case in which the two points are direct neighbours on the graph associated with the graphic metric space. From the definition of normal Lipschitz constant we have that $\bar{k}$ is the smallest constant such that $\forall x_1, x_2 \in S : \frac{|f(x_1) - f(x_2)|}{d(x_1, x_2) \cdot (f_{max} - f_{min})} \leq \bar{k}$. Since $\frac{|f(x_1) - f(x_2)|}{f_{max} - f_{min}}$ is always smaller or equal to 1, and $d(x_1, x_2)$ is always larger or equal to 1, then $\bar{k}$ is always equal or smaller than 1. Also, from its definition it is clear that $\bar{k}$ is always a positive number.

To prove the second part of the theorem, let us consider the function $f(x) = x/a$ defined on an interval $[0, a]$ of the real-line, where $a \in [0, 1]$ is a parameter of the function. The Lipschitz constant of f is the slope its graph, which is $k = 1/a$. The minimum fitness is $f_{min} = f(0) = 0$ and the maximum fitness is $f_{max} = f(a) = 1$. Consequently the normalized Lipschitz constant of f is $\bar{k} = \frac{k}{f_{max} - f_{min}} = 1/a$. Since a can be arbitrarily chosen in $[0, 1]$, $\bar{k}$ can be any arbitrarily large positive real. □

The *maximum global variation of the fitness* is $\Delta F = \max_{f_1, f_2 \in F} |f_1 - f_2| = |f_{max} - f_{min}|$. The *maximum local variation of the fitness* of a landscape based on a graphic space is $\Delta f = \max_{x_1, x_2 \in S : d(x_1, x_2) = 1} |f(x_1) - f(x_2)|$.

Theorem 3.9.2. *In landscapes based on graphic metric spaces the normalized Lipschitz constant is equal to the maximum local variation over the maximum global variation.*

Proof. Let us consider a landscape on a graphic metric space with maximum local variation Δf.

The maximum fitness variation of solution s_1 and s_2 at a distance n is smaller or equal to $n \cdot \Delta f$. This holds because it exists a path of $n+1$ direct neighbors solutions in the graph associated with the metric, starting from s_1 and ending in s_2, in which any two successive solutions cannot have fitness variation larger than Δf. So, since the summation of all the fitness variations of successive solutions in this path is an upper-bound of the variation of the fitness values of s_1 and s_2, they can differ at most for $n \cdot \Delta f$.

By definition of Lipschitz constant k, for any distance n we have that the smallest k has upper-bound $\frac{n \cdot \Delta f}{n} = \Delta f$. The worst case is $n = 1$ for which $k = \Delta f$. So, the Lipschitz constant of the landscape equals the maximum local variation of the fitness. The normalized Lipschitz constant $\bar{k}$ is therefore $\bar{k} = \frac{\Delta f}{\Delta F}$, which is the ratio among the maximum local variation and the maximum global variation of the fitness landscape. □

In the light of the two theorems above, the interpretation of the normalised Lipschitz constant becomes evident when we consider combinatorial spaces. In this case, fitness landscapes associated with combinatorial optimization problems are typically associated with unweighted neighbourhood structures, hence with graphic distances. Therefore, the normalised Lipschitz constant ranges from 0 (excluded), when all direct neighbours have the same fitness, to 1, when the maximum and minimum are direct neighbours. The closer to 0, the smoother the landscape. The closer to 1, the more discontinuous the landscape.

When the fitness distribution is fixed, the Lipschitz constant can be used to compare the *relative smoothness* of fitness landscapes based on different metrics.

For any fitness distribution, the normalized Lipschitz constant gives an *absolute measure of smoothness* of the fitness landscape (particularly well-suited for fitness landscapes based on graphic metric spaces). Notice, however, that it makes sense to compare smoothness via normalised Lipschitz constant only on spaces with the same number of elements.

Lipschitz continuity and smooth landscape

In the following, we elicit the connection between a smooth landscape in the Lipschitzian sense and in the statistical sense (section 3.9.2).

The correlation function of a gaussian random field is closely related to other two functions, the covariance function and the semi-variogram[23].

For an isotropic random field, its covariance function $C(\delta)$ gives the covariance of the values of the random field at the two locations x and y at a distance $d(x, y) = \delta$. The correlation function $r(\delta)$ coincides with the covariance function when the distribution of the values f of the random field (at any location) are transformed into normal form (mean zero and unitary variance). So, we have: $r(\delta) = \frac{C(\delta)}{var(f)}$.

For an isotropic random field, the semi-variogram is defined as the expected squared increment of the values between locations x and x_δ at a distance $d(x, x_\delta) = \delta$:

$$\gamma(\delta) = \frac{E((f(x) - f(x_\delta))^2)}{2}.$$

The semi-variogram is related to the covariance function by $\gamma(\delta) = C(0) - C(\delta) = var(f) - C(\delta)$.

The semi-variogram is of particular interest to us because it can be related with the Lipschitz constant k of the fitness landscape:

$$\gamma(h) = \frac{E((f(x) - f(x_\delta))^2)}{2} \leq \frac{(k \cdot \delta)^2}{2}.$$

Notice that the semi-variogram is bounded from above by an increasing function of k. So, for smaller values of k, which means for a smoother landscape in the Lipschitzian sense, the bound to the semi-variogram becomes tighter. From the relations between the semi-variogram, the covariance function and the correlation function, a bound of the semi-variogram from above translates into a bound to the correlation function from below, and the tighter the bound form above for the former, the tighter the bound from below for the latter. So, for smaller values of k, the correlation function is increasingly bounded by how quick it can decreases. In other words, *a smoother landscape in the Lipschitzian sense gives a stricter bound on the worse smoothness in the statistical sense.*

Intuitively, Lipschitzian continuity can be thought as worst-case continuity, whereas statistical continuity as average-case continuity. Notice that a fitness landscape can be very smooth in the statistical sense, and be very discontinuous in the Lipschitzian sense. So, large values of the Lipschitz constant do not necessarily imply lack of smoothness in the statistical sense. However, a smooth landscape in the worst-case sense, it is necessarily a smooth landscape in the average case. This last point is very important because, if we can tell that a fitness landscape has a small Lipschitz constant, then we can infer that the same fitness landscape is smooth in the statistical sense. However, the Lipschitz constant in itself, being a relative measure of smoothness, it cannot tell in absolute terms how smooth a fitness landscape is. Fortunately, the normalized Lipschitz constant is an absolute measure of smoothness for fitness landscapes based on graphic spaces typically associated with combinatorial problems. So, *at least for combinatorial problems, we have a clear rule: values of $\bar{k}$ close to zero indicates smooth landscape in the Lipschitzian sense and also in the statistical sense.*

To recap, the chain of implications that links the normalized Lipschitz constant to the performance of an evolutionary algorithm with geometric operators is as follows:

1. small normalized Lipschitz constant
2. smooth landscape in the worse case
3. smooth landscape in the average case
4. high heritability of geometric operators
5. high evolvability of geometric operators
6. high performance of evolutionary algorithm

Valid realizations of a statistical model do not need to be the outcome of a random experiment, they can be constructed deterministically. What make them valid realizations is that they meet the hypotheses behind the statistical model even if they do not have a stochastic origin: if they meet these hypotheses, the results of the statistical model apply to them.

We have shown how the class of Lipschitz continuous landscape relates with the hypothesis of smoothness of our statistical model of smooth landscape. How the other hypotheses behind our statistical model hold? The requirement of the the fitness distribution of being gaussian-like is likely to be true for many interesting real-world problems: average fitness solutions are easy to sample at random, and as the fitness of the required solution increases, they become much and much harder to get by random sampling (so they become rarer and rarer, and the overall fitness distribution is bell-shaped). The requirement of variability between the fitness of neighboring solutions depends on how the distance associated with the fitness landscape relates with the fitness (so, it is case-specific). A way to counteract the lack of variability in the fitness landscape, in case needed, is to instill more variability in the probability distribution of the search operator (e.g., using a mutation operator that picks with higher chances offspring at larger distances from the parent). Intuitively, the requirement of isotropy (or lack of overall trend) is linked with a requirement of symmetry between the fitness of solutions and distance. The landscapes normally associated with combinatorial optimization problems are generally highly symmetric by construction. However, the presence of a general trend cannot be ruled out. Mutation is less affected by global trends in the fitness landscape, so combination of mutation and crossover may counteract the negative effect of a global trend on crossover, if present.

3.10 Embedding problem knowledge in the search

3.10.1 Problem knowledge and fitness landscape

Geometric operators are defined as functions of the distance associated with the search space. However, the search space does not come with the problem itself. The problem consists only of a fitness function to optimize[6], that defines what a solution is and how to evaluate it, but it does not give any structure on the solution set. The act of putting a structure over the solution set is part of the search algorithm design and it is a designer's choice. A fitness landscape is the fitness function plus a structure over the solution space. So, for each problem, there is one fitness function but as many fitness landscapes as the number of possible different structures over the

[6]In optimization there is usually a set of constraints besides the objective function. These constraints restrict the domain of the objective function. However, from a formal point of view, objective function and constraints can be seen together as an unconstrained objective function defined directly on the restricted domain.

solution set. In principle, the designer could choose the structure to assign to the solution set completely independently from the problem at hand. However, because the search operators are defined over such a structure, doing so would make them decoupled from the problem at hand, hence turning the search into something very close to random search. To avoid this, one needs to exploit problem knowledge in the search by using a distance that makes sense for the problem at hand. In the case of geometric operators, we have seen in section 3.9 that they work well on smooth landscapes. So, we will need to choose an operator associated with a distance that together with the problem at hand gives rise to a smooth landscape.

The distinction between problem and fitness landscape is also conceptually important because it clarifies the relation among the fundamental notions of problem, problem knowledge, fitness landscape and search algorithm as follows. *The problem knowledge used by an evolutionary algorithm that uses geometric operators is embedded in the connectivity structure of the landscape. The landscape is therefore a knowledge interface between a problem without structure and a formal search algorithm that has no direct link with the problem whatsoever and yet, it can perform better than random search.*

Another way of embedding problem knowledge into a formal search algorithm, not based on the fitness landscape, was proposed by Radcliffe [125]. His theory is based on the notion of forma, which is a representation-independent generalization of schema. In Radcliffe's theory, the problem knowledge is embedded in formal search operators, which transmit formae from parents to offspring, by the designer of the algorithm who chooses problem-specific formae according to what he believes are important characteristics of the problem at hand. In chapter 14, we will present further connections between forma theory and the geometric framework.

3.10.2 Using the problem description to embed knowledge in the landscape

One can exploit problem knowledge in the search by carefully designing the connectivity structure of the fitness landscape and choosing a distance that, in connection with the problem at hand, gives rise to a smooth landscape. To this end, the notion of Lipschitz continuity is of fundamental importance because it allows us to choose a smooth landscape directly from the problem description, by inspection of its objective function. We have seen in section 3.9.3, that in case of graphic distances, the smaller the maximum local fitness variation (of two neighboring solutions), the smoother the fitness landscape. For many combinatorial optimization problems, it is straightforward to select a neighbourhood structure associated with a fitness landscape with a small maximum local fitness variation (respect to the maximum global fitness variation) from their objective function. Specifically, this can be usually done by selecting a solution representation related with the formal mathematical representation of a solution employed in the algebraic definition of the objective function of the problem, together with an edit move based on such a representation.

We illustrate how to choose a smooth fitness landscape and associated geometric operators on a simple problem.

Let us consider an instance of size $n = 100$ of the onemax problem. The objective function (to maximize) of the onemax problem is: $\sum_{i=1}^{n} x_i$ where a solution $x = (x_1, \cdots, x_n)$ is a binary vector of size n.

We can *choose* to represent the genotype of a solution after the formal representation employed in the objective function, which is, using a binary vector. The important thing to realize is that this is a design choice based on the knowledge of the objective function of the problem. In fact, for example, we could have chosen to represent the genotype of a solution as the integer number whose binary code corresponds with the formal representation of the solution employed in the objective function. Given that we have chosen the binary representation for the genotype, we can *choose* the bit-flip move as edit move on this genotype, to define which solutions are neighbors. So, two genotypes one bit-flip away are neighbors. This is again a design choice. For binary strings, we could have chosen another edit move, for example, the swap move, which swaps the contents of two locations of the binary string. In that case, two genotypes one swap away would have been neighbors. The choice of the genotype and of the edit move, together are equivalent to the choice of the landscape associated with the instance of the onemax problem under consideration.

The maximum global variation of the fitness is $\Delta F = 100$, because the smallest fitness is 0 and the largest is 100. The maximum local variation of the fitness is $\Delta f = 1$, because two neighboring strings one bit-flip away differ in fitness always of 1. The normalized Lipschitz constant $\bar{k}$ of the landscape is the maximum local variation over the maximum global variation. The smaller this measure, the smoother the landscape. Since we have $\bar{k} = \Delta f/\Delta F = 1/100$, *this fitness landscape is very smooth* when compared with the two extreme of smoothness 0 (no change between neighbor solutions) and 1 (maximum and minimum are neighbors). So, we expect both geometric crossover and geometric mutation associated with the search space (Hamming space) of the fitness landscape to perform well (with respect to random search on the same problem instance). We have seen, in section 3.3, that the search operators associated with the Hamming space are the traditional mask-based crossover and the traditional point mutation for binary strings, which, in fact, are known to perform well on the onemax problem.

The same technique we used to choose a smooth landscape for the onemax problem can be easily used for other problems. In chapter 5, we apply it to choose smooth fitness landscapes and associated geometric operators well-suited for the n-queens problem and the TSP problem. In chapters 9 and 11, we use the same technique to design crossover operators for, respectively, Sudoku and for the graph partitioning problem.

3.10.3 Other methods to embed problem knowledge in the landscape

Depending on what we know about a problem, we can use other methods to choose good geometric operators for a specific problem at hand which are indirectly related with the notion of smooth landscape.

The first method is the "good mutation, good crossover" heuristics: if we know a good mutation for a problem, then the geometric crossover associated with it through the same distance is a good crossover for the problem. This is true, or likely to be true in most cases, because the heritability of geometric mutation and the heritability of geometric crossover are both increasing functions of the smoothness of the associated fitness landscape. If a mutation is good for a problem, then its heritability is high, and this is most likely due to the fact that the fitness landscape is smooth. Therefore also the geometric crossover associated with this fitness landscape has high heritability, hence good performance.

The second method is the "good neighborhood structure, good crossover and good mutation" heuristics: if a neighbourhood structure obtained by choosing solution representation and edit move for the problem at hand is good for local search, or other meta-heuristics, then it is also likely to be good for geometric crossover and geometric mutation based on it. This hypothesis makes sense because it mirrors the well-known rule of thumb that a good neighbourhood structure for a problem works well with different meta-heuristics [48]. This connects with the notion of smoothness of the fitness landscape because a general heuristic way to design good neighbourhood structures for local search and other meta-heuristics states that neighboring solutions should have similar fitness (which is, in fact, a requirement of smoothness). This rule of thumb has been rediscovered by many authors in many fields of search and optimisation [122].

The third method is "use geometric operators based on a meaningful distance for the specific interpretation of the solution representation". The same solution representation may allow for various notions of distance. Normally, every notion of distance is meaningful with regards to a specific interpretation of the solution representation. For example, in chapter 5 we will see that a permutation can have three natural interpretations: one in which the relative order of the elements is relevant, one in which their absolute positions are relevant, and one in which their adjacency relation is relevant. Depending on the interpretation of the permutation, the same edit move can be seen, at a phenotypic level, as a small change or a major change. For example, the inversion move, which inverts the order of a contiguous block of elements of the permutation, does a minimal change when one thinks of a permutation in terms of adjacency of its elements, but a major change when the same permutation is seen as a priority list, in which the relative order of the elements is important. A small change at a phenotypic level is reflected into a small change in terms of fitness values. So, a fitness landscape based on a move that induces a small change at a phenotypic level gives rise to a smooth landscape. Therefore, if the designer knows for the problem at hand, what a good representation is, and the right interpretation to give to it, he/she can, from this knowledge, select a distance that gives rise to a smooth landscape as basis for geometric crossover and geometric mutation, hence leading to good performances.

3.10.4 Analytic smoothness and computational complexity

In section 3.10.2, we have shown that the Lipschitz condition can be derived directly from the objective function of the problem at hand in analytical form. This is very different from other types of practices of analysis of the continuity of the fitness landscape, such as for example fitness-distance correlation, that require to be determined experimentally. Apart from the obvious, practical advantage of the Lipschitz condition of being a measure of continuity of the landscape that does not require to write a computer program and run experiments to be determined, there are other important consequences of being an analytical condition.

The first important consequence of the fact that the Lipschitz condition can be derived analytically is that the Lipschitz condition can be used as a direct design method, rather than a tool for analysis only. Given an objective function, it is relatively easy to figure out from its algebraic structure a representation and an edit move for which the worst-case local fitness variation is minimal. This cannot be achieved with an experimental measure of continuity.

A second important consequence of the Lipschitz condition being analytical is that it is not a property of a single fitness landscape, but it is a property of a family of fitness landscapes. This is because the objective function, from which the Lipschitz condition is derived, is a description of all instances of a problem. For example, let us consider a version of the TSP problem in which the distance between two cities is bounded by a constant d_{max}. Different TSP problem instances have different matrices of distances between cities, but they have the same objective function, which is a function of the distance matrix of the specific problem instance. It is easy to show that choosing the 2opt neighbourhood structure, one obtains a family of fitness landscapes, one for each TSP problem instance, which are all Lipschitz continuous with the same Lipschitz constant (or bounded by it), which depends only on d_{max}. Therefore, the Lipschitz condition is a worst-case condition encompassing all instances, and it gives a bound on the worst-case smoothness across all instances of the problem. This is impossible for an empirical measure of smoothness, which always concerns only a single fitness landscape.

A third important consequence is related with the one just above. The objective function of a problem describes all instances of the problem, across all sizes of the problem. So, the Lipschitz condition is a function of the size of the problem instance and it relates how worse-case smoothness changes with the problem instance size. This is very important because it means that the Lipschitz condition can tell something about scalability of the performance of evolutionary algorithms, which is, about their computational complexity.

Continuing the example in section 3.10.2, for the one-max problem of generic size n the maximum global variation of the fitness is $\Delta F = n$, because the smallest fitness is 0 and the largest is n. The maximum local variation of the fitness is $\Delta f = 1$, because two neighboring strings one bit-flip away differ in fitness always of 1. The normalized Lipschitz constant $\bar{k}$ of the landscape is $\bar{k} = \Delta f / \Delta F = 1/n$. So, the Lipschitz condition is a function of the problem instance size. This tells something about the scalability of geometric operators of this family of fitness landscapes: the smoothness $\bar{k}$ of the landscape is inversely proportional to the problem size n. *So, for increasing size of the problem instance, the fitness landscape becomes smoother. Therefore, this may counteract the increasing difficulty of the search due to a larger space, making the performance of the evolutionary algorithm scale better than random search.* This seems to hold in the case of the one-max problem. Evolutionary algorithms with traditional crossover and mutation operators for binary strings, that are geometric under Hamming distance, hit the optimum, on average, in $n \log n$ trials. Random search, in comparison, takes 2^n trials, on average, to hit the optimum.

As for the case of the onemax problem, we have found also that the smoothness of the landscape is inversely proportional to the problem size, for n-queens problem and TSP (see chapter 5), for Sudoku (see chapter 9) and for the graph partitioning problem (see chapter 11). One may conjecture that this is a characteristic common to many combinatorial problems because it happens when (i) the maximum local fitness variation is independent from the problem size, which is when the impact of an edit move on the fitness is independent from the problem size, and (ii) the global fitness variation is increasing with the problem size, which is when for larger problem instances the minimum fitness gets smaller and the maximum fitness gets larger. Both (i) and (ii) seem to be common properties of fitness landscapes normally associated with many combinatorial optimization problems.

A word of warning: the fact that combinatorial optimization problems that are NP-Hard can be associated with fitness landscapes that are increasingly smooth for increasing sizes of problem instance does not imply that NP-Hard problems are solvable efficiently (in polynomial time) by evolutionary algorithms! This is because the assumptions behind the notion of standard computational complexity and their computational classes are definitely not met by our computational complexity scenario (which would consider average-case computational complexity over all realizations of fitness landscapes of a parametric statistical model, and average running time of the algorithm to reach a good approximation of the best solution). Also, as we will see in chapter 5 for the case of TSP which is NP-Hard, even if increasingly smooth fitness landscapes with the problem size do exist for this problem, the geometric operators associated with these landscapes cannot be implemented in polynomial time, so in fact the complexity of the search is transferred from the fitness landscape to the search operator implementation.

3.11 Summary

In this chapter, we have introduced a geometric framework for evolutionary algorithms that clarifies the connections between representation, genetic operators, neighbourhood structure and distance in the landscape. Thanks to this framework a novel and general way of looking at crossover (and mutation) that is based on landscape topology and geometry has been put forward. Traditional crossover and mutation for binary strings have been shown to fit our geometric framework, which, as we will see later also encompasses a variety of other representations and associated operators. We have explained why geometric crossover works well on smooth landscapes by showing that evolvability and heritability of geometric crossover are linked with landscape smoothness. Also, we have shown how problem knowledge and fitness landscape are related and how to embed problem knowledge in the search done by geometric operators.

This framework presents a number of additional advantages. The theory is representation independent, and, therefore, it offers a unique opportunity for generality and unification. The theory provides a natural, direct and automatic way of deriving (designing) both mutation *and* crossover from the neighbourhood structure of a landscape. Conversely, if one adopts our geometric operators, one and only one fitness landscape is induced: that is we *do not* have a different landscape for each operator, but a common one for both. In the next chapter, we will apply the framework to real valued representations and operators.

Chapter 4

Real Vectors

As we have seen in the previous chapter, geometric crossover is a representation-independent generalization of the class of traditional mask-based crossover for binary strings, which is based on the distance of the search space seen as a metric space. Although real-coded representations allow for a very familiar notion of distance, namely the Euclidean distance, there are also other distances for it. Also, topological transformations of the real space give rise to further notions of distance. In this chapter, we study the geometric crossovers associated with these distances in a formal and very general setting and show that many pre-existing genetic operators for the real-coded representation are geometric crossovers. We also propose a methodology to remove an inherent bias of pre-existing operators by formally transforming topologies.

4.1 Introduction

As we saw in chapter 3, geometric crossover is a representation-independent operator defined over the distance of the search space. Informally, geometric crossover requires the offspring to lie between parents.

Geometric crossover encompasses naturally combinatorial and continuous spaces. So far we have focused on the study of geometric crossover for binary spaces and shown that their "geometrization" is possible and surprisingly insightful for the analysis and design of search operators. The definition of geometric crossover seems so natural for continuous spaces that only a limited investigation seems to be required. However, this is not true: continuous spaces are as rich in variety as combinatorial spaces. Indeed, beside the traditional Euclidean distance there are many other distances for real vectors. Hence, there are many different types of geometric crossovers for real vectors with quite different search properties. These make them suitable for different types of continuous problems.

In this chapter we start to study the geometric crossovers for continuous spaces. Natural starting points are Minkowski distances that are simple generalizations of the Euclidean distance. Geometric crossovers based on Minkowski distances have an inherent bias toward the center of the space. This bias could be potentially harmful for the search. We then study geometric crossovers for "glued" versions of these spaces for which the bias disappears. We also show that many pre-existing recombination operators for real vectors are geometric crossovers.

The reminder of the chapter is organized as follows. In Section 4.2, we review the most frequently used genetic operators for the real-code representation.

In Section 4.3, we present geometric crossovers based on p-norms. Geometric crossovers based on p-norms are biased towards the center of the space. In Section 4.4, we present geometric crossovers based on glued spaces that are unbiased. In Section 4.5, we show that a number of pre-existing recombination operators for the real-code representation are geometric and the implications of this. In Section 4.6, we discuss and summarise the findings.

4.2 Genetic operators for real-coded representations

In this section we review the most frequently used recombination operators for the real-code representation. These operators are described in detail in [8]. In the literature many recombination operators for real-coded representations are presented. In the following we give a taxonomy of recombination operators. This does not consider the specific probability distribution of the offspring but only what offspring can be generated with probability greater than zero given two parents. This taxonomy is important in relation with geometric crossover because the position of a recombination operator in this taxonomy is sufficient to tell whether it is geometric and what properties it possesses. This will be shown in Section 4.5.

There are two main families of recombination operators [114]: discrete recombinations and blend recombinations.

The discrete recombination family [133] is the straightforward extension to real vectors of the family of mask-based crossover operators for binary strings including n-point and uniform crossover. The mask is still a binary vector dictating for each position (or locus) of the offspring from which parent the (real) value for that position is taken.

The blend recombination family [83] does not exchange values between parents like discrete recombinations but it averages or blends them. Blend recombinations can be distinguished into line recombinations and box recombinations [39]. Line recombination returns offspring on the (Euclidean) line segment connecting the two parents. Box recombination returns offspring in the box (hyper-rectangle) whose diagonally opposite corners are the parents. Important variations of the last two recombination operators are the extended-line recombination and the extended-box recombination [113]. Extended-line recombination picks offspring on a segment passing through the parent vectors but extending beyond them. Analogously, extended-box recombination picks offspring on an extended box whose main diagonal passes through the parents but extends beyond them.

The most common form of mutation for real vectors generates an offspring vector by adding a vector m of random variables with zero expectation to the parent vector. There are two types of mutations, bounded and unbounded, depending on whether the range of values of the random variable is bounded or unbounded. The most frequently used bounded mutations are the creep mutation and single-variable mutation [26]. The most frequent unbounded mutations are the Gaussian and Cauchy mutations [132][167].

In the creep (or hyper-box) mutation, $m \sim U([-a, a]^n)$ is a vector of uniform random variables where a is a parameter defining the range of the offspring area. This operator yields offspring within a hyper-box centered on the parent vector.

In the single-variable mutation, m is a vector in which all entries are set to zero except for a random entry which is a uniform random variable $\sim U([-a, a])$.

Bounded mutation operators may get stuck in local optima. In contrast, unbounded mutation operators guarantee asymptotic global convergence [139]. The main type of unbounded mutation is the Gaussian mutation for which m is a multivariate Gaussian distribution.

4.3 Geometric crossovers based on p-norms

4.3.1 Problem versus fitness landscape

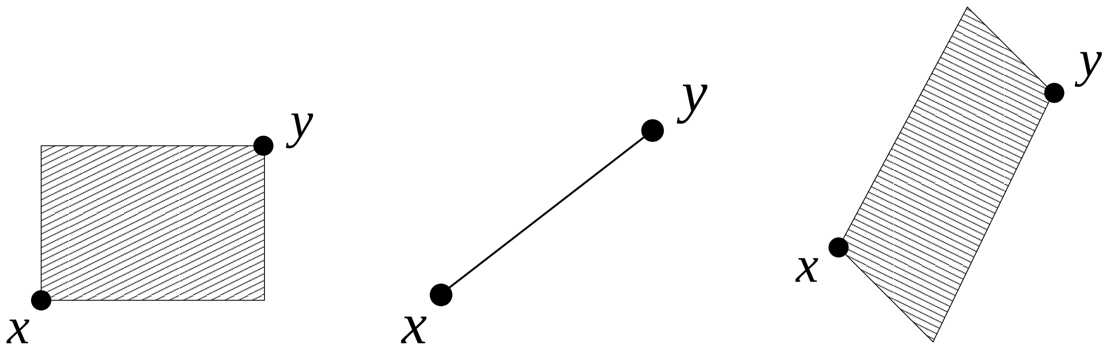
For continuous optimization problems, more than for combinatorial optimization problems, there is the assumption that problem and fitness landscape are completely interchangeable notions. However, this is not true: the problem is simply the objective function and it does not come with any *a priori* notion of distance; the fitness landscape is the objective function plus a notion of distance. The two concepts are normally understood as equivalent because the Euclidean distance is taken for granted.

This assumption however is not always a good one. In fact some problems become easier to solve when the distance chosen for the landscape is more natural for the problem addressed than the Euclidean distance. As we have seen in chapter 3, a more natural distance, better suited to the specific interpretation of the solution for the problem at hand, is likely to give rise to a smoother fitness landscape. Hence, the geometric operators associated with this distance are likely to perform better than those associated with the Euclidean distance. So, it is important to study geometric crossovers and geometric mutations for real-coded representations based on a number of alternative distances.

4.3.2 Metric segments and balls induced by p-norms

Geometric crossover depends on the metric of the given space. Although the Euclidean distance is ordinarily used for $\mathbb{R}^n$, there exist many other metrics for $\mathbb{R}^n$. We study geometric crossover for the real vector space under a more general and abstract metric: the Minkowski distance.

Given a vector space X, a *norm* is a real-valued function on X satisfying the properties of positiveness, positive homogeneity and subadditivity [152]. Considering $\mathbb{R}^n$ as a vector space,



(A) $[x;y]_{d_1}$ (B) $[x;y]_{d_2}$ (C) $[x;y]_{d_\infty}$

Figure 4.1: Metric segments induced by p-norms

we define the p-norm on $\mathbb{R}^n$, where p is a real number, as

$$\|x\|_p = \left\{ \sum_{i=1}^{n} |x_i|^p \right\}^{\frac{1}{p}}.$$

The special case where $p \to \infty$, the ∞-norm, is defined as follows:

$$\|x\|_\infty = \max_{1\le i\le n} |x_i| \;.$$

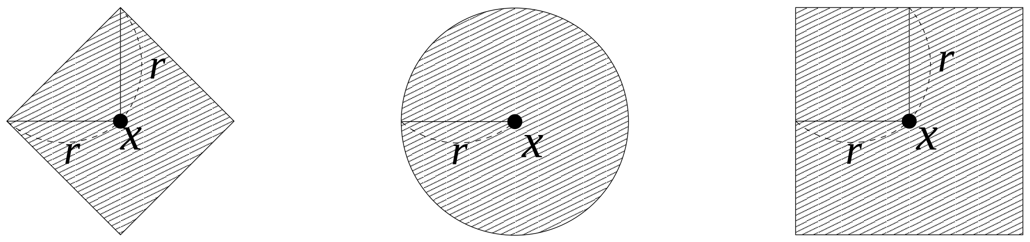
Once a norm $\|\cdot\|$ is defined, we can define a metric on $\mathbb{R}^n$ by setting $d(x,y) := \|x-y\|$. The family of Minkowski metrics $d_p(x,y)$ is defined as $\|x-y\|_p$ on $\mathbb{R}^n$. The metric segment between two points x and y on $\mathbb{R}^n$ under the metric d_p induced by the p-norm is defined as follows:

$$[x;y]_{d_p} = \{z \in \mathbb{R}^n : \|x-z\|_p \;+ \|z-y\|_p = \|x-y\|_p\}.$$

Also, the metric r-ball centered on a point x in $\mathbb{R}^n$ under the metric d_p is the set

$$B_{d_p}(x;r) = \{z \in \mathbb{R}^n : \|x-z\|_p \le r\}.$$

Figures 4.1 and 4.2 show examples of metric segments and metric balls on $\mathbb{R}^2$ for the cases $p = 1, 2, \infty$. Since $[x;y]_{d_2} \subseteq [x;y]_{d_1}$ and $[x;y]_{d_2} \subseteq [x;y]_{d_\infty}$, a geometric crossover under the metric d_2 is also geometric under the metrics d_1 and d_∞. Since $B_{d_1}(x;r) \subseteq B_{d_2}(x;r) \subseteq B_{d_\infty}(x;r)$, a geometric $r-$mutation under d_1 is also a geometric $r-$mutation under d_2, which is, in turn, geometric under d_∞.



(A) $B_{d_1}(x;r)$ (B) $B_{d_2}(x;r)$ (C) $B_{d_\infty}(x;r)$

Figure 4.2: Metric balls induced by p-norms

4.3.3 Geometric crossovers and their implementation

We can design new geometric crossovers using the metric segments induced by p-norms by choosing offspring that lie in the segments.

Although new crossovers are theoretically meaningful, if they cannot be implemented efficiently, they are useless. We consider the 1-norm, 2-norm and ∞-norm, and provide methods to implement geometric crossovers induced by them.

A *convex combination* is a linear combination of points (vectors) where all coefficients are non-negative and sum up to 1. The metric induced by 2-norm is the Euclidean distance. In this case, a metric segment is just the line segment that is familiar to us. The Euclidean line segment is the set of points formed by the convex combination of the two end-points of the segment.

Geometric crossover chooses offspring in the segments between parents. We can implement uniform geometric crossover UGX_2 choosing a random vector in the segment between parents as offspring. If we use the metric d_2, the segment is the convex combination of parents. We implement UGX_2 by choosing a random real value λ between 0 and 1 and computing $\lambda x + (1-\lambda)y$. Figure 4.3 shows the algorithm of UGX_2. Traditional arithmetic crossover [13] is a geometric crossover under d_2. In arithmetic crossover, the value of λ is fixed to $1/2$; it chooses only *midpoints* between parents. However, it is not uniform geometric crossover under d_2. A restricted version of line recombination proposed by Mühlenbein and Schlierkamp-Voosen [114] is the uniform geometric crossover under d_2.

The metric d_1 induced by 1-norm is the Manhattan distance. To implement geometric crossover under the Manhattan distance, we can get hints from Figure 4.1. In $\mathbb{R}^2$, the segment

```
$UGX_2(x, y)$
{
    $\lambda =$ a random uniform real number in $[0, 1]$;
    for $i = 1$ to $n$
        $z_i = \lambda x_i + (1 - \lambda) y_i$;
    return $z = (z_1, z_2, \ldots, z_n)$;
}
```

Figure 4.3: Uniform geometric crossover under d_2

between two points is a rectangle bounded by the coordinate value of each point. Theorem 4.3.1 shows that this property holds in general.

Theorem 4.3.1. $[x; y]_{d_1} = \{(z_1, z_2, \ldots, z_n) : \min(x_i, y_i) \leq z_i \leq \max(x_i, y_i),\ \forall i = 1, 2, \ldots, n\}$.

Proof. Suppose that $\min(x_i, y_i) \leq z_i \leq \max(x_i, y_i)$ for each $i = 1, 2, \ldots, n$. Then, $\sum_{i=1}^n |x_i - z_i| + \sum_{i=1}^n |z_i - y_i| = \sum_{i=1}^n (|x_i - z_i| + |z_i - y_i|)$. For each i, if $x_i \geq y_i$, $y_i \leq z_i \leq x_i$, and hence $|x_i - z_i| + |z_i - y_i| = x_i - y_i = |x_i - y_i|$. If $x_i < y_i$, $x_i \leq z_i \leq y_i$, and hence $|x_i - z_i| + |z_i - y_i| = -x_i + y_i = |x_i - y_i|$. Hence, $\sum_{i=1}^n |x_i - z_i| + \sum_{i=1}^n |z_i - y_i| = \sum_{i=1}^n |x_i - y_i|$. This implies that $z \in [x; y]_{d_1}$.

Now, suppose that $\sum_{i=1}^n |x_i - z_i| + \sum_{i=1}^n |z_i - y_i| = \sum_{i=1}^n |x_i - y_i|$. Assume that $\min(x_i, y_i) \leq z_i \leq \max(x_i, y_i)$ does not hold for all i. That is, there exists k such that $z_k > \max(x_k, y_k)$ or $z_k < \min(x_k, y_k)$.

If $z_k > \max(x_k, y_k)$, $\sum_{i=1}^n |x_i - y_i| = \sum_{i=1}^n |x_i - z_i| + \sum_{i=1}^n |z_i - y_i| = \sum_{i \neq k} (|x_i - z_i| + |z_i - y_i|) + |x_k - z_k| + |z_k - y_k| \geq \sum_{i \neq k} (|x_i - z_i| + |z_i - y_i|) + 2z_k - x_k - y_k$. If $x_k \geq y_k$, $z_k \leq x_k$. This contradicts the assumption that $z_k > \max(x_k, y_k)$. If $x_k < y_k$, $z_k \leq y_k$. This is also a contradiction.

In the case that $z_k < \min(x_k, y_k)$, we can also obtain a contradiction in a similar way. So, we showed that $\sum_{i=1}^n |x_i - z_i| + \sum_{i=1}^n |z_i - y_i| = \sum_{i=1}^n |x_i - y_i|$ for all i. □

For Theorem 4.3.1, the coordinate values of each point in a segment under d_1 are bounded by the coordinate values of any pair of endpoints of the segment. So, we can implement UGX_1, the uniform geometric crossover under d_1, by choosing a random real number between the coordinate values of the parents for each coordinate. Figure 4.4 shows this algorithm. Traditional n-point/uniform crossovers (discrete recombinations) are geometric under d_1 since they choose extreme points of the segment under d_1 as offspring. However, they are not uniform geometric crossover under d_1. A restricted version of intermediate recombination (box recombination) proposed by Mühlenbein and Schlierkamp-Voosen [114] is the uniform geometric crossover under d_1.

```
$UGX_1(x, y)$
{
    for $i = 1$ to $n$
        $z_i$ = a random real number in $[\min(x_i, y_i), \max(x_i, y_i)]$;
    return $z = (z_1, z_2, \ldots, z_n)$;
}
```

Figure 4.4: Uniform geometric crossover under d_1

In Figure 4.1, $[x; y]_{d_\infty}$ is the parallelogram the sides of which have slopes ± 1. For higher dimensional cases, we can also guess that the shape of the segment is something the surfaces (hyperplanes) of which have slopes ± 1. Theorem 4.3.2 represents $[x; y]_{d_\infty}$ formally.

Theorem 4.3.2. *Let k be the one of the indices such that $\|x - y\|_\infty = |x_k - y_k|$. If $x_k \geq y_k$, $[x; y]_{d_\infty} = \{(z_1, z_2, \ldots, z_n) : \min(x_k, y_k) \leq z_k \leq \max(x_k, y_k)$ and for each i, $|x_i - z_i| \leq |x_k - z_k|$ and $|z_i - y_i| \leq |z_k - y_k|\}$.*

Proof. Let $z \in [x; y]_{d_\infty}$. Then, $\|x - z\|_\infty + \|z - y\|_\infty = \|x - y\|_\infty$. There exist s and t satisfying $\|x - z\|_\infty = |x_s - z_s|$ and $\|z - y\|_\infty = |z_t - y_t|$, i.e., $|x_s - z_s| \geq |x_i - z_i|$ and $|z_t - y_t| \geq |z_i - y_i|$ for all i.

The above formula can be rewritten as $|x_s - z_s| + |z_t - y_t| = |x_k - y_k|$. Then, $|z_t - y_t| = |x_k - y_k| - |x_s - z_s| \geq |z_k - y_k|$. So, $|x_s - z_s| \leq |x_k - y_k| - |z_k - y_k| \leq |(x_k - y_k) - (z_k - y_k)| = |x_k - z_k|$. This implies $|x_s - z_s| = |x_k - z_k|$. Similarly, $|z_t - y_t| = |z_k - y_k|$.

$|x_s - z_s| + |z_t - y_t| = |x_k - z_k| + |z_k - y_k| = |x_k - y_k|$, i.e., $z_k \in [x_k; y_k]_{d_1}$ in $\mathbb{R}$. Hence, $\min(x_k, y_k) \leq z_k \leq \max(x_k, y_k)$ by Theorem 4.3.1. Moreover, for all i, $|x_i - z_i| \leq |x_s - z_s| \leq |x_k - z_k|$ and $|z_i - y_i| \leq |z_t - y_t| \leq |z_k - y_k|$.

Now, assume that z satisfies $\min(x_k, y_k) \leq z_k \leq \max(x_k, y_k)$ and for each i, $|x_i - z_i| \leq |x_k - z_k|$ and $|z_i - y_i| \leq |z_k - y_k|$ hold. Then, $\|x - z\|_\infty = |x_k - z_k|$ and $\|z - y\|_\infty = |z_k - y_k|$. If $x_k \geq y_k$, $\|x - z\|_\infty + \|z - y\|_\infty = |x_k - z_k| + |z_k - y_k| = x_k - z_k + z_k - y_k = x_k - y_k = \|x - y\|_\infty$. If $y_k > x_k$, $\|x - z\|_\infty + \|z - y\|_\infty = |x_k - z_k| + |z_k - y_k| = z_k - x_k + y_k - z_k = y_k - x_k = \|x - y\|_\infty$. So, $z \in [x; y]_{d_\infty}$. □

By Theorem 4.3.2, the implementation of UGX_∞ has become possible though we cannot imagine the shape of the segment $[x; y]_{d_\infty}$ in the concrete.

The algorithm for UGX_∞ is given in Figure 4.5. We first choose the index k such that $\|x - y\|_\infty = |x_k - y_k|$, i.e., $|x_k - y_k| \geq |x_i - y_i|$ for every i. Then, we choose the value of z_k between x_k and y_k. Using this value of z_k, for each coordinate, we choose z_i satisfying $|x_i - z_i| \leq |x_k - z_k|$ and $|z_i - y_i| \leq |z_k - y_k|$. This is done by calculating the minimum and maximum values that z_i can be and choosing a random real number between them.

```
$UGX_\infty(x, y)$
{
    $k = \operatorname{argmax}_{1 \le i \le n} |x_i - y_i|$;
    $z_k$ = a random real number in $[\min(x_k, y_k), \max(x_k, y_k)]$;
    for $i = 1$ to $n$
        $z_i$ = a random real number in
                $[\max(x_i - |x_k - z_k|, y_i - |z_k - y_k|),$
                    $\min(x_i + |x_k - z_k|, y_i + |z_k - y_k|)]$;
    return $z = (z_1, z_2, \ldots, z_n)$;
}
```

Figure 4.5: Uniform geometric crossover under d_∞

In the next section we present geometric crossovers for glued spaces in a formal setting.

4.4 Geometric crossovers in glued space

In general, the solution space of real-coded problems is bounded. Let $X = \{x \in \mathbb{R}^n : l_i \le x_i < u_i \text{ for each } i\}$ be the solution space where $l = (l_1, l_2, \ldots, l_n)$ is a lower bound and $u = (u_1, u_2, \ldots, u_n)$ is an upper bound. If we apply geometric crossover on this bounded domain, offspring have a bias toward the center of the space: geometric crossover produces offspring toward the center of the space with higher probability from uniformly distributed parents. This is illustrated in Figure 4.6. The abscissa represents the one-dimensional Euclidean space in the range $[0, 100]$. The uniform geometric crossover associated with this space is the operator that returns offspring uniformly at random in the interval between the two parents. The ordinate is the probability of obtaining an offspring by applying the uniform geometric crossover to parents drawn uniformly at random. Offspring closer to the centre of the space (50) have higher probabilities to be generated than offspring at the boundaries of the space (0 and 100). So, the uniform geometric crossover on this space is biased toward the centre. Interestingly, the bias of geometric crossover is not inherent in its definition, it depends on the specific metric space the geometric crossover is specialised for. In chapter 15, we will say more about the bias of geometric crossover and elicit its origin in the asymmetry (non-isotropicity) of the search space.

One method to compensate for such a bias is modifying the definition of the search operator to

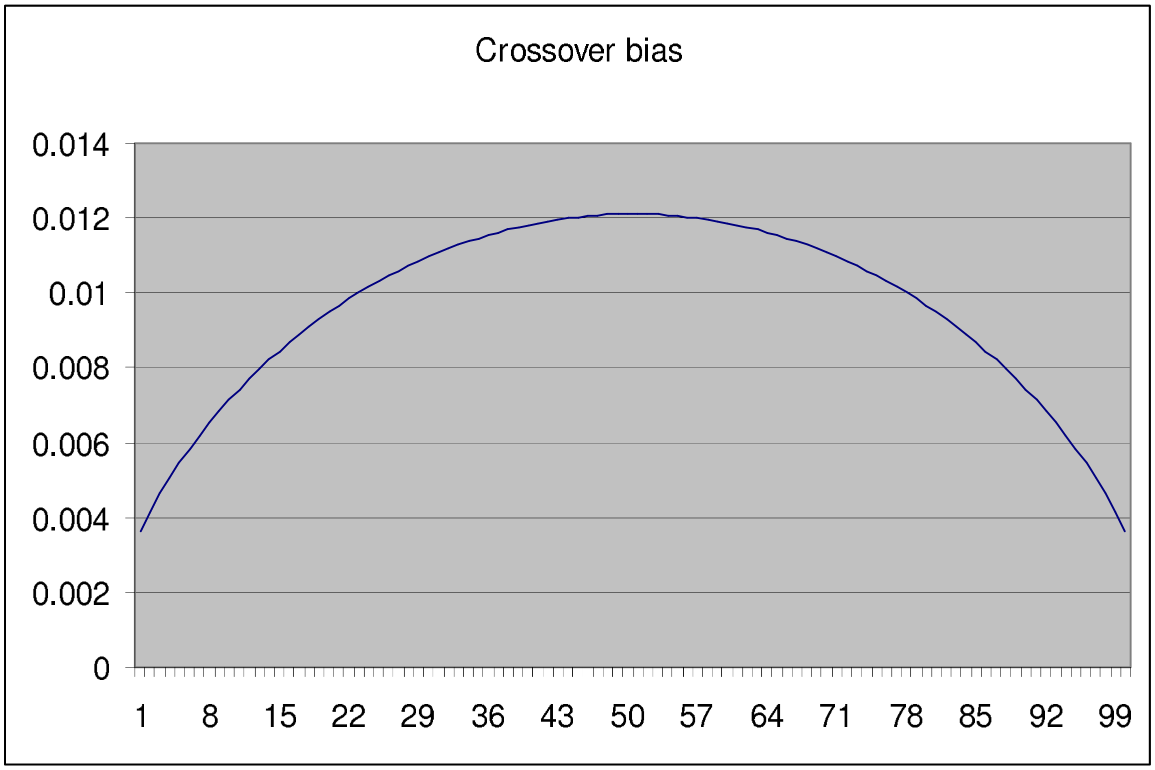


Figure 4.6: Bias toward the center of the space of geometric crossover.

pick offspring in the extended line segment passing through the parents. An alternative method to eliminate this bias consists of keeping the same operator and modifying the underlying metric space. In the following, we pursue the latter approach.

To eliminate the bias toward the center of the space we glue together all the opposite boundaries of the space (by identifying u_i to l_i for each i). Figure 4.7 shows this glued space for the case of $\mathbb{R}^2$. Intuitively, this transformation eliminates the bias toward the centre of geometric crossover because it gives rise to a completely symmetric space, the surface of a torus, in which every point can be considered as the centre (a formal result is presented in chapter 15).

Suppose X is a metric space with metric d and $\sim$ is an equivalence relation on X. The metric d_q of the *quotient metric space* on the *quotient set* $X/\sim$, which is the set consisting of all equivalence classes of $\sim$, is defined as follows:

$$d_q(x, y) := \min_{x' \in \langle x \rangle, y' \in \langle y \rangle} d(x', y').$$

Formally, the glued space is a quotient space. To construct a quotient space which gives an

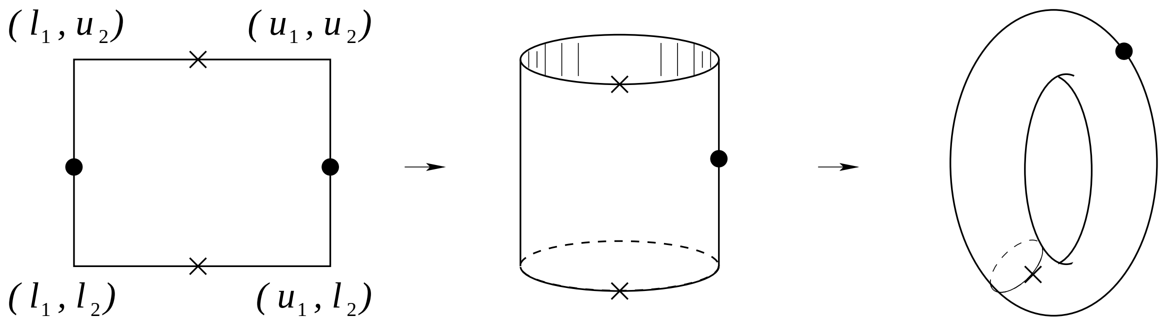


Figure 4.7: Glued space on $\mathbb{R}^2$. This can be considered as a quotient space.

effect equivalent to gluing, the following equivalence relation on $\mathbb{R}^n$ is defined:

Definition 4.4.1. $x \sim y$ if and only if for each $i = 1, 2, \ldots, n$, there exists $a_i \in \mathbb{Z}$ such that $x_i - y_i = a_i(u_i - l_i)$.

Theorem 4.4.1. *The relation $\sim$ is an equivalence relation.*

Proof. Assume that x, y and $z \in \mathbb{R}^n$.
(i) *Reflexive*: $x_i - x_i = 0(u_i - l_i)$ for each $i = 1, 2, \ldots, n$. Since $0 \in \mathbb{Z}$, $x \sim x$.
(ii) *Symmetric*: If $x \sim y$, for each $i = 1, 2, \ldots, n$, there exists $a_i \in \mathbb{Z}$ such that $x_i - y_i = a_i(u_i - l_i)$. Then, $y_i - x_i = -(x_i - y_i) = -a_i|u_i - l_i|$ and $-a_i \in \mathbb{Z}$. Hence, $y \sim x$.
(iii) *Transitive*: If $x \sim y$ and $y \sim z$, for each $i = 1, 2, \ldots, n$, there exists $a_i \in \mathbb{Z}$ such that $x_i - y_i = a_i(u_i - l_i)$ and $b_i \in \mathbb{Z}$ such that $y_i - z_i = b_i(u_i - l_i)$. $x_i - z_i = (x_i - y_i) + (y_i - z_i) = a_i(u_i - l_i) + b_i(u_i - l_i) = (a_i + b_i)(u_i - l_i)$. $a_i + b_i \in \mathbb{Z}$ since a_i and $b_i \in \mathbb{Z}$. So, $x \sim z$. $\square$

Let $\langle x \rangle$ be the equivalence class of $x \in \mathbb{R}^n$. In Figure 4.8, points indicated by bullets are in the same equivalence class on $\mathbb{R}^2$.

X can be considered as a quotient set $\mathbb{R}^n/\sim$ by considering $x \in X$ as $\bar{x} \in \mathbb{R}^n/\sim$. However, this gives another topology to the same set. We need to define a distance tailored to this new topology. We define a new distance on $\mathbb{R}^n/\sim$ using the distance on $\mathbb{R}^n$. Let $x, y \in X$.

Theorem 4.4.2. *If the distance d is induced by a norm, d_q is a metric for X.*

Proof. Assume that x, y and $z \in X$.
(i) *Identity*: $0 \leq d_q(x, x) \leq d(x, x) \leq 0$.
(ii) *Symmetry*: $d_q(x, y) = d(x', y')$ for some $x' \in \langle x \rangle$ and $y' \in \langle y \rangle$. Then, $d_q(x, y) = d(x', y') = d(y', x') \geq d_q(y, x)$. Similarly, $d_q(y, x) \geq d_q(x, y)$.
(iii) *Triangular inequality*: $d_q(x, y) + d_q(y, z) = d(x', y') + d(y'', z'')$ for some $x' \in \langle x \rangle$, $y', y'' \in \langle y \rangle$, and $z'' \in \langle z \rangle$. Since d is induced by a norm, $d(y'', z'') = d(y'' - y'' + y', z'' - y'' + y') =$

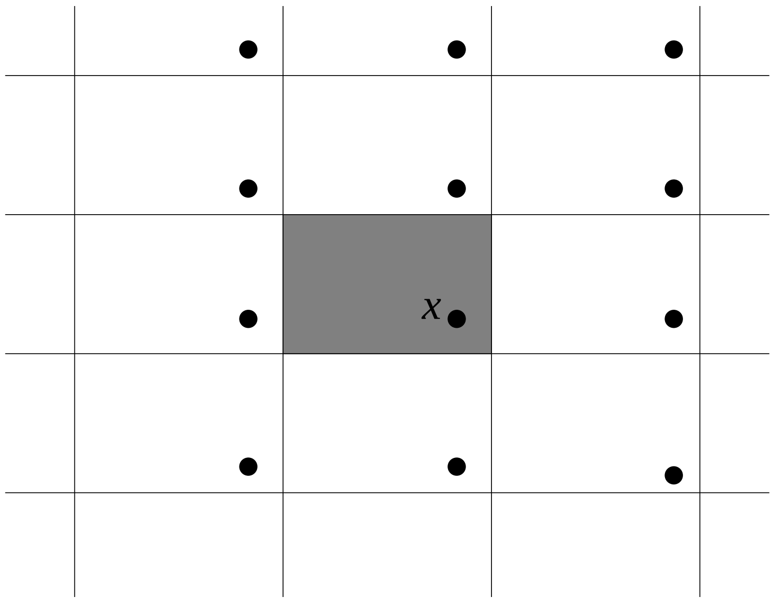


Figure 4.8: Equivalence class of x on $\mathbb{R}^2$. The shadowed rectangle represents given bounded real space X. Each rectangle has the same size as X.

$d(y', z'' - y'' + y')$. Since $y'_i - y''_i = k_i|u_i - l_i|$ for each i, $z'' - y'' + y' \in \langle z \rangle$. Hence,

$$\begin{aligned} d_q(x, y) + d_q(y, z) &= d(x', y') + d(y'', z'') \\ &= d(x', y') + d(y', z'' - y'' + y') \\ &\geq d(x', z'' - y'' + y') \\ &\geq d_q(x, y). \end{aligned}$$

□

It is impossible to calculate $d_q(x, y)$ considering all points in the equivalence classes of x and y since the number of the points is infinite. Fortunately, however, there is a practical way to calculate it.

Let $x, y \in X$. For each i, let $T_i(y) = \{y_i, y_i + (u_i - l_i), y_i - (u_i - l_i)\}$ and $M_i(y) = \underset{m \in T_i(y)}{\operatorname{argmin}}\{|x_i - m|\}$. $M_i(y)$ is a set because there can be more than one $m \in T_i(y)$ which minimises $|x_i - m|$. Let $M(y) = \{(y'_1, y'_2, \ldots, y'_n) \in \mathbb{Z}^n : y'_i \in M_i(y) \text{ for each } i\}$.

Theorem 4.4.3. *Let x and $y \in X$. If a metric d is induced by a p-norm and $y^* \in M(y)$, $d(x', y') \geq d(x, y^*)$ for all $x' \in \langle x \rangle$ and $y' \in \langle y \rangle$, i.e., d_q is well-defined and $d_q(x, y) = d(x, y^*)$. Moreover, $M(y) = \{y' \in \langle y \rangle : d_q(x, y) = d(x, y')\}$.*

Proof. $d(x', y') = (\sum_{i=1}^{n} |x'_i - y'_i|^p)^{1/p}$.

For each i, $|x'_i - y'_i| = |x'_i - y'_i + x_i - x_i + y_i - y_i| = |(x_i - y_i) + (x'_i - x_i) + (y_i - y'_i)|$. Since $x \sim x'$ and $y \sim y'$, $(x'_i - x_i) + (y_i - y'_i) = k_i(u_i - l_i)$ for some $k_i \in \mathbb{Z}$. Then, $|x'_i - y'_i| = |(x_i - y_i) + k_i(u_i - l_i)|$.

If $k_i \geq 2$, $|(x_i - y_i) + k_i(u_i - l_i)|$ is positive since $x_i - y_i$ cannot be greater than $u_i - l_i$. So, $|x'_i - y'_i| = |(x_i - y_i) + k_i(u_i - l_i)| > |(x_i - y_i) + (u_i - l_i)| = |x_i - \{y_i + (u_i - l_i)\}| \geq |x_i - y^*_i|$. Similarly, in the case that $k_i \leq -2$, $|(x_i - y_i) + k_i(u_i - l_i)|$ is negative. Hence, $|x'_i - y'_i| = |(x_i - y_i) + k_i(u_i - l_i)| > |(x_i - y_i) - (u_i - l_i)| = |x_i - \{y_i - (u_i - l_i)\}| \geq |x_i - y^*_i|$. Finally, if $k_i = 0, 1, -1$, $|x'_i - y'_i| = |(x_i - y_i) + k_i(u_i - l_i)| \geq |x_i - y^*_i|$ by the definition of $M(y)$.

So, $d(x', y') = \{\sum_{i=1}^n |x'_i - y'_i|^p\}^{1/p} \geq \{\sum_{i=1}^n |x_i - y^*_i|^p\}^{1/p} = d(x, y^*)$.

Now, we will show that $M(y) = \{y' \in \langle y \rangle : d_q(x, y) = d(x, y')\}$. Suppose that $y' \in \langle y \rangle$ satisfies $d_q(x, y) = d(x, y')$ but $y' \notin M(y)$. Then, there exists a nonempty index set $J = \{j : |x_j - y'_j| \neq |x_j - y^*_j|\}$. For each $j \in J$, $|x_j - y'_j| = |x_j - y_j - k_j(u_j - l_j)|$. If $k_j \geq 2$ or $k_j \leq -2$, $|x_j - y'_j| > |x_j - y^*_j|$ by the same way as the above. If $k_j = 0, 1, -1$, $|x_j - y'_j| \geq |x_j - y^*_j|$. By the assumption, $|x_j - y'_j| \neq |x_j - y^*_j|$ and hence $|x_j - y'_j| > |x_j - y^*_j|$. Therefore, $d(x, y') = (\sum_{i=1}^n |x_i - y'_i|^p)^{1/p} > d(x, y^*)$ and it is contradiction. □

According to Theorem 4.4.3, to calculate $d_q(x, y)$, we need only to find y^* by choosing a minimizer among the three elements of $T_i(y)$ (for each coordinate) and get the Euclidean distance between x and y^*. Now, the segment between x and y on the quotient space is induced by the segment between x and y^* on $\mathbb{R}^n$.

Theorem 4.4.4. *If d is induced by a p-norm, the line segment $[x; y]_{d_q}$ is the set*

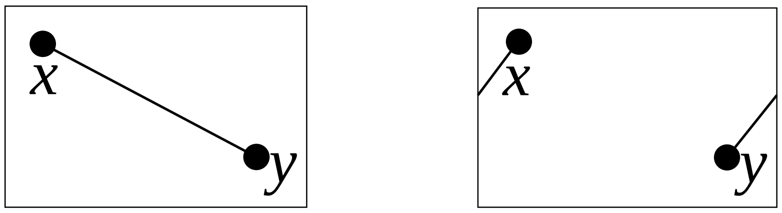
$$\bigcup_{y' \in M(y)} \{z \in X : z \sim z' \text{ for some } z' \in [x; y^*]_d\}.$$

Proof. Let $y^* \in M(y)$ and $z \sim z'$ for some $z' \in [x; y^*]_d$. Then, $d_q(x, z) + d_q(z, y) \leq d(x, z') + d(z', y^*) = d(x, y^*) = d_q(x, y^*)$. Since $d_q(x, z) + d_q(z, y) \geq d_q(z, y)$ by triangular property, $d_q(x, z) + d_q(z, y) = d_q(z, y)$. So, $z \in [x; y]_{d_q}$.

Now, let $z \in [x; y]_{d_q}$. There exist z' such that $d_q(x, z) = d(x, z')$ and y' such that $d_q(z, y) = d(z, y')$ in $\mathbb{R}^n$ by Theorem 4.4.3. Let $y^* := y' - z + z'$. Then, $y^* \sim y' \sim y$ and $d_q(x, y) = d_q(x, z) + d_q(z, y) = d(x, z') + d(z, y') = d(x, z') + d(z', y' - z + z') = d(x, z') + d(z', y^*) \geq d(x, y^*)$ since d is a metric. By the definition of d_q, $d_q(x, y) \leq d(x, y^*)$ is true and hence $d_q(x, y) = d(x, y^*)$. This implies $y^* \in M(y)$ and $d(x, z') + d(z', y^*) = d(x, y^*)$. □

Figure 4.9 (A) shows the segment on Euclidean space and (B) shows the segment on glued space (quotient space). In the quotient space, segments may cross the boundaries.

It could be argued that glued spaces remove the inherent bias of geometric crossover at the cost of introducing discontinuity at those points where the opposite sides of the space are glued together. However, one could also argue that the discontinuity is already present in the original fitness landscape because at the boundaries of the space the fitness landscape is disconnected, hence discontinuous. The gluing operation just rearranges the location of the original discontinuity.



(A) $[x; y]_{d_2}$ (B) $[x; y]_{d_q}$

Figure 4.9: Line segments on Euclidean space and glued space

A way to remove this discontinuity altogether is gluing the opposite sides of the fitness landscape to the opposite sides of a mirror image of itself. The new landscape is continuous at all gluing points and its search space is isotropic. The cost of this operation, however, is doubling the size of the search space for each dimension. Nevertheless, this does not necessarily implies that the new landscape would be harder to search because of its rise in size. In fact, the ratio of good solutions to bad solutions remains the same as in the original landscape.

4.5 Geometricity of pre-existing crossovers

In this section, we consider pre-existing operators for real-coded representations. Also, we check whether they are geometric operators and if so for what distances.

It is immediate to see that line recombination and box recombination are geometric crossovers under Euclidean and Manhattan distance respectively because offspring are in the segments between parents under these two distances.

Discrete recombination is geometric under Manhattan distance because it can be seen as a special case of box recombination in which offspring are at the corners of the hyper-box identified by the parents. It is easy to verify that discrete recombination is also geometric under Hamming distance extended to real vectors. This distance is simply the number of positions containing different values in the two vectors.

Are extended-line recombination and extended-box recombination geometric? Let us consider the case of extended-line recombination. Obviously this recombination is not a geometric crossover under the Euclidean distance, because of the possibility of generating offspring outside the Euclidean segment between parents. However, this does not exclude that the extended line recombination may be geometric under some other notion of distance. That indeed would make it a geometric crossover. So, wouldn't it be possible to do some topological transformation of the underlying real space, a gluing for example, and find a space endowed with a distance for which segments are extended segment in the Euclidean space? Even if this seems to be a promising way of approaching the problem, it actually leads nowhere. In chapter 14 we will show axiomatically that there is no distance under which extended-line crossover is geometric. Hence, there is no topological transformation of the space that does the trick. So, extended-line crossover is not a geometric crossover. The extended-box recombination can be proven to be non-geometric analogously.

The results for real vectors extend immediately to integer vectors. Since integer vectors can be understood as a subsets of real vectors, cardinal and ordinal recombinations between integer vectors can be seen as special cases of, respectively, discrete and box recombinations. So, they are geometric crossovers under Hamming and Manhattan distances, respectively.

The geometricity of mutation operators for the real-coded representation is immediate. Creep mutation is the uniform geometric a-mutation under d_∞. Single-variable mutation is a geometric a-mutation under any Minkowski distance. Gaussian mutation is a geometric δ-mutation under any Minkowski distance, where δ is the semi-diameter of the search space.

4.6 Summary and discussion

In this chapter we have applied the geometric framework presented in chapter 3 to the real-vector representation. We have made the following contributions:

- Although the definition of geometric crossover is fairly intuitive for the case of the Euclidean metric, other distances are suitable for real vectors. We have studied formally and in a very general setting geometric crossovers for the family of Minkowski metrics and given efficient algorithms to implement them.

- We have seen that geometric crossover specified for Minkowski metrics is a biased operator:

it produces offspring toward the center of the space with higher probability from uniformly distributed parents. The origin of this bias has to be found in the fact that these spaces are non-isotropic.

- Minkowski spaces can be made isotropic by gluing the opposite sides of the search box (or interval) in $\mathbb{R}^n$ together and considering the distances associated with these glued spaces. We have then studied formally and in full generality the unbiased geometric crossovers associated with these new spaces.

- We have also shown that a number of pre-existing recombination operators for real vectors are geometric crossovers under some Minkowski distances.

In this chapter we have built new geometric crossovers associated with non-traditional metrics for continuous spaces with the aim of removing search biases. Another important reason to consider non-traditional metrics associated with real spaces is their potential use in specific applications. For example, we could consider vectors as polar coordinates instead of Cartesian coordinates and define metrics and geometric crossovers suited to them. These crossovers are likely to perform well on problems that are naturally expressed in terms of polar coordinates such as problems naturally defined on a sphere.

In general, building geometric operators associated with non-standard metrics for the real-coded representation well-suited to specific problems is a potentially powerful and straightforward way of embedding problem knowledge in the search. This possibility has been completely neglected in continuous optimization where the Euclidean distance has almost always been implicitly chosen.

Chapter 5

Permutations

In previous chapters we have shown how geometric crossover and geometric mutation generalise traditional crossover and mutation operators for binary strings and real vectors. In this chapter we explore how the geometric framework applies to the permutation representation (see [51] for an introduction). This is one of the most-frequently used representations in EAs. Many combinatorial optimisation problems are naturally cast using permutations.

The permutation representation is challenging for various reasons: because of the inherent difference between permutations and the representations that inspired the abstraction; because the whole notion of geometry over permutation spaces radically departs from traditional geometries and it is almost unexplored mathematical territory; because the many notions of distance available and their subtle interconnections make it hard to see the right distance to use; because the various interpretations of permutations available make the meaning of a permutation ambiguous, and it is, therefore, difficult to decide how to treat it; because of the existence of various permutation-like representations that are incorrectly confused with permutations; and, finally, because of the existence of many mutation and recombination operators and their many variations for the same representation. This chapter shows that the application of our geometric framework naturally clarifies and unifies the important domain of the permutation representation and the related operators, in which there was little or no hope to find order. In addition, the abstract geometric framework is used to improve the design of crossover operators for well-known problems naturally connected with the permutation representation.

5.1 Introduction

When applied to permutations, traditional crossover operators can produce invalid offspring. So, researchers have proposed a variety of operators specifically designed for permutations. They range from general-purpose operators working reasonably well on a wide spectrum of problems, such as the Partially Matched Crossover [53], to specialised operators that work best on a specific class of problems [42], such as Edge Recombination Crossover for TSP [163].

In chapter 3 we have shown how geometric crossover generalises the notion of crossover for binary strings. Differently from the binary string case, for which a single natural distance (the Hamming distance) is defined, like real vectors, permutations allow for various notions of distance that are all equally natural.

In this chapter we will do the following. Section 5.2 introduces various notions of distance for permutations. Section 5.3 focuses on distances based on permutation interpretation and discusses the difficulties of using geometric crossovers based on such distances. Section 5.4 introduces various edit distances and draws a parallel between "interpretation" distances and edit distances. Section 5.5 shows that geometric crossover for permutations is naturally suited for edit distances and it is intimately connected with the notion of sorting algorithm. Section 5.6 reviews the most frequently used recombination operators for permutations and highlights their affinity with different permutation interpretations. Section 5.7 shows that a number of pre-existing recombination operators for permutations are in fact geometric crossovers under different edit distances. Section 5.8 presents a new geometric crossover that extends the cycle crossover to permutations with repetitions. Section 5.9 presents an experimental comparison on the n-queens problem of a number of geometric crossovers based on three classical sorting algorithms: selection sort, bubble sort and insertion sort. Section 5.10 presents an analysis of geometric crossover for the TSP problem, and reports experimental results that corroborate the analysis. The proposed geometric crossover for TSP beats edge recombination, which is one of the best crossover for this problem.

5.2 Metrics on permutations

Unlike binary strings where a single, natural definition of distance, the Hamming distance, is normally used, for permutations many notions of distance are equally natural. This situation is complicated by the fact that such distances relate to each others in various ways with subtle dependencies. A further complication arises from the fact that permutations and circular permutations (and also permutations with repetitions to a lesser extent) are treated as if they were the same representation, which is incorrect. Indeed, although they are connected, these are different representations and they allow for different notions of distance. For a survey of metrics on permutations see [29].

Distances for permutations have different origins:

1. Notions of distance arising from the *interpretation* of permutations: these measure the distance between the objects represented by two permutations.

2. Notions of distance directly connected with the *syntax*: these distances measure how two permutations differ in their syntax.

3. Notions of distance connected with the notion of mutation or *edit* distance: these distances measure the minimum number of moves necessary to transform a permutation into another by the application of a syntax modification operator.

These three types of distances are related. For example, an edit distance is also a syntactic distance but a syntactic distance is not necessarily an edit distance (one can define a measure of syntactic similarity that is not defined on edit moves). Also, an edit distance is also a graphic distance (see chapter 3), but a graphic distance is not necessarily an edit distance.[1] In the following three sections, we study the connections between these three notions of distance and their suitability as bases for geometric crossover.

[1]It can be argued that any graphic distance is an edit distance under an arbitrary set of edit operations. However, we consider "simple" edit distances based on a small set of moves with a compact syntactic definition.

In principle, given *any* notion of distance over the solution set, the corresponding geometric crossover and geometric mutation operators are well-defined. What is a good distance then? A good distance is one that (i) gives rise to a fitness landscape (i.e., a solution set with its fitness function plus a notion of distance) easy to search for the specific search operator employed[2] and that (ii) allows such an operator to be implemented efficiently. The first point connects with *landscape design*, the second with *distance duality.* Here we concentrate on the latter.

Geometric crossover and mutation are well-defined for any notion of distance (whether based on syntax or not). So, operators are well-defined independently from the underlying representation. In practice, however, the genetic operators must be implemented. If they are not based on a notion of edit distance linking them tightly to the solution representation, they become difficult or even *impossible to implement efficiently.* To understand the reasons for this we need to recall from chapter 3 the notion of *distance duality.* The notion of edit distance arising from the syntax of configurations has a natural dual interpretation:

1. seen in the configuration space, it is a measure of *similarity* (or dissimilarity) between two syntactic objects;

2. seen in the neighbourhood graph, it is the *length of the shortest path* connecting two vertices and, therefore, it is a measure of *spatial remoteness* between points when interpreting such a structure in a *geometric sense.*

For each representation and edit move definition, this duality manifests itself in a different way. In the case of permutations the duality implies that picking elements in the segment (shortest path) between parents is equivalent to picking elements on a *minimal sorting trajectory* from one parent permutation to the other. This connection between the geometric notion of "belonging to a segment" and its syntactic dual of "being on a minimal sorting trajectory" is ultimately what allows geometric crossover to be *actually implemented* in an efficient way. So, even if a geometric crossover is representation-independent, when dealing with its implementation, the

[2]As we have seen in chapter 3, a rule of thumb to pick a good distance as a base for geometric crossover is to choose a distance that is meaningful for the interpretation of the solution representation (phenotype).

specific representation used makes indeed the difference between an *efficient* operator and an *inefficient* one.

5.3 Permutation interpretations and related distances

To fully understand distances for permutations, we cannot separate them from permutation interpretations. Permutations can be used to represent solutions to different types of problems for which different relations among the elements in the permutation are relevant. There are three major interpretations of a permutation [8]. For example, in TSP, permutations represent tours and the relevant information is the *adjacency relation* among the elements of a permutation. In resource scheduling problems, permutations represent priority lists and the relevant information is the *relative order* of the elements of a permutation. In other problems, the important characteristic is the *absolute position* of the elements in the permutation.

Let us consider the permutation (345261) that can be thought as being produced by shuffling at random the elements of the identity permutation (123456). If adjacency is important then the fact that elements 4 and 5 are adjacent is relevant as is the fact that elements 3 and 2 are not. If the important aspect is relative order then what is relevant is that 4 precedes 5 and that 3 precedes 2. If absolute order is important then the relevant point is that 3 is at position 1, 4 at position 2, etc.

For each permutation interpretation, it is possible to write a binary matrix that represents the binary relation among elements in the permutation associated with that specific interpretation. So, we can have a relative order matrix (ROM), an absolute position matrix (APM) and an adjacency matrix (AM). For example, for the permutation (345261) we have the following matrices:

$$ROM = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$APM = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$AM = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The matrix ROM describes the binary relation of strict relative order precedence among all elements in the permutation. In the matrix, there is an entry for each possible pair of elements of the permutation. The entry is 0 if the element associated with its row does not precede in the permutation the element associated with its column. Otherwise the entry is 1.

The matrix APM describes the binary relation of absolute position of the elements in the permutation. In the matrix, rows are associated with positions of the permutation, columns are associated with elements of the permutation. The entry is 0 if the associated element with its column is not at the position associated with its row. If it is, the entry is 1.

The matrix AM describes the binary relation of adjacency of the elements in the permutation. In the matrix, there is an entry for each possible pair of elements of the permutation. The entry is 0 if the elements associated with its row and column are not adjacent in the permutation.

Otherwise the entry is 1.

It is possible to define three distance functions for permutations based on relative order matrices, absolute position matrices and adjacency matrices. The distance between two permutations is then the Hamming distance between their corresponding matrices in the three interpretations. We refer to these distances as *relative order distance* (ROD), *absolute position distance* (APD) and *adjacency distance* (AD).

For example, the ROM matrices associated with the permutations (345261) and (123456) are, respectively, the ROM matrix provided in the example above and a triangular matrix filled with 1 in all entries above the main diagonal, and 0 elsewhere. The ROD distance between these permutations is the Hamming distance between these two matrices, which is 16. That gives a measure of their diversity in terms of relative order of their elements. The minimum ROD distance is always zero; the maximum ROD distance for permutations occurs between two permutations with completely reversed order, which, for permutations with six elements, is 30.

Theorem 5.3.1. *ROD and APD are metrics for permutations. AD is a pseudo-metric for permutations and a metric for reversible permutations.*

Proof. For ROD: Let us consider the subset **ROM** of all binary matrices of dimension $n \times n$ that describe valid binary relations of strict relative order precedence among all elements of any permutation of size n. The Hamming distance is a metric over the set of all binary matrices of dimension $n \times n$. Since, **ROM** is a subset of the set of all binary matrices of dimension $n \times n$, then the Hamming distance is a metric on such a restricted domain too. It is easy to see that the function that given any permutation of size n returns its associated ROM is a bijection, because given the ROM one can identify uniquely its associated permutation. Then, since the Hamming distance on **ROM** is a metric, the same distance is a metric on the set of permutations of size n.

For APD, the same argument holds.

For AD: the function that given any permutation of size n returns its associated AM is not a bijection, because the same matrix can be associated with exactly two permutations, one with the exact opposite order of the elements of the other one. This implies that the AD distance between these two distinct permutations is zero. This contradicts the axiom of identity of a metric. So, AD is not a metric on permutations. However, if we consider a permutation and its reversed as being the same object, a reversible permutation, on the set of reversible permutations, the function associating reversible permutations to their AM matrices is a bijection. So, in this case, the same argument of ROD holds. Hence, AD is a metric on reversible permutation and a pseudo-metric on permutations. □

Once we have these distances, we have a notion of segment. For example, the segment between two permutations p_1 and p_2 under ROD includes all the permutations whose ROM are between the ROMs of p_1 and p_2. This means that this segment includes all those permutations whose relative order relation among their elements is compatible with the common relative order among the elements of p_1 and p_2.

Continuing the example above, the permutation $p_3 = (341256)$ is in the ROD-segment between $p_1 = (345261)$ and $p_2 = (123456)$. This can be verified by computing the ROM for p_3, which is:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Since $ROD(p_1, p_2) = 16$, $ROD(p_1, p_3) = 8$ and $ROD(p_3, p_2) = 8$ then $p_3 \in [p_1, p_2]_{ROD}$.

Geometric operators can be defined using these notions of distance. So we can define *rigorously* relative order geometric crossover (ROX) and mutation (ROM), absolute position geometric crossover (APX) and mutation (APM), and adjacency geometric crossover (AX) and mutation (MX). ROX transmits perfectly to the offspring permutations the common relative order relation among elements of parent permutations. APX transmits perfectly to the offspring permutations the common absolute positions of the elements of parent permutations. AX transmits perfectly to the offspring (reversible) permutations the common adjacency relation of the elements of parent (reversible) permutations.

Let us consider the ROX crossover. This crossover could be implemented by recombining the ROMs of the parents permutations using a simple extension to matrices of the traditional crossover for binary strings and converting the offspring ROM into the corresponding offspring permutation. The price to pay to recombine permutations in this way is that offspring matrices are not guaranteed to be valid/feasible ROMs (i.e., binary matrices) corresponding to permutations. Hence, one may need to deal with this form of infeasibility, for example, by using some form of repair mechanism. Note that in this case the repaired offspring is not guaranteed anymore to cover the common relative order of the parents permutation perfectly. Analogous considerations hold for APX and AX crossovers.

The infeasibility problem arises from the fact that the distances considered in this section as basis for geometric crossover are not directly based on permutations, but they are based on an auxiliary matrix representation. In the next section, we consider distances defined directly on the "syntax" of permutations, edit distances, for which the infeasibility problem does not exist.

5.4 Edit distances and mutations

Various mutation operators are defined for permutations. The most common are [8]:

Inversion or 2-change (block-reversal): This operator selects two points along the permutation then reverses the segment between the points. It is particularly well-suited for the TSP and for all the problems that naturally admit a permutation representation in which adjacency among elements plays an important role.

Insert and block-transposition: This operator selects one element and inserts it at some other position in the permutation. There is a variant: one selects two elements and then moves the second element before the first. This move is irreversible because it is not possible to return to the previous permutation by a single further application of this move. These operators are used with scheduling problems in which relative order of elements is important.

Swap and adjacent swap (two-element swap): The swap operator selects two elements and swaps them. The adjacent swap swaps two contiguous elements.

Scramble: This operator selects a sublist of elements of the permutation and randomly reorders the elements while leaving the other elements in the permutation in the same absolute position.

Since each notion of mutation is naturally connected with a notion of edit move, it is also connected with a notion of edit distance for permutations. However, even if the notion of mutation is linked to the notion of edit move, they do not coincide: an edit move is a deterministic and unitary syntactic transformation, whereas a mutation is a non-deterministic operator with a given probability distribution and it does not need to correspond all the times to a single edit move (a mutation can generate offspring more than one edit move away from their parent).

A valid edit move must be symmetric and connected (see chapter 3). All the mutation operators listed above, except for the irreversible variant of the insert mutation, are associated with edit moves that are symmetric and fully connected. For example, the inversion operator is symmetric (re-reversing the same sub-list produces the original permutation) and connected (by repeated reversions it is possible to reach any permutation from any other permutation). Also the adjacent swap operator is symmetric and connected (bubble sort based on adjacent swap is able to sort any permutation of elements). The same holds for the swap operator.

Therefore, we can talk of *reversal distance*, *transposition distance*, *swap distance*, *adjacent swap distance*, *scramble distance* and so on. Notice that there are a number of variations for each of these distances which result from imposing constraints on the edit move.

5.4.1 Relations between edit distances and interpretation distances

Depending on the interpretation of the permutation, the same mutation can be seen as a small change or a major change. For example, the inversion operator does a minimal change when one thinks of a permutation in terms of adjacency, but a major change when the same permutation is seen as a priority list (relative order).

As we have seen in chapter 3, a single mutation should represent a minimal change at a phenotypic level to be likely to give rise to a smooth fitness landscape. According to this principle, there are three mutation operators that do a different minimal change in a permutation, one for each interpretation. When the permutation is thought as an adjacency relation then the minimal mutation operator is the inversion operator: while reversing the order of a sub-list, only two adjacency links (edges) are changed. When the permutation represents a relative order the minimal mutation operator is the adjacent swap operator that affects only the relative order of a pair of elements. When the absolute position of elements in the permutation is relevant, the minimal mutation operator is the swap operator that changes the absolute positions of only two elements.

5.5 Sorting crossovers

For each notion of edit distance there is a corresponding notion of geometric crossover. So, we can define many possible crossovers for permutations, each induced from a corresponding mutation. Since these are crossovers based on similar, but not identical, neighbourhood structures, they will tend to have similar behaviours. Not all geometric crossovers based on edit distances have efficient implementations. Indeed, constraints on edit moves transform the complexity of computing the distance, hence of implementing the corresponding crossover, from polynomial to NP-hard [145].

Because of the distance duality, a point on a segment between two permutations, under a given edit distance, is on a minimal sorting trajectory connecting the two permutations. This allows to actually implement such geometric crossovers based on edit distances by *sorting algorithms*. Some edit distances give rise to crossovers that can be implemented exactly and efficiently. Others give rise to crossovers that are possible to implement efficiently (in polynomial time) only in an approximated way. Quite interestingly, bubble sort and insertion sort fit the definition of geometric crossover for, respectively, the adjacent swap distance and the swap distance. So, *ordinary sorting algorithms can actually be used as crossovers*!

Not all classic sorting algorithms can be used as a base for geometric crossover though. Only those algorithms that use the same move throughout the sorting, and that are guaranteed to sort always using the minimum number of move applications can be used. Technically, these algorithms when applied to permutations solve the “minimal permutation sorting by x problem” [157] where x stands for the move used. Bubble sort, insertion sort and selection sort belong to this class of sorting algorithms, for which the sorting move, respectively adjacent swap, insertion and swap, is pre-specified and fixed over the whole execution of the algorithm. Some more effective sorting algorithms, such as quick sort, use different moves while progressing with the sorting, so they cannot be used as base for geometric crossovers.

In the following we highlight some important differences between sorting algorithms and crossover operators for permutations based on edit moves:

1. Sorting algorithms sort permutations into the fully ordered permutation $(123\cdots)$. Crossover operators sort one parent permutation toward the order of the other parent permutation, which can be any permutation. However, sorting algorithms can be easily adapted to sort any permutation into any other (by renaming the elements of both permutations).

2. Traditional sorting algorithms are designed to sort vectors of any type of objects coming with a well-defined notion of order between objects: they can sort permutations, as well as real vectors, list of words, etc. Crossover operators sort permutations, which are special in that the ordering rank of each element is already known, and it is specified by the element itself (for example, the element 2 is to be placed in position 2 in the ordered permutation). This additional information can be used to build different and more efficient sorting algorithms for permutations that do not resemble the classical ones.

3. Traditional sorting algorithms are designed to use the minimal number of moves (exchanges between elements), and also to do the minimal number of comparisons between elements. Crossover operators, like the traditional sorting algorithms, require sorting using the minimal number of moves. However, crossover operators do not require optimality in the number of comparisons.

4. The purpose of sorting algorithms is to sort the elements of a vector. The purpose of crossover operators is, instead, to return an offspring permutation on the minimal sorting trajectory between parent permutations. This can be done by interrupting the sorting algorithm at a random point and returning the partially sorted permutation as offspring.

5. There are two distinct categories of sorting algorithms: deterministic and non-deterministic. Deterministic sorting algorithms perform the sorting always in the same way given the same permutation to sort. For example, deterministic bubble sort scans the permutation always from left to right and applies the first adjacent swaps that make closer the current permutation to the complete ordered one. Non-deterministic sorting algorithms are randomised sorting algorithms that do not select the next move deterministically but according to some randomised strategy. For example, the uniform non-deterministic sorting strategy for bubble sort is scanning the current permutation considering all the sorting moves (those that get the current permutation one move closer to the complete order), and then selecting one at random and applying it. Crossover operators may benefit form non-deterministic sorting. That is, given the same permutation to sort, crossover may return offspring on different minimal sorting trajectories in different executions.

5.5.1 Geometric operators implementation

In the following, we present two simple and efficient algorithms that implement geometric crossover and geometric mutation for permutations based on edit distances.

Sorting crossovers

In Algorithm 1 we report the pseudo-code for a generic sorting crossover based on any edit move. Firstly, the elements of parent1 are renumbered (normalized) according to the order of the elements in parent2. This is done by permutation composition of parent1 with the inverse permutation of parent2. This transformation allows us to reduce the task of sorting one permutation toward a second arbitrary permutation into the standard task of sorting a permutation into a completely ordered permutation. The following step is obtaining the sequence of edit moves that order the normalised parent1 on a minimal sorting trajectory. This can be done by modifying any sorting algorithm that satisfies the requirement of being a “minimal sorting by x” algorithm in a way that, while sorting, it collects the sequence of sorting moves it uses, and it returns it. The distance between the two parent permutations based of the edit move considered is the number of moves on the sorting trajectory between the two permutations returned by the modified sorting algorithm. A crossover point is then selected at random on the sorting trajectory and the offspring permutation is obtained by applying the sequence of moves to the first parent permutation until the crossover point has been reached.

Algorithm 1 Sorting crossover

```
 1: Input: parent1, parent2 Output: offspring
 2: normalised_parent1 = compose(parent1, inverse(parent2))
 3: moves_sequence = sort(normalised_parent1)
 4: distance = length(moves_sequence)
 5: crossover-point = uniform_random(integer_range[0, distance])
 6: offspring = parent1
 7: while crossover-point > 0 do
 8:     offspring = compose(offspring, moves_sequence[crossover-point])
 9:     crossover-point = crossover-point - 1
10: end while
11: return offspring
```

We obtain two different types of crossover depending on the fact we use a deterministic or a non-deterministic sorting algorithm as base for crossover. The offspring of a deterministic sorting crossover all lie on the same geodesic between their parents. The offspring of a non-deterministic sorting crossover, instead, cover all segment between their parents (because the union of all geodesics connecting two points coincides with the segment between them). In this respect, the deterministic sorting crossover resembles the traditional one-point crossover for binary strings (see chapter 13), whereas the non-deterministic sorting crossover resembles the uniform crossover. Notice, however, that the actual probability distribution of the offspring over the segment is not necessarily uniform, and it depends on the specific geometry of the space induced by the specific edit move considered.

Edit mutation

In Algorithm 2 we report a generic geometric mutation that can be used with any solution representation coming with a notion of edit distance (not only with permutations). The parameter p_m is the mutation probability. The neighbours of a given syntactic configuration are all those syntactic configurations within the reach of a single edit move. The mutation operator can reach any point in the space from any other with an exponentially decreasing probability for increasing distance.

Algorithm 2 Mutation operator

```
1: Input: parent, mutation probability $p_m$ Output: offspring
2: offspring = parent
3: while $p_m$ > uniform_random([0, 1]) do
4:     neighbours = get_neighbours(offspring)
5:     offspring = uniform_random(neighbours)
6: end while
7: return offspring
```

5.6 Existing crossovers and permutation interpretations

There are a number of crossover operators defined for permutations. For a good overview see [8] and [9] that contain the descriptions of the crossovers considered in this section and the next ones. Most of them were devised with a *specific interpretation* of the permutation in mind. This is reflected in their names. So, for example, Davis's *order crossover* [25] emphasises the fact that a permutation is seen as a relative order, *cycle crossover* [117] preserves absolute positions, and *edge recombination crossover* [163] focuses on the adjacency relation of the elements in the permutation.

Some crossovers achieve their goals of transmitting a specific relationship among elements from the parents to the children perfectly (*perfect crossovers*), others achieve their goals only approximately (*imperfect crossovers*). For example cycle crossover transmits perfectly the common positional information of parents to children and so is a perfect crossover. Both Davis's order crossover and edge recombination are imperfect crossovers in that they are not able to transmit perfectly the common relative order of the parents and the adjacency relation, respectively. However, the common relative order is much easier to transmit perfectly than the adjacency relation (because the distance associated with the adjacency relation, reversal distance, is NP-Hard to compute). Indeed, another crossover, the merge crossover [11], perfectly transmits the relative order of parents to children.

Some crossover operator is deliberately designed to be a trade-off, transmitting part of the relative order, part of the absolute position and part of the adjacency relation present in the parent permutations to the offspring permutations (*hybrid crossovers*). This is indeed possible since the three relations have subtle interdependencies. One of such crossover operators is the partially mapped crossover (PMX) [53]. Hybrid crossovers have the advantage to work reasonably well independently from the specific interpretation of the permutation. However, when hybrid crossovers are compared with perfect crossovers for a specific interpretation of the permutation on a problem in which this interpretation is relevant, the hybrid ones tend to perform worse than the perfect ones.

5.7 Geometricity of pre-existing crossovers

In this section, we consider some of the most frequently used recombination operators for permutations and analyse whether they are geometric crossovers and, if so, under what distances.

In section 5.5, we showed that geometric crossovers for permutations under edit distances are naturally associated with sorting algorithms. This allows us to implement these crossovers using sorting algorithms. However, using sorting algorithms is not the only way to implement these crossovers. *Interestingly, pre-existing crossovers for permutations that fit the definition of geometric crossover under edit distances are also associated with sorting algorithms even if they may not look so.* Any crossover for permutations which is found to be geometric under edit distance produces offspring on the minimal sorting trajectory between parents according to some edit move. The reason why such a crossover may not look like a sorting crossover is because it picks offspring on a minimal sorting trajectory without generating explicitly the permutations on this trajectory. Instead, it generates directly offspring permutations by a single syntactic manipulation of the parent permutations equivalent to the application of a sequence of single edit moves on a minimal sorting trajectory between parents. This is made possible by the special property of permutations with regard to sorting (see section 5.5). Those of the recombination operators for permutations analysed in the following, which are geometric crossovers, are sorting crossovers of the type just described.

5.7.1 Order crossover

Davis's order crossover [25] was expressly designed to transmit information about the relative order. Figure 5.1 shows an example of this crossover. It begins by copying a randomly chosen segment of parent 1, the crossover section, into the offspring. Then, starting from the second crossover point in parent 2, it copies the remaining unused numbers into the offspring in the order they appear in parent 2, wrapping around at the end of the list.

```
Parent 1: A B . C D E F . G H I
Crossover section: _ _ C D E F _ _ _

Parent 2: h d . a e i c . b f g
Available elements in order: b g h a i

Offspring: a i C D E F b g h
```

Figure 5.1: Example of Davis's order crossover.

Davis's order crossover is a hybrid crossover in that tends to preserve, only partially, all the three relations (adjacency, relative order and absolute position) among elements. We illustrate this in the following. The elements in the crossover section preserve relative order, absolute position and adjacency from parent 1. The elements copied from parent 2 do not preserve neither position nor adjacency. In the example in figure 5.1, position is not respected because, for example, at position two the offspring differs from both its parents. Adjacency is not respected either: in both parents B (b) and C (c) are adjacent, but they are not adjacent in the offspring. Also, the relative order information contained in the parents is passed to the offspring only partially. In the example, H (h) precedes I (i) in both parents, but in the offspring i precedes h. In chapter 14, we will prove that Davis's order crossover is not a geometric crossover.

5.7.2 Partially mapped crossover

Partially mapped crossover (PMX) was proposed by Goldberg and Lingle [53]. Figure 5.2 shows an example of this crossover.

```
Parent 1: A B . C D E . F G
Crossover section: _ _ C D E _ _

Parent 2: c f . e b a . d g

Offspring: _ _ C D E b _
Offspring: a _ C D E b _
Offspring: a f C D E b g
```

Figure 5.2: Example of partially mapped crossover.

Like order crossover, PMX begins by copying a randomly chosen segment of parent 1, the crossover section, into the offspring. Then, consider the elements between the crossover points in parent 2 which have not already been copied to the offspring. In the example, these elements are b and a. For each of those elements, look in the offspring to see what element has been copied in its place from parent 1. In the example, b in parent 2 is at the position of D in the offspring, and a in parent 2 is at the position of E in the offspring. Place each of these elements in parent 2, say b, in the offspring at the position occupied in parent 2 by the element which is in place of b in the offspring (D). In the example, the position of D (d) in parent 2 is immediately after the second crossover point, so b is placed at that position in the offspring. In the offspring, at the position of a in parent 2, there is E. The position in the offspring corresponding with the position of E (e) in parent 2, is already filled by another element (C). In this case, the element coming from parent 2 (a) goes in the offspring at the position occupied in parent 2 by the element C (c), which in the example is position one. When finished with the elements of the crossover section, the rest of the offspring can be filled from parent 2 without changing positions of the elements. In the example, f and g are filled from parent 2 to the offspring.

PMX, like order crossover, is a hybrid crossover in that tends to preserve, only partially, all the three relations (adjacency, relative order and absolute position) among elements. Adjacency is not perfectly transmitted: in the example, in both parents a (A) and b (B) are adjacent, but in the offspring they are not adjacent. Common relative order of the elements of the parents is not preserved either: in the example, in both parents b (B) comes before d (D), but in the offspring b comes after D. Also, position-wise values are not perfectly transmitted: in the example, at position six the elements in the parents are F and d, but the element at that position in the offspring (b) is neither of them. Notice, however, that when both parents have the same element at the same position, also the offspring will have that element at that position.

Theorem 5.7.1. *PMX is geometric under swap distance.*

Proof. The recombination done by PMX can be rephrased as follows: return as offspring parent 2 after having sorted its elements in the crossover section to the order of the elements of parent 1 in the crossover section using the minimum number of swaps.

The offspring so generated is on the minimal sorting trajectory by swaps between parent 1 and parent 2 because it can be thought as being obtained by applying the selection sort algorithm, which produces intermediate permutations on a minimal sorting trajectory between initial an target permutations, to sort parent 2 towards parent 1, and stopped it when the crossover section of parent 2 is ordered as the crossover section of parent 1. □

5.7.3 Cycle crossover

Cycle crossover [117] is concerned with preserving as much information as possible about the absolute position in which elements occur. Figure 5.3 shows an example of this crossover. Cycle crossover has two phases. Firstly, it divides the elements into cycles. A cycle is a subset of elements that has the property that each element always occurs paired with another element of the same cycle when the two parents are aligned[3]. Secondly, the offspring are created by selecting randomly cycles from the parents. In the example, the elements a, h, i, j and l belong to cycle 1, the elements b, g and k belong to cycle 2, the element c belong to cycle 3, and the remaining elements (d, e and f) belong to cycle 4. The offspring was formed by passing to the offspring the elements of cycles 1 and 3 from parent 1, and the elements of cycle 2 and 4 from

[3]The procedure to find cycles for a generalization of cycle crossover for permutations with repetitions is illustrated in section 5.8.

parent 2.

```
Positions:    1 2 3 4 5 6 7 8 9 10 11 12
Parent 1:     A B C D E F G H I J  K  L
Parent 2:     h k c e f d b l a i  g  j
Cycle label:  1 2 3 4 4 4 2 1 1 1  2  1

Offspring:    A k C e f d b H I J  g  L
```

Figure 5.3: Example of cycle crossover.

Since all elements in the offspring occupy the same positions as in one of the two parents, *cycle crossover preserves perfectly the absolute positions of the parents.* Relative order is not transmitted perfectly (see elements h and k in the example). Also, adjacency relation is not preserved perfectly (in the example, in both parents a and k are not adjacent, but in the offspring they are adjacent).

Theorem 5.7.2. *(sub-geometric crossover) Let $M = (S, d)$ a metric space and $XO : S \times S \to S$ a geometric crossover on M. Let $M' = (S', d)$ a sub-metric space of M, such as $S' \subseteq S$ and let $XO' : S' \times S' \to S'$ the recombination operator obtained by restricting the domain and the codomain of XO from S to S'. Then XO' is geometric crossover on M'.*

Proof. $\forall x, y \in S' : [x, y]_M \supseteq [x, y]_{M'}$ because every segment in M' is the restriction on S' of the same segment in M. Since X is geometric crossover on M we have that $\forall x, y \in S : XO(x, y) \subseteq [x, y]_M$. So, $\forall x, y \in S' : XO'(x, y) \subseteq [x, y]_{M'}$. Hence, XO' is geometric crossover on M'. □

Theorem 5.7.3. *Cycle crossover is geometric under Hamming distance restricted to permutations.*

Proof. For a locus i the offspring equals parent 1 or parent 2. Hence, for theorem 5.7.2 cycle crossover is geometric crossover under Hamming distance because it can be seen as the restriction from the set of integer vectors to the subset of permutations of the traditional mask-based homologous crossover for integer vectors which is geometric under Hamming distance (see chapter 9). □

In section 5.8, we will show that cycle crossover is also geometric under swap distance.

5.7.4 Merge crossovers

Merge crossovers 1 and 2 [11] use a global precedence vector to recombine parent permutations (see figure 5.4). Given any two elements in the permutation, the global precedence vector indicates which element has higher priority for processing (elements which appear earlier in the vector have higher precedence).

```
Parent 1:    C F G B A H D I E J
Parent 2:    E B G J D I C A F H
Precedence: A B C D E F G H I J

Offspring:   C B G F A H D E I J (MX1)

Offspring:   C E B F G A H D I J (MX2)
```

Figure 5.4: Example of merge crossover 1 and 2.

Let us consider merge crossover 1. Parents are processed from left to right. At position 1, parent 1 has C and parent 2 has E. Since C has precedence on E, C is copied to the offspring at position 1. Since C has already been allocated a position in the offspring, the C which appears later in parent 2 is swapped with E at the position 1 of parent 2. The second position will be processed next in the same way. When all positions have been processed, both parents are transformed into copies of the same offspring. Merge crossover 2 differs from merge crossover 1 in that when an element is added to the offspring it is deleted from both parents instead of being swapped. The deletion treats permutations as lists, therefore, the deletion of an element does not leave an empty position in the permutations. After all elements have been processed, both parents are transformed into empty lists.

Merge crossover 1 does not preserve perfectly the absolute position relation between elements (see elements of the parents and of the offspring at position 4 in the example). It does not transmits the adjacency relation either (elements F and A are non-adjacent in both parents, but they are adjacent in the offspring). Although in this example the common relative order of the elements of the parents is perfectly transmitted to the offspring, this does not hold in general[4].

Merge crossover 2 does not preserve perfectly the absolute position relation between elements (see elements of the parents and of the offspring at position 4 in the example). It does not transmits the adjacency relation either (elements B and G are adjacent in both parents, but they are not adjacent in the offspring). However, *merge crossover 2 transmits perfectly the common relative order of the elements of the parents to the offspring.* We show this formally in theorem 5.7.5.

Theorem 5.7.4. *Merge crossover 1 is geometric crossover under swap distance.*

Proof. The offspring obtained by merge crossover 1 is on the minimal sorting trajectory by swaps between the two parent permutations because for each position the swap performed makes either parent 1 one swap closer to parent 2 or parent 2 one swap closer to parent 1. Information on which parent has to go forward the other is in the precedence list. When the two parents have become identical, it means that they have met on a shortest sorting trajectory. That exact permutation is the offspring of merge crossover 1. □

Theorem 5.7.5. *Merge crossover 2 transmits perfectly the common relative order of the elements of the parents to the offspring.*

Proof. We need to prove that for any two elements x and y, if x precedes y in both parents, then x precedes y also in the offspring. The opposite relation holds by symmetry. Since x precedes y in both parents, it is always processed before y in the construction of the offspring, in one parent or the other. When x is encountered, say in parent 1, x is copied to the offspring, if it has higher priority than the first element of parent 2. In this case x precedes y in the offspring. If x has lower priority, then the first element of the parent 2 is copied to the offspring. The first element of parent 2 cannot be y when x is the first element of parent 1 or before because, since also in parent 2 x precedes y, the condition in which both parents have x in position 1 must occur before than the condition in which parent 1 has x in position 1 and parent 2 has y in position 1. When both parents have x in position 1, x is copied to the offspring and deleted from both parents. So, also in this case x is copied in the offspring before y. □

Corollary 5.7.6. *Merge crossover 2 is geometric crossover under ROD (hamming distance between relative order matrices).*

Proof. For theorem 5.7.5 for any two elements x and y, we have the following cases: (i) if x precedes y in both parents, x precedes y in the offspring. In this case, the entry (x, y) of the ROM matrices of both parents and of the offspring is 1. (ii) If x follows y in both parents, x follows y in the offspring. In this case, the entry (x, y) of the ROM matrices of both parents and

[4]For a counterexample, consider parent permutations ($ACDEB$) and ($BACDE$), and precedence vector ($BCDEA$). In this case, the offspring permutation is ($BCDEA$). In both parents, A precedes C, but in the offspring C precedes A.

of the offspring is 0. (iii) If x precedes y in one parent, and x follows y in the other, then x can either precede or follow y in the offspring. In this case, the entry (x, y) of the ROM matrices of the parents are 1 and 0, and the entry (x, y) of the ROM matrix of the offspring is 1 or 0. Therefore, the ROM matrix of the offspring obtained with merge crossover 2 can be thought as the result of the ROX geometric crossover of the ROM matrices of its parents which is geometric under ROD. So, merge crossover 2 is also geometric under ROD. □

5.8 Cycle crossover for permutations with repetitions

In this section we present a generalization of cycle crossover. In chapter 11, we will apply this crossover to the graph partitioning problem. The present section is organized as follows. In Section 5.8.1 we introduce the notions of repetition class of a permutation with repetitions and class-preserving geometric crossover. In Section 5.8.2 we describe a generalization of cycle crossover for permutations with repetitions. This is a class-preserving geometric crossover under Hamming distance. In Section 5.8.3 we point out its properties. In Section 5.8.5 we show that cycle crossover for simple permutations is geometric under swap distance but the extension of cycle crossover to permutations with repetitions is not geometric under swap distance. This comes as a surprise because when the extended cycle crossover is applied to simple permutations it behaves exactly as the simple cycle crossover, hence it becomes geometric under swap distance. We elicit the origin of this counterintuitive result.

5.8.1 Geometric crossover for permutations with repetitions

Permutations with Repetitions: In a permutation every elements occurs exactly once, e.g., (21453). In a permutation with repetitions the same value may occur more than once, e.g., (214154232), where 1 occurs twice, 2 occurs 3 times, 3 occurs once, 4 twice, and 5 once. The numbers of occurrences of each element define the repetition class of the permutation. Two permutations with repetitions in which elements have the same numbers of repetitions belong to the same repetition class. All simple permutations (without repetitions) belong to the same repetition class.

Class-preserving Neighborhood Structure: Let us consider the set P of all the permutations with repetitions such that the element 1 is repeated n_1 times, element 2 is repeated n_2 times, and so on. So n_i is the size of group i. In P, any permutation has a fixed number of repetitions for each group. So, all permutations with repetitions in P belong to the same repetition class.

We can define a neighborhood structure on P as follows: the neighbor is generated by swapping the positions of any two different elements in the permutation. For example, swapping positions of the elements at positions 2 and 5 in 123231 gives 133221, so they are neighbors.

This operation transforms a permutation in P into another permutation in P: it does not change the sizes of the groups. In other words, the swap operation is a *class-preserving transformation.*

It is also easy to see that this neighborhood is symmetric (if p_1 is in the neighborhood of p_2, then p_2 is in the neighborhood of p_1) and connected (it exists a finite number of swaps that transform any permutation in P into any other permutation in P).

Notice that many other edit operations for simple permutations can be naturally extended to the case of permutations with repetitions and are indeed class-preserving transformations. For example, the block-reversal move that reverses a block of consecutive elements in the permutation does not change the class of permutation either. In this section we focus on the case of the swap move.

Class-preserving Swap Distance: Passing from the neighborhood structure (symmetric and connected) to the corresponding notion of distance is straightforward: the distance between two solutions is the length of a shortest path connecting them in the neighborhood structure. This means that the distance between two permutations with repetitions (of the same repetition class) is the minimum number of swap operations required to transform one into the other. This is an edit distance. Hence, it is a metric.

Class-preserving Geometric Crossover: By definition of geometric crossover, once we have a distance over a space we can define a crossover for that space. The geometric crossover has to pick offspring on shortest paths connecting two parent solutions. The extension of sorting crossovers for simple permutations to permutations with repetitions *seems* straightforward. For simple permutations, selection sort can be used as a base for geometric crossover under swap distance because it sorts simple permutations within the minimum number of swaps. However, when applied to permutations with repetitions, selection sort is not guaranteed to sort using the minimum number of swaps, although its deviation from this behaviour is marginal in practice (we explain why this happens in section 5.8.5). Therefore, partially sorted permutations with repetitions are not necessarily on the minimal sorting trajectory (shortest path) between parent permutations with repetitions, and selection sort cannot be used to implement a perfect geometric crossover under swap distance. However, in practice, a recombination based on selection sort is class-preserving and it produces a very good approximation of geometric crossover under swap distance. Hence, it can be thought to be a geometric crossover plus a little mutation.

An alternative way to build geometric crossovers for permutations with repetitions under swap distance is to generalize PMX and cycle crossover that are geometric crossovers under swap distance for simple permutations. However, since also these crossover operators are in fact sorting crossovers (in disguise), the above discussion about the non-geometricity of sorting crossover for permutations with repetitions applies to PMX and cycle crossover too. However, cycle crossover is geometric also under Hamming distance. This allows us to generalize it to the case of permutations with repetitions and preserve geometricity under Hamming distance.

5.8.2 Generalization of cycle crossover

The extension of the cycle crossover we propose produces offspring of the same repetition class of the parents. This crossover has two phases: (i) finding cycles and (ii) mixing cycles. We explain these below by means of an example.

Phase I (finding cycles)

Let us consider two parents of the same repetition class ($n_1 = 2$, $n_2 = 2$, and $n_3 = 2$) as shown in Figure 5.5a. In order to identify cycles, we proceed as follows:

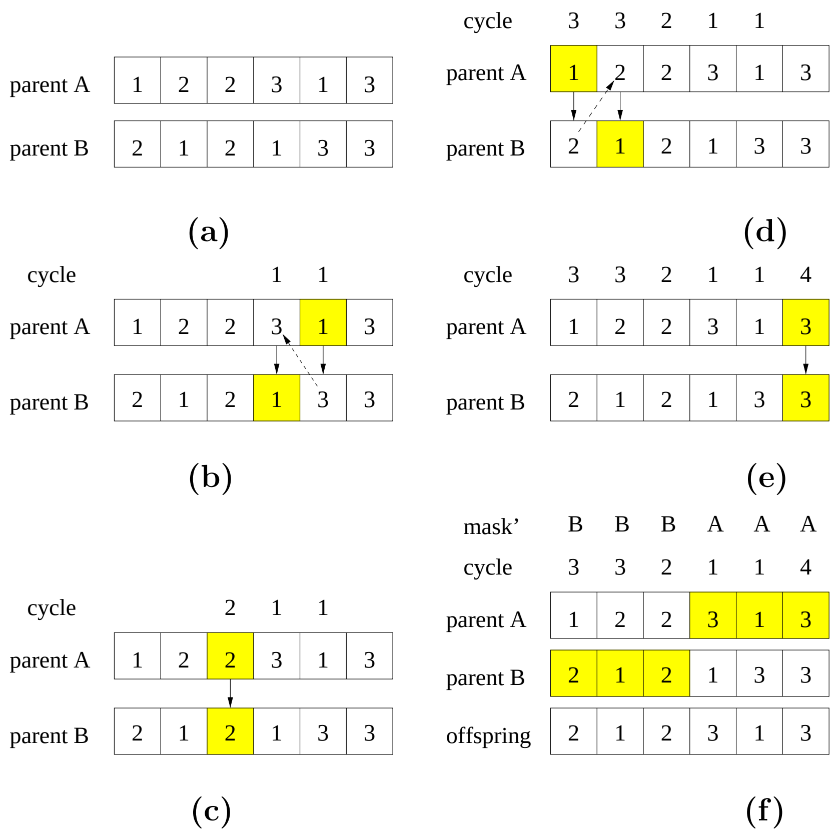


Figure 5.5: Cycle crossover step by step

- (1) Pick a random position in parent A, e.g., position 5. In this position in parent A, we have the element 1 corresponding (at the same position) to the element 3 in parent B. We denote this correspondence with $1 \rightarrow 3$. Mark position 5 (in both parents) as taken. The marking refers always to both parents.

- (2) Consider the corresponding element in parent B and pick at random any of its occurrence in parent A (among non-taken positions). In our example, pick any 3 in parent A, let us pick the one at position 4. The corresponding element in parent B is 1: $3 \rightarrow 1$. Mark position 4 as taken.

- (3) Repeat (1) & (2) continuing the process of path extension until we get an element of parent B identical to the first element we considered in parent A. When this happens, we have found a cycle. In our example, the last element we got in parent B is 1 that is the same as the first element we considered in parent A. So we found the cycle: $1 \rightarrow 3, 3 \rightarrow 1$.

This is shown in Figure 5.5b. Notice that the cycle involves the same number of repetitions in both parents. In our example, 1×1 and 3×1. Excluding the elements of cycle 1 from the two parents leaves the remaining elements with the same number of repetitions in the two parents. So, the “leftover” permutations are still of the same repetition class. In our example, the leftover repetition class is: $n_1 = 1$, $n_2 = 2$, and $n_3 = 1$.

- (4) Repeat loop (1)–(3) to find more cycles until all positions have been marked with a cycle tag.

Continuing our example: ($1'$) Pick one free position in parent A at random. Say position 3. We have a 2. ($2'$) Corresponding element in parent B: $2 \rightarrow 2$. ($3'$) We already found a cycle: $2 \rightarrow 2$ (Figure 5.5c).

We then start searching for a third cycle: ($1''$) Pick one free position in parent A at random. Say position 1. We have a 1. ($2''$) Corresponding element in parent B: $1 \rightarrow 2$. Pick a 2 in parent A: the only one available is the one at position 2. Its corresponding element in parent B is: $2 \rightarrow 1$. ($3''$) We have found another cycle: $1 \rightarrow 2, 2 \rightarrow 1$ (Figure 5.5d).

We run the process one further time: ($1'''$) The only free position in parent A is position 6. We have a 3. ($2'''$) The corresponding element in parent B is 3. ($3'''$) We have found a cycle: $3 \rightarrow 3$ (Figure 5.5e).

All the positions have been assigned to a cycle, so Phase I is over. Notice that the last iteration is always guaranteed to terminate with a cycle (and not with a simple sequence). The last position marked must be the end of the cycle.

Phase II (mixing cycles)

- (1) Create a crossover mask with one entry for each cycle by randomly flipping a coin as many times as the number of cycles detected in the previous phase. In our example, we have four cycles and, say, the crossover mask we generate is $mask$ = (ABBA). The entries in the mask indicate from which parent each cycle is inherited. In this example, the offspring will inherit cycles 1 and 4 from parent A and cycles 2 and 3 from parent B.

- (2) We convert this "cycle" mask into a standard recombination mask by relabeling all the entries c_i in cycle as follows: $c_i \rightarrow mask(c_i)$ obtaining a new mask *mask'*.

- (3) We perform standard mask-based crossover on the two parents using *mask'*, obtaining the offspring as shown in Figure 5.5f.

Notice that by construction every offspring has the same number of repetitions of the parents. This is because exchanging paired cycles among parents is a repetition-class preserving operation.

5.8.3 Properties of cycle crossover

The new crossover has the following properties:

1. It is repetition class preserving.

2. It is a proper generalization of cycle crossover: when applied to simple permutations it behaves exactly like cycle crossover.

3. It is geometric under Hamming distance because at each position the element in the offspring equals the element at that position in one of the parents.

4. It is defined over the induced sub-metric space obtained by restricting the original vector space endowed with Hamming distance to the space of permutations with repetition of the same repetition class. The latter space is much smaller than the former, hence quicker to search.

5. Applying this crossover to permutations with repetitions of different repetition class (with minor modifications, see section 5.8.4), one obtains offspring with intermediate repetition class with respect to the repetition classes of the parents.

5.8.4 Pseudo-code for finding cycles in cycle crossover

Algorithm 3 Pseudo-code for finding cycles in cycle crossover

```
Require: $parent_1[1..n]$ and $parent_2[1..n]$ {$n$: # of genes}
  $pcnt_1[1..K] \leftarrow 0$, $pcnt_2[1..K] \leftarrow 0$; {$K$: # of partitions}
  for each gene $i$ do
    Increase $pcnt_1[parent_1[i]]$ by one;
    Increase $pcnt_2[parent_2[i]]$ by one; {# of genes which each partition has}
  end for
  $no\_cycle \leftarrow 0$, $count \leftarrow 0$, and $visited[1..n] \leftarrow$ FALSE;
  repeat
    Increase $no\_cycle$ by one;
    Choose a random gene $k$ such that $visited[k] =$ FALSE;
    $visited[k] \leftarrow$ TRUE, $cycle[k] \leftarrow no\_cycle$, and increase $count$ by one;
    $start \leftarrow parent_1[k]$ and $corr \leftarrow parent_2[k]$;
    Decrease $pcnt_1[start]$ and $pcnt_2[corr]$ by one;
    if $pcnt_1[corr] \neq 0$ then
      repeat
        Choose a random gene $k$ such that $visited[k] =$ FALSE and $parent_1[k] = corr$;
        $visited[k] \leftarrow$ TRUE, $cycle[k] \leftarrow no\_cycle$, and increase $count$ by one;
        Decrease $pcnt_1[corr]$ and $pcnt_2[parent_2[k]]$ by one;
        $corr \leftarrow parent_2[k]$;
        if $pcnt_1[corr] = 0$ then
          break; {if it is impossible to find a cycle, we regard this path as a cycle}
        end if
      until $start = corr$
    end if
  until $count = n$
  return $cycle[1..n]$ and $no\_cycle$; {$cycle[k]$: cycle # that the $k^{th}$ gene belongs to, $no\_cycle$: # of cycles}
```

5.8.5 Non-geometricity under swap distance

Theorem 5.8.1. *Cycle crossover for permutations with repetitions is not geometric under swap distance.*

Proof. We prove it by giving a counter-example. Let us consider two parents $A = (122313)$ and $B = (213132)$. We can have (at least) two cycles decompositions after Phase I:

cycle	1	1	2	3	3	2
parent A	1	2	2	3	1	3
parent B	2	1	3	1	3	2

and

cycle	1	2	1	1	2	2
parent A	1	2	2	3	1	3
parent B	2	1	3	1	3	2

By combining cycles of the first decomposition, one obtains offspring that are always in the segment between parents under swap distance. However, by combining cycles of the second decomposition, one obtains an offspring that is not in the segment between parents under swap distance:

mask’	A	B	A	A	B	B
cycle	1	2	1	1	2	2
parent A	1	2	2	3	1	3
parent B	2	1	3	1	3	2
offspring	1	1	2	3	3	2

Note that $d_{\text{sw}}(A, B) = 3$. This can be seen from the first decomposition in cycles (minimal): 3 cycles of length 2 is equivalent to 3 swaps away.

We also have that $d_{\text{sw}}(A, \mathit{offspring}) = 2$: By construction, the minimal decomposition of the offspring and parent A in cycles is the cycle of length 3 taken from parent B. It means that they are 2 swaps away.

Finally we have $d_{\text{sw}}(\mathit{offspring}, B) = 2$: By construction, the minimal decomposition of the offspring and parent B in cycles is the cycle of length 3 taken from parent A. It means that they are 2 swaps away.

Since $d_{\text{sw}}(A, B) < d_{\text{sw}}(A, \mathit{offspring}) + d_{\text{sw}}(\mathit{offspring}, B)$, cycle crossover is non-geometric under swap distance. □

Theorem 5.8.2. *Cycle crossover for permutations with repetitions restricted to the decomposition with the most cycles is geometric under swap distance.*

Proof. Let A and B be two parents. For each decomposition in cycles there is (at least) a sequence of swaps associated with it that transforms parent A into parent B using exactly $n - mc$ swaps, where n is the length of the parents and mc is the number of cycles in the decomposition. This is because parent A can be transformed into parent B by one cycle at a time; for each cycle the transformation requires exactly a number of swaps equal to the cycle size minus one; summing up for all cycles we obtain $n - mc$ swaps.

Conversely, for each sequence of l swaps transforming parent A into parent B there is (at least) a cycle decomposition with $n - l$ cycles.

All offspring of cycle crossover for a given cycle decomposition of the parents are on the sequences of swaps associated with the decomposition in cycles.

All offspring of cycle crossover restricted to the decomposition of their parents with the most cycles are in the segment between parents under swap distance because they are on shortest paths of swaps linking one parent to the other. The paths associated with this decomposition are the shortest because (i) every path is associated with a cycle decomposition, (ii) the cycle decomposition associated with them is the one with the most cycles, and (iii) the more the cycles, the shorter the paths. □

Corollary 5.8.3. *Cycle crossover for simple permutations is geometric under swap distance.*

Proof. Simple permutations can be thought as permutations with one repetition for each element.

We know from abstract algebra that there is a unique decomposition in cycles for simple permutations. A unique decomposition is also the one with the most cycles. Hence, for theorem 5.8.2, in the special case of simple permutations, all the offspring are within the segment between parents. □

In summary, for simple permutations, cycle crossover is geometric under swap distance and Hamming distance. When considering the more general case of permutations with repetitions, cycle crossover is geometric under Hamming distance but it is not geometric under swap distance.

With similar arguments as for cycle crossover, it can be shown that PMX and all sorting crossovers under swap move are in general non-geometric crossovers under swap distance for permutations with repetitions.

5.9 Sorting crossovers for the n-queens problem

The eight queens' puzzle is the problem of placing eight chess queens on an 8 by 8 chessboard such that none of them is able to capture any other. The piece color is ignored, and any piece is assumed to be able to attack any other. That is to say, no two queens should share the same row, column, or diagonal. The generalized problem of placing n "non-dominating" queens on an n by n chessboard is a good example of a simple but non-trivial constraint satisfaction problem and, for this reason, is often used as an illustrative problem for non-traditional approaches, such as constraint programming, logic programming or genetic algorithms [140].

5.9.1 Solution representation, fitness function and genetic operators

Various encodings and solution representations for the n-queens problem have been suggested [34]; some of them reduce the search space enormously compared to more naive approaches thereby speeding up the search. Here, we use a permutation to represent a potential solution: the position of an element in the permutation identifies a row in the chessboard and the value of the element itself specifies the column of the location the queen in that row. This sets exactly n

queens on an n by n chessboard and reduces the search space, in that, column and row conflicts are eliminated. Only diagonal conflicts may arise. The fitness function is the number of conflicts among pairs of queens and has to be minimized.

The mutations and crossovers we consider are based on the generic mutation operator in algorithm 2 and on the generic sorting crossover operator in algorithm 1 presented in section 5.5.

We want to compare crossover operators derived from three classical sorting algorithms: selection sort, bubble sort and insertion sort. As already mentioned in section 5.5, these algorithms fit the definition of algorithm doing "minimal permutation sorting by x" paradigm. They are based on swap move, adjacent swap move and insertion move, respectively. As seen in section 5.4.1, the first two moves are good moves for absolute position and relative order problems, respectively. The third one does not associate to a clear-cut interpretation of the permutation and it mixes characters of the previous two.

For each sorting algorithm, we propose two types of crossover. The first type deterministically sorts the first parent permutation toward the second parent permutation (using always the same trajectory), collects all the permutations on the sorting trajectory and returns one of them at random. The second crossover type chooses a minimal sorting trajectory between two permutations at random (non-deterministic sorting) and then, analogously to the previous one, returns as offspring one of the permutations on that trajectory at random. We also consider three types of mutations, one for each move operator.

For reference, we compare the sorting crossover operators with the traditional partially matched crossover (PMX) recombination with swap mutation.

5.9.2 Experiments

In our experiments we used an evolutionary algorithm with the parameters shown in Table 5.1. The results of the experiments are shown in Figures 5.6 and 5.7.

Figure 5.6 compares the evolutionary algorithm using three different types of mutation, swap mutation (swp-only), adjacent swap mutation (adjswp-only) and insertion mutation (ins-only) without crossover. The abscissa is number of generations and the ordinate is the fitness of the

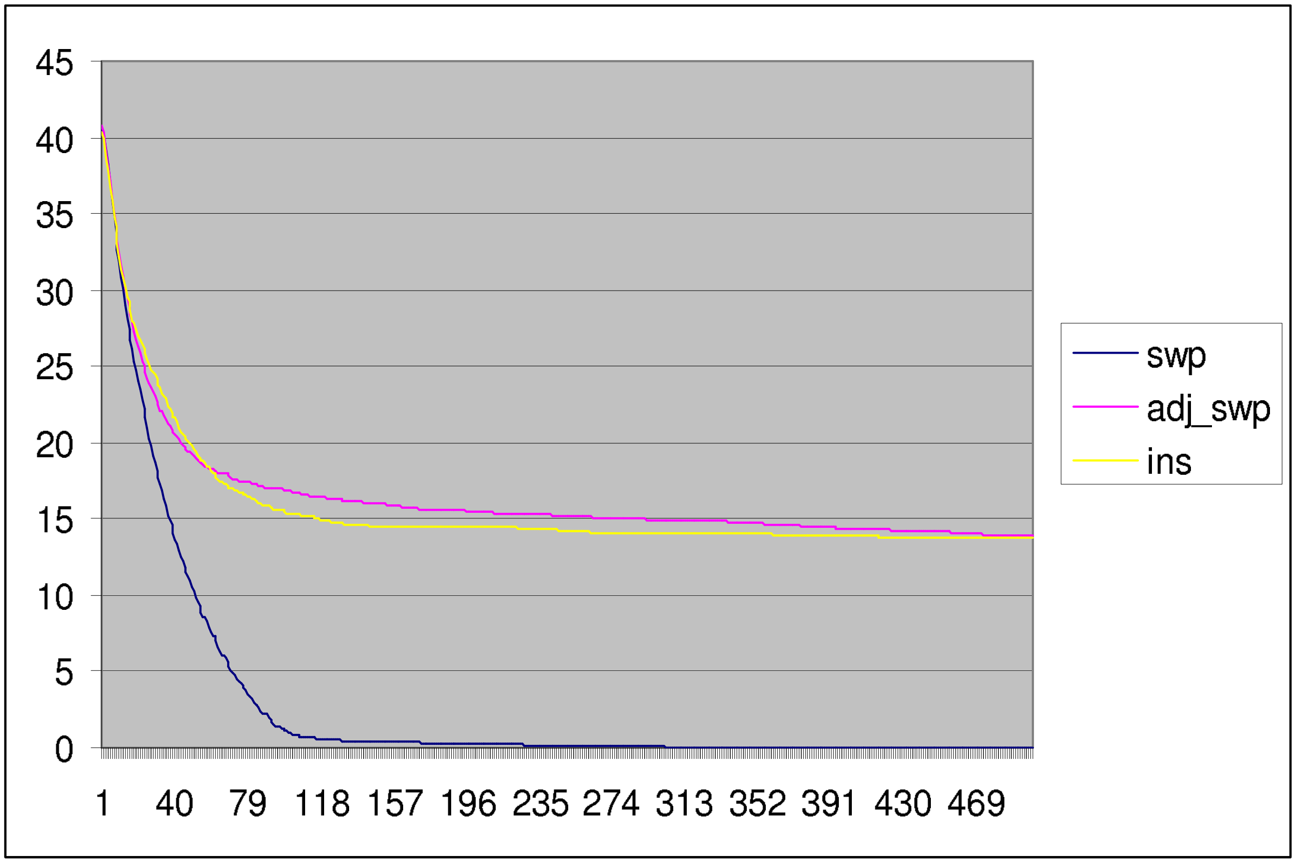


Figure 5.6: Comparison among mutations for the n-queens problem.

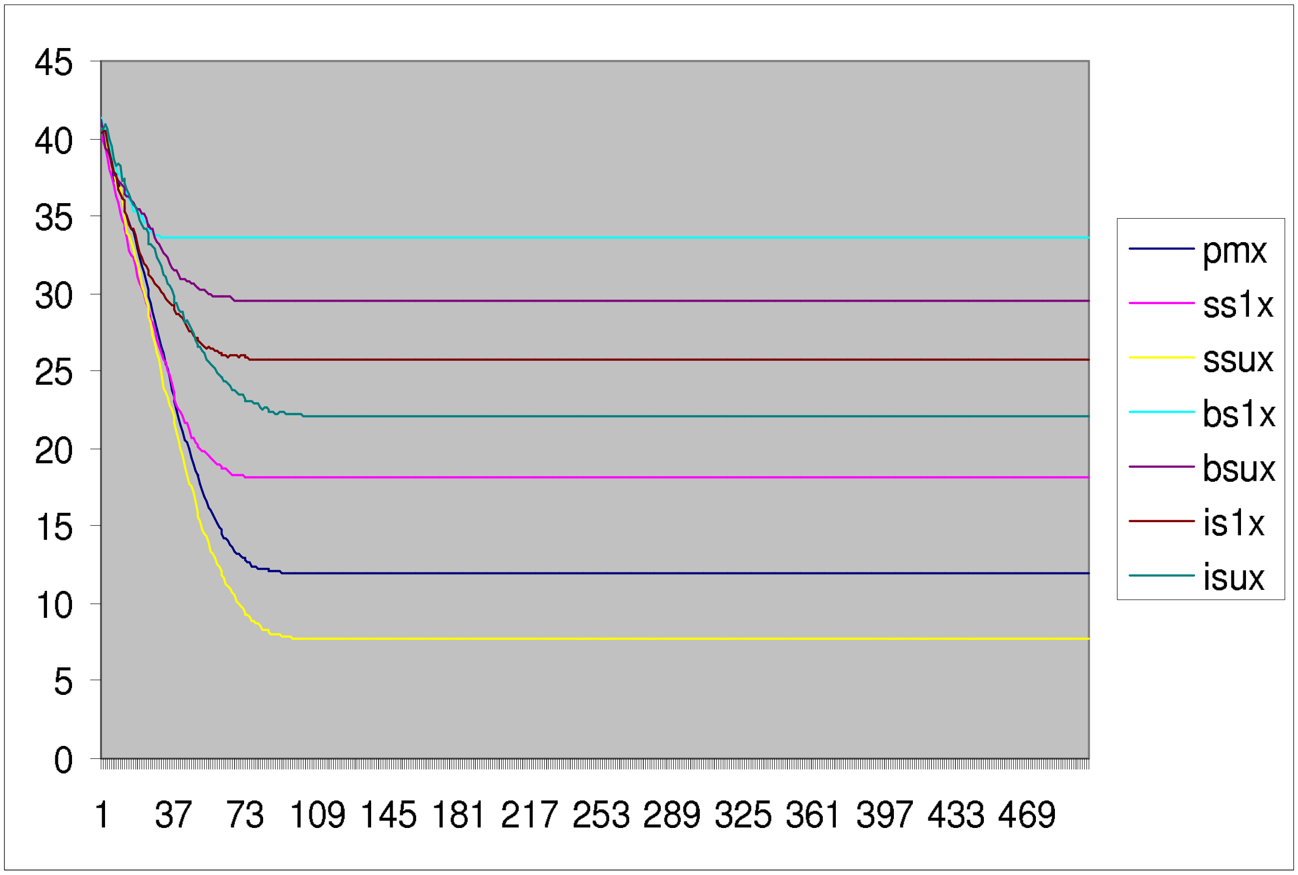


Figure 5.7: Comparison among crossovers for the n-queens problem.

Table 5.1: Parameters of the evolutionary algorithm for the n-queens problem

Problem size	100
Population size	5000
Mutation probability	0.1
Crossover probability	0.5
Generations	500
Selection	tournament size 2
Statistics	average 30 runs

best individual in the population averaged over 30 runs. Since the objective function has to be minimized, the swap mutation is clearly much superior to the other two mutations that perform similarly, perhaps with the insertion mutation performing slightly better.

Figure 5.7 compares the performance of the evolutionary algorithm using the following crossovers: partially matched crossover (PMX), deterministic and non-deterministic selection sort crossover based on the swap move (SS1X, SSUX), deterministic and non-deterministic bubble sort crossover based on the adjacent swap move (BS1X, BSUX), deterministic and non-deterministic insertion sort based on the insertion move (IS1X, ISUX). No mutation was used. The picture gives a clear ranking of the performance of crossovers: (1) SSUX, (2) PMX, (3) SS1X, (4) ISUX, (5) IS1X, (6) BSUX and (7) BS1X.

Each non-deterministic sorting crossover outperforms the corresponding deterministic one. This is not necessarily due to an inherent superiority of non-deterministic sorting crossovers over the deterministic counterparts. Non-deterministic crossovers are probably advantaged since they are more explorative and compensate for the lack of mutation.

The hypothesis that a move that produces a good mutation operator produces also a good corresponding crossover operator, put forward in chapter 3, is clearly corroborated by the experiment. Indeed, in our rank of crossovers, both SSUX and SS1X, based on the swap move, are superior to both ISUX and IS1X, based on the insertion move, which, in turn, are superior to both BSUX and BS1X, based on the adjacent swap move. This classification mirrors exactly the classification of mutations on performance. Interestingly, the non-deterministic sorting crossover based on the swap move (SSUX) outperforms the PMX operator.

5.10 Sorting crossover for TSP

Edge recombination [163] is an operator expressly designed for TSP. It considers a solution as a tour of cities. Therefore, rather than being defined for permutations it is defined over *circular permutations*. In its various improvements its stated objective is to greedily recombine parent tours in order to transmit as much as possible the adjacency relation, introducing in the offspring tours the minimum number of “foreign” edges not present in either parent [163].

As in the linear case, also for circular permutations it is possible to write an adjacency matrix. Again, the segment between the parent circular permutations (under Hamming distance for the adjacency relation matrix) contains all the feasible offspring circular permutations that perfectly respect the adjacency relation of their parents. The geometric crossover for circular permutations under this notion of distance is well-defined and actually achieves what *edge recombination can only aspire to*.

In the case of circular permutations, the block-reversal move is the notion of edit distance closest to the adjacency matrix distance. In a single application to a tour, this does the minimal change to the adjacency relation among elements in the permutation. This move is the well-known 2-change move, and it is the basis for successful local search algorithms for TSP [48]. Figure 5.8 shows the possible offspring (the segment) between two circular permutations (parents) under geometric crossover.

Analogously to the linear case, the circular permutations in the segment under reversal distance are those laying in a minimal sorting trajectory from a parent circular permutation to the other. Sorting circular permutations by reversals is NP-hard [145]. So, *the geometric crossover under this notion of distance cannot be implemented efficiently.*

Sorting circular permutations by reversals is tightly connected with the problem of sorting linear permutations by reversals. So, all the algorithms developed for the latter task can be used with minor modifications also for the former [145]. Sorting linear permutations by reversals is NP-hard, too [15]. However, a number of approximation algorithms (running in polynomial time) exist to solve this problem within a bounded error from the optimum [68]. This allows

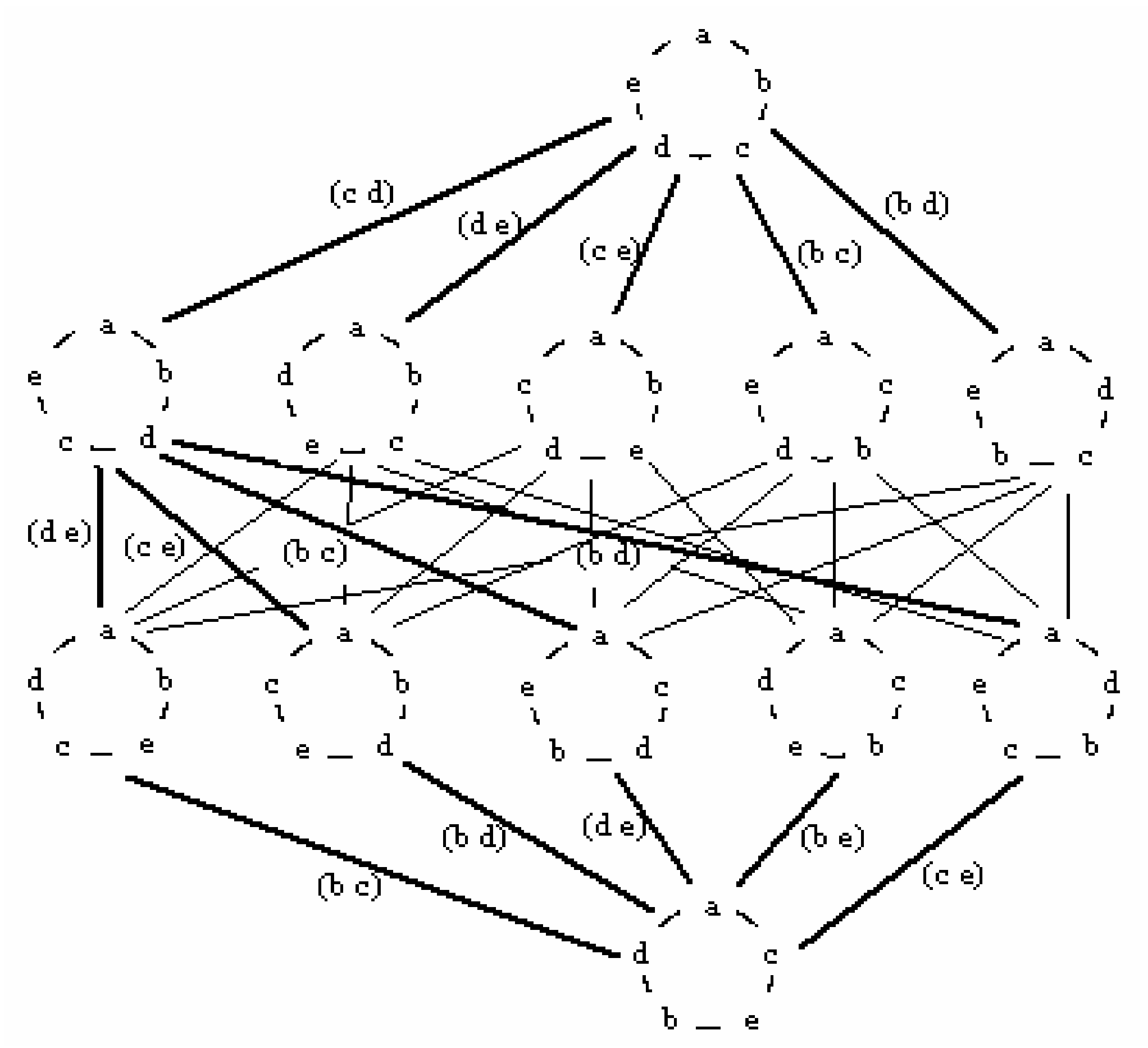


Figure 5.8: Example of geometric crossover between two circular permutations.

implementing efficiently approximate geometric crossovers.

5.10.1 Experiments

We have implemented an efficient approximated geometric crossover for circular permutations under reversal distance based on the algorithm for sorting permutations by reversals by Kececioglu and Sankoff[5] [68]. This algorithm achieves a worst-case approximation of factor 2, which

[5]The author of the thesis wants to express his gratitude to John Kececioglu and David Sankoff for making available the source code of their algorithm.

means that, in the worst case, it sorts permutations using twice the minimum number of reversals needed, however, on average the approximation factor is close to 1. It has (worst-case) time complexity $O(n^2)$, where n is the number of elements of the permutations to sort. The approximated geometric crossover uses the algorithm for sorting permutations by reversals to generate all permutations on a sorting trajectory by reversals between the two parent permutations, and it returns one of those permutations drawn at random as offspring.

We have tested our sorting crossover on the following frequently used problem instances of the well-known library of TSP problems TSPLIB[6] : eil51, gr96, eil101, lin105, d198, kroA200, lin318 and pcb442. The number in the name is the size of the instance (number of cities).

We compared three different recombination operators under the same setting: PMX, Edge Recombination (EX), which is known to be a very good operator for TSP, and sorting by reversal geometric crossover (SBRX). Since we wanted to compare the performance of the recombination operators alone, no mutation was used. We used a generational genetic algorithm with probability of crossover 1 and linear ranking. The population size was set to 20 times the problem instance size. As stop criteria we used population convergence. The ratio between population size and problem instance was chosen to allow the best recombination to reach a near optimal solution before population convergence for all instances. No other parameter tuning was performed.

We run 30 independent runs for each problem. In table 5.3 is reported, for each operator, the average and the standard deviation of the best fitness found, and the average and standard deviation of the number of fitness evaluation until convergence. In terms of solution quality, PMX is always the worst. ERX performs slightly better than SBRX for small instances (eil51 and gr96), but SBRX performs better on all the other instances. In particular, SBRX performs better and better against ERX as the instance size grows. In terms of number of iterations to convergence, the operators with better performance takes more time to converge. However, the extra time is only a fraction of the overall running time.

[6]TSPLIB can be downloaded from http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

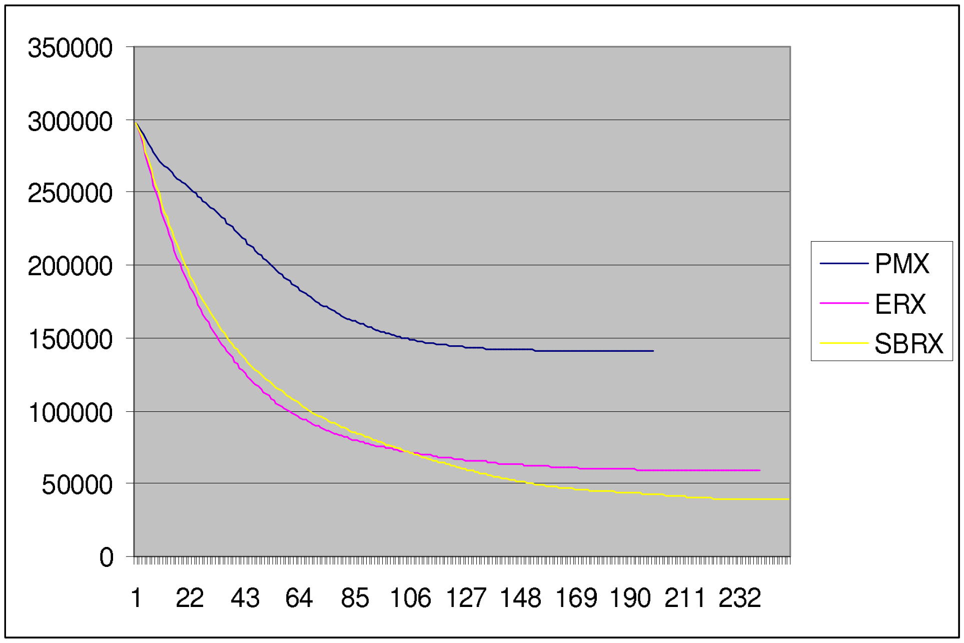


Figure 5.9: A typical run (instance KroA200).

In figure 5.9 the graph of the fitness of the best in the population (ordinate) over time (abscissa) of a run of the problem KroA200 is shown. This is a typical run: PMX produces the worst result in terms of fitness and stops earlier than the other two recombination operators; ERX has better fitness than SBRX until halfway, then SBRX continues to improve and passes ERX, and ERX converges earlier than SBRX.

5.11 Summary

Permutation interpretations and geometric aspects are intimately connected. In Table 5.2, we propose a summary of important attributes of geometric crossovers for permutations we considered in this chapter.

Permutations differ strongly from binary strings and real vectors, from which our geometric framework has originally arisen. For example, the geometric notion of orthogonality, applicable to binary strings and real vectors, is not present in permutations. One might wonder, then, whether there would be any hope of applying our geometric framework to permutations. In this chapter, we have shown that the geometric framework can be applied to permutations, so

Table 5.2: Attributes of crossovers for permutations (summary)

Crossover Example	**Permutation Interpretation**	**Edit Move**	**Sorting Algorithm**	**Computational Complexity**	**Problem Example**
Cycle crossover	Position	swap	Selection Sort	P	n-queens
Edge recombination	Adjacency	block-reversal	Sorting by Reversals	NP-hard	TSP
Merge crossover	Relative order	adjacent swap	Bubble Sort	P	JSSP
PMX	Hybrid	swap	-	P	general?

paving the way to a complete geometric unification of EAs. In addition we have shown that our geometric interpretation sheds light on this representation, revealing a hidden and intimate connection between geometry of permutations, crossover and sorting algorithms.

Geometric crossover and mutation are well-defined once one has a notion of distance over the solution set. The permutation representation allows for three notions of non-edit distances connected with the permutation interpretations. Three geometric crossovers based on the permutation interpretations are therefore well-defined. However, such geometric crossovers are hard or even impossible to implement efficiently, in that they are based on non-edit distances.

Most of the pre-existing crossover operators for permutations are designed around interpretations. We have shown that they fit, some exactly and some approximately, the geometric crossover definitions connected with permutations interpretations.

The permutation representation also allows for a number of edit distances connected with various notions of mutation. Each notion of edit distance induces a notion of geometric crossover. Because of the distance duality, under a given edit distance a point on a segment between two permutations is on a minimal sorting trajectory connecting the two permutations. This allows implementing such crossovers using *sorting algorithms*. Some edit distances give rise to crossovers that can be implemented exactly and efficiently. Other edit distances give rise to crossovers can be implemented efficiently (in polynomial time) only in an approximated way.

The three geometric crossovers induced by permutation interpretations are tightly connected with three geometric crossovers based on edit distances. The connection relies on the principle of “minimal change”.

We have shown a number of important recombination operators for permutations used in practice are geometric crossovers under different edit distances: PMX is geometric under swap distance, cycle crossover is geometric under both hamming distance restricted to permutations and swap distance, merge crossovers 1 and 2 are geometric crossovers under swap distance and relative order distance, respectively. Also, edge recombination aims to be a geometric crossover under the distance associated with the adjacency relation among elements of the permutation, but it succeeds at this only partially.

We have studied the subtle issues about the geometricity of crossovers arising when geometric crossovers for simple permutations are extended to the more general case of permutations with repetitions. We have proposed a generalization of cycle crossover to permutations with repetitions which is geometric under hamming distance but is not geometric under swap distance.

An experimental comparison on the n-queens problem of a number of geometric crossovers based on three classical sorting algorithms have been presented. The experiments corroborate the “good mutation, good crossover” rule-of-thumb proposed in chapter 3.

We argued that the fitness landscape for the TSP problem based on reversals distance for circular permutations is smooth, hence the associated geometric crossover should perform well. Experimental results agree with this prediction: the proposed geometric crossover for TSP beats edge recombination, which is one of the best crossover for this problem.

Table 5.3: Results of the comparison of recombination operators for the TSP

Problem(Opt)	PMX				ERX				SBRX			
	Fitness		Iterations		Fitness		Iterations		Fitness		Iterations	
	avg	std	avg	std	avg	std	avg	std	avg	std	avg	std
eil51(426)	631.4	38.6	81369.7	10036.5	426.4	1.2	118221.3	11596.1	431.8	4.6	84594.2	5340.6
gr96(51229)	149118	12110.8	210688	27685.9	58608.7	5356.5	427072	100460.5	61446.7	1633.2	258560	12116.1
eil101(629)	1413.4	86.2	213042.7	19913.4	738.1	63.3	369458	127840.5	686.4	27	272498	18398.3
lin105(14379)	40969.6	3257.5	247520	23564.5	16515.1	1675.4	481460	148344.8	15804.9	368.5	312900	15602.3
d198(15780)	54746.1	4052.8	750684	63448	29673.5	1430	829224	187930	18609.8	542.1	908292	129387.5
kroA200(29368)	141416.3	11771.9	634666.7	85717.9	58603.4	1760.4	763200	95640.4	38282.9	836.7	873200	59653.7
lin318	271538.3	12078.8	1284296	100264	113785.3	6947.5	1341960	324625.9	76735.3	5069.9	1833376	104026.1
pcb442	354119.5	14325.1	2110403	155927.4	166380.4	11564.3	2079463	435060.2	106840.9	2775.2	3023575	193856

Chapter 6

Sets

This chapter extends the geometric framework for interpreting crossover and mutation presented in chapter 3 to the case of sets and related representations. We show that a geometric duality exists between the set representation and the vector representation. This duality reveals the equivalence of geometric crossovers for these representations.

6.1 Introduction

Sets, multisets and partitions are natural representations for many important combinatorial optimization problems such as grouping problems, graph coloring and so on. The set representation for evolutionary algorithms was theoretically studied by Radcliffe [127] within his forma analysis framework.

In this chapter we use the geometric framework to study and design crossover operators for the set representation and related representations such as multi-sets and partitions, in their fixed-size and variable-size variants. We also show an illuminating isometric duality between the spaces associated to the set representation and the vector representation that enables us to prove the equivalence of geometric crossovers for these representations.

The chapter is organised as follows. In section 6.2, we extend the geometric framework to sets, multisets and partitions of variable-size. In section 6.3, we consider fixed-size sets, multisets and partitions of variable-size. In section 6.4, we illustrate the duality between sets and vectors.

6.2 Geometric crossover for variable-size sets, multi-sets and partitions

We consider problems where solutions are naturally represented as sets of objects taken from a reference set (universal set). We also consider the simple extension to multi-sets, i.e., sets that are allowed to contain repetitions of the same object. A set can be seen also as a bipartition of the universal set (objects in the set and remaining objects in the universal set). A natural extension of the notion of set in this sense is to consider generic multi-partitions of the universal set. We will study this case too.

There is a further aspect of the set representation that has a major impact on the associated geometric crossovers: the search being restricted to fixed-size sets versus the variable-size case. In this section, we study sets, multi-sets and partitions for the easier variable-size case. In section 6.3, we consider the fixed-size case.

6.2.1 Distances and crossovers for sets

Let U be the universal set and $A, B \subseteq U$. The *symmetric distance* between sets is $d(A, B) = |A \Delta B|$, where $A \Delta B = A \cup B \setminus A \cap B$ is the symmetric difference between sets. The symmetric distance is a metric [30]. When $A = B$, $d(A, B) = 0$; when $A \cap B = \emptyset$, then A and B are at maximum distance and $d(A, B) = |A| + |B|$. The *ins/del edit distance* between A and B is the minimum number of elements that need to be deleted or inserted for A to be transformed into B (and *vice versa*).

Theorem 6.2.1. *The symmetric distance is the same as the ins/del edit distance.*

Proof. The edit distance corresponds to the symmetric distance because the minimum number of elements that need to be deleted from A are $|A \setminus B|$ and the number of those that need to be added is $|B \setminus A|$. It is easy to see that $d(A, B) = |A \Delta B| = |A \setminus B| + |B \setminus A|$. □

Corollary: Since any edit distance is a metric [22], also the symmetric distance is a metric.

Theorem 6.2.2. *Given two parent sets, A and B, any recombination operator OP that returns offspring O such as $A \cap B \subseteq O \subseteq A \cup B$ is geometric crossover under symmetric distance.*

Proof. Proving geometricity under symmetric distance is equivalent to proving geometricity under ins/del edit distance. Any intermediate set C on the minimal ins/del move path to

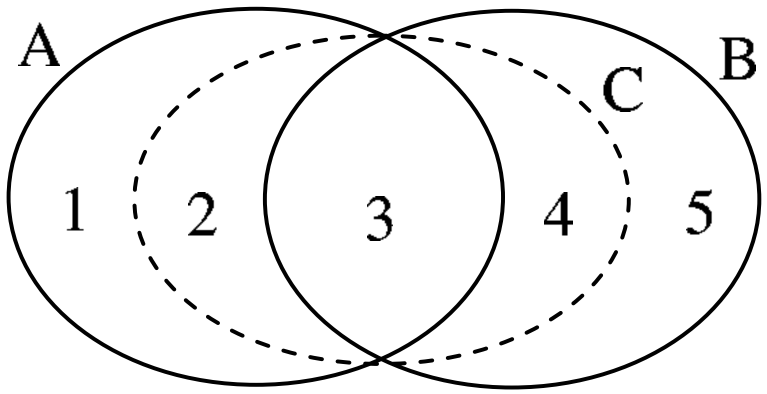


Figure 6.1: Venn diagram linking offspring set and parent sets.

transform A into B is between A and B (in the segment $[A, B]$) under ins/del edit distance (see Fig. 6.1). Every O such that $A \cap B \subseteq O \subseteq A \cup B$ belongs to such a path because: A can be transformed into B by inserting in A the elements $O \setminus A$, removing from A the elements $A \setminus O$ and then by inserting in B the elements $B \setminus O$, and removing from B the elements $O \setminus B$. So $d(A, O) = |O \setminus A| + |A \setminus O|$ and $d(B, O) = |O \setminus B| + |B \setminus O|$ $\square$

Example

Let $U = \{a, b, c, d\}$ be the universal set and $A = \{a, b\}$ and $B = \{b, c\}$ two parent sets (note that $A, B \subseteq U$). The symmetric distance between A and B is $d_\Delta(A, B) = |A \setminus B| + |B \setminus A| = 1+1 = 2$. Let GX_Δ be a geometric crossover under symmetric distance. Any offspring of A and B, $O = GX_\Delta(A, B)$, respects the condition $A \cap B \subseteq O \subseteq A \cup B$. So, in our example we have: $\{b\} \subseteq O \subseteq \{a, b, c\}$. The sets O that satisfy this condition are: $\{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}$. It is easy to verify that every O is in the segment between A and B under d_Δ. For example, if we consider $O = \{a, b, c\}$, we have $d_\Delta(A, O) + d_\Delta(O, B) = (0 + 1) + (1 + 0) = 2 = d_\Delta(A, B)$.

6.2.2 Distances and crossover for multi-sets

A multi-set (sometimes also called a *bag*) differs from a set in that each member has a *multiplicity*. This is a natural number indicating how many times the member occurs in the multi-set. A multi-set can be formally defined as a pair (A, m) where A is some set and $m : A \to \mathbb{N}$ is a function from A to the set of natural numbers $\mathbb{N}$. The set A is called the *underlying set of elements*. The *size* of the multi-set (A, m) is the sum of all multiplicities for each element of A: $|(A,m)| = \sum_{a\in A} m(a)$. A *submultiset* (B,n) of a multiset (A,m) is a subset $B \subseteq A$ and a function $n : B \to \mathbb{N}$ such that $\forall b \in B : n(b) \leq m(b)$. The usual operations of union and intersection for sets can easily be generalized to multisets. Suppose (A,m) and (B,n) are multisets. The *union* can be defined as $(A \cup B, f)$ where $f(x) = \max\{m(x), n(x)\}$. The *intersection* can be defined as $(A \cap B, f)$ where $f(x) = \min\{m(x), n(x)\}$.

Hence we can define the *symmetric difference* between multisets as $(A\Delta B, f)$ where $f(x) = \max\{m(x), n(x)\} - \min\{m(x), n(x)\} = |m(x) - n(x)|$. The *symmetric distance* for multisets becomes $d((A,m),(B,n)) = |(A,m)\Delta(B,n)| = \sum_{x\in A\Delta B} |m(x) - n(x)|$. The symmetric distance between multisets can be seen as a simple generalization of the *ins/del edit distance* for sets in which the edit move becomes the insertion or deletion of a *single occurrence* of an element.

The geometricity theorem for sets under symmetric distance (theorem 6.2.2) can be extended to the case of multisets.

Theorem 6.2.3. *Given two parent multisets (A,m) and (B,n) any recombination operator OP that returns offspring (O,f) such as $(A,m) \cap (B,n) \subseteq (O,f) \subseteq (A,m) \cup (B,n)$ is geometric crossover under symmetric distance.*

The proof of theorem 6.2.3 is a simple generalization of the proof of theorem 6.2.2.

Example

Let $U = \{a,b,c,d\}$ be the universal set, $A = \{a,b\}$ and $B = \{b,c\}$ be two sets of U. Let us consider the parent multiset $M_A = \{a,a,b\} = (A,m)$ where $m(a) = 2$ and $m(b) = 1$ and the parent multiset $M_B = \{b,b,c,c\} = (B,n)$ where $n(b) = 2$ and $n(c) = 2$. Their sizes are $|M_A| = 3$ and $|M_B| = 4$. Their union is $M_A \cup M_B = (A,m) \cup (B,n) = (A \cup B, f)$ where $f = max(m,n)$. In our example we have $M_A \cup M_B = \{a,a,b,b,c,c\}$. Their intersection is $M_A \cap M_B = (A,m) \cap (B,n) = (A \cap B, f)$ where $f = min(m,n)$. In our example we have $M_A \cap M_B = \{b\}$. Their symmetric difference is $M_A \Delta M_B = (A\Delta B, f)$ where $f = max(m,n) - min(m,n)$. In our example we have $M_A \Delta M_B = \{a,a,b,c,c\}$. The symmetric distance between M_A and M_B is, therefore, $d_\Delta(M_A, M_B) = |M_A \Delta M_B| = 5$. Let GX_Δ be a geometric crossover under symmetric distance for multisets. Any offspring M_O of M_A and M_B, $M_O = GX_\Delta(M_A, M_B)$, respects the condition $M_A \cap M_B \subseteq M_O \subseteq M_A \cup M_B$. So, in our example we have: $\{b\} \subseteq M_O \subseteq \{a, a, b, b, c, c\}$. Thus, any multiset $M_O = (O, f)$ such as $0 \leq f(a) \leq 2, 1 \leq f(b) \leq 2, 0 \leq f(c) \leq 2$ is a possible offspring of M_A and M_B.

6.2.3 Distances and crossover for partitions

A partition of a set X is a division of X into non-overlapping subsets that cover all of X. When the set X is partitioned into n subsets we say that they form an n-partition of X. A n-partition generalizes the notion of set (a set A can be seen as partitioning the universal set U in two subsets A and $\overline{A}$ (bipartition)).

In this chapter we restrict ourselves to partitioning problems with labeled partitions and a fixed number of partitions. A labelled (or ordered) partition is a partition in which each partitioning subset is identified by a label (so, a labelled partition can be seen as a *vector* of sets). In contrast, in an unlabelled partition each partitioning subset is identified by its members (so, an unlabelled partition can be seen as a *set* of sets). In this section, we consider the case where the same partition may have different size in different solutions. In section 6.3 we will consider the case in which all solutions are required to have the same size for the same partition.

The symmetric distance between two n-partitions $\mathcal{A} = \{A_1, \ldots, A_n\}$ and $\mathcal{B} = \{B_1, \ldots, B_n\}$ of a set X is a simple generalization of the symmetric distance for sets: $d(\mathcal{A}, \mathcal{B}) = \sum_i |A_i \Delta B_i|$.

The edit distance between two n-partitions is a natural generalization of the ins/del edit distance for sets. We define the edit distance between two n-partitions as the minimum number of edit moves necessary to transform one partition into the other. The edit move considered is moving one element from one subset to another. This edit move transforms a partition of X into another partition of X for which the conditions of full coverage of X and non-overlap of subsets are respected. The reason why this edit distance is a generalization of the ins/del edit distance for sets is that, when one considers a set A as a bipartition of the universal set U into A and $\overline{A}$, inserting or deleting one element from A implies respectively deleting or inserting the same element in $\overline{A}$. So, this is equivalent of moving one element from A to $\overline{A}$. Unlike the case of sets, however, the symmetric distance between partitions does not equal their ins/del edit distance (although these distances are related).

Example

Let $X = \{a, b, c, d\}$ be the universal set (the set to be partitioned), and be $\mathcal{A} = (\{a, b\}, \{c, d\})$ and $\mathcal{B} = (\{b, c, d\}, \{a\})$ two ordered (or labeled) bipartitions of X. Since we consider ordered partitions, $(\{a, b\}, \{c, d\}) \neq (\{c, d\}, \{a, b\})$. The edit distance between $\mathcal{A}$ and $\mathcal{B}$ is the minimum number of elements that need to be transferred from one subset to another to transform $\mathcal{A}$ into $\mathcal{B}$ (or *vice versa*). In our case, in order to transform $\mathcal{A}$ into $\mathcal{B}$, we need to transfer c and d from the second subset to the first subset and transfer a from the first subset to the second for a total of 3 edit moves. So, the edit distance $ed(\mathcal{A}, \mathcal{B}) = 3$. The geometric crossover under edit distance requires the offspring partition $\mathcal{O} = (O_1, \ldots, O_n)$ to satisfy the condition: $\forall i : A_i \cap B_i \subseteq O_i \subseteq A_i \cup B_i$. Notice that the sets O_i need to form a partition of X. Hence, they need to be chosen so as to be non-overlapping and covering X completely (so, their choices cannot be made independently). In our example we have $\{b\} \subseteq O_1 \subseteq \{a, b, c, d\}$ and $\emptyset \subseteq O_2 \subseteq \{a, c, d\}$. Considered independently, O_1 can be any subset of X including $\{b\}$ (8 possible subsets) and O_2 can be any subset of $\{a, c, d\}$ (8 possible subsets). This would suggest there are $8 \times 8 = 64$ combinations. However, since O_1 and O_2 need to form a partition of X, we have only 8 choices (and not 64). These are of the form: $\mathcal{O} = (O_1, \overline{O}_1)$ where $\{b\} \subseteq O_1 \subseteq \{a, b, c, d\}$.

6.3 Geometric crossover for fixed-size sets, multi-sets and partitions

Let U be the universal set and $\mathcal{X}_n$ the set of all subsets of U of size n, $\mathcal{X}_n = \{A : A \subseteq U, |A| = n\}$, and let $A, B \in \mathcal{X}_n$. Note $n \leq |U|$ or $\mathcal{X}_n = \emptyset$.

The *edit distance between sets under element substitution move* between A and B is the minimum number of elements of A that need to be substituted with elements in $U \setminus A$ to transform A into B (or *vice versa*). Since this distance is an edit distance, it is a metric.

For any two sets of the same size, their ins/del edit distance is twice their substitution edit distance. This is because every substitution is equivalent to one deletion and one insertion operation and there are no shorter chains of operations to transform one set into another of the same size using deletions and insertions. The substitution edit distance is well-defined only for sets of the same size because sets of different sizes cannot be transformed into each other by substitutions only.

Theorem 6.3.1. *Given two parent sets $A, B \in \mathcal{X}_n$ any recombination operator OP that returns offspring $O \in \mathcal{X}_n$ such that $A \cap B \subseteq O \subseteq A \cup B$ is a geometric crossover under substitution edit distance. So, this geometric crossover is a geometric crossover under ins/del edit distance restricted to sets of size n.*

Proof. if we restrict the image set of a geometric crossover from X to $S \subseteq X$ we obtain a new geometric crossover that for any two parents $a, b \in S$ returns offspring in $[a, b] \cap S$. So, restricting the geometric crossover associated to ins/del edit distance from the set 2^U to the set $\mathcal{X}_n \subseteq 2^U$, we obtain a new geometric crossover based on the ins/del distance that returns offspring of the same size of the parents. □

This restricted crossover is also a geometric crossover under substitution edit distance. This is because given $A, B \in \mathcal{X}_n$, $O \in [A, B]$ under ins/del edit distance iff $O \in [A, B]$ under substitution edit distance because ins/del edit distance is twice of substitution edit distance and proportional metrics have the same metric intervals.

Example

Let $U = \{a, b, c, d\}$ be the universal set and $A = \{a, b\}$ and $B = \{b, c\}$ two parent sets in U. The substitution edit distance between A and B is $d_{sub}(A, B) = 1$ since A can be transformed into B by substituting the element b with c. Their ins/del edit distance, which equals their symmetric distance, is $d_{\Delta}(A, B) = 2 \cdot d_{sub}(A, B) = 2$.

All the offspring of A and B produced by the geometric crossover under ins/del edit distance GX_{Δ}, $O = GX_{\Delta}(A, B)$, respect the condition $A \cap B \subseteq O \subseteq A \cup B$. These are the sets: $\{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}$. The offspring obtained by geometric crossover under substitution edit distance are those that have the same size as the parents: $\{a, b\}, \{b, c\}$. They are in the segment between parents A and B under substitution edit distance (in this case the only valid offspring are the parents themselves).

6.4 Geometric duality of sets and vectors

In this section we show that the same metric spaces considered in sections 6.2 and 6.3, arising from the set and related representations, also arise from the vector representation and permutations with repetitions. In other words, set spaces and vector spaces are isometric. This enables us to show that the geometric crossovers considered in section 6.2 and 6.3 for sets, multi-sets and partitions all have equivalent dual geometric crossovers based on vectors in the variable-size case and on permutations with repetitions in the fixed-size case (see Table 6.1).

6.4.1 Equivalence of crossovers for sets and binary strings

Definition 6.4.1. (Isometry) Let X and Y be metric spaces with metrics d_X and d_Y. A map $f : X \to Y$ is called *distance preserving* if for any $x, y \in X$ one has $d_Y(f(x), f(y)) = d_X(x, y)$. A distance preserving map is automatically injective. A *global isometry* is a bijective distance preserving map. Two metric spaces X and Y are called *isometric* if there is a global isometry from X to Y.

Geometric crossovers based on isometric spaces are equivalent in the following sense. Let us consider two isometric metric spaces X and Y with distances d_X and d_Y, and be $f : X \to Y$ a global isometry. Then we have that $\forall x, y \in X : f([x, y]_{d_X}) = [f(x), f(y)]_{d_Y}$. So, any segment s_x in X has a (one-to-one) corresponding segment s_y in Y, and every point in s_x has a (one-to-one) corresponding point in s_y. So, any geometric crossover defined on X has a (one-to-one) corresponding geometric crossover on Y via the isometry f.

Variable size sets

In the following we show that the space of variable size sets endowed with the symmetric distance is isometric via the indicator function to the space of binary strings endowed with Hamming distance. Hence, the symmetric crossover for sets is equivalent to homologous crossover for binary strings. In the following, we prove the duality and illustrate it with an example.

Definition 6.4.2. (Indicator function) The indicator function of a subset A of a set U is a function $I_A : U \to \{0, 1\}$ defined as $I_A(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \notin A. \end{cases}$

Let U be the universal set, d_Δ the symmetric distance between sets and $M = (2^U, d_\Delta)$ the metric space based on the set of all subsets of U together with the symmetric distance.

Let I_A be the indicator function of $A \subseteq U$ where $U = \{x_1, \cdots, x_n\}$ and V_A be the vector $(I_A(x_1), \cdots, I_A(x_n))$. The map $V : A \to V_A$ mapping a set A and its indicator values vector V_A is bijective.

Let $M' = (\{0,1\}^n, d_H)$ be the metric space of the binary strings of size n endowed with the Hamming distance d_H.

Theorem 6.4.1. *The metric spaces $M = (2^U, d_\Delta)$ and $M' = (\{0,1\}^n, d_H)$ are isometric.*

Proof. It is sufficient to prove that the the map $V : A \to V_A$ is a distance preserving map. It is immediate to see that for any $A, B \subseteq U$ we have $d_\Delta(A, B) = d_H(V_A, V_B)$. To transform A into B with the minimum number of ins/del operations, the elements that need to be inserted into A are those x_i for which $V_A(i) = 0$ and $V_B(i) = 1$ and the elements that need to be deleted from A are those x_j for which $V_A(j) = 1$ and $V_B(j) = 0$. These are the only positions in which V_A and V_B differ. Since V is bijective, the opposite implication, $V_A, V_B \in \{0,1\}^n : d_\Delta(V^{-1}(V_A), V^{-1}(V_B)) = d_H(V_A, V_B)$, is also true. This completes the proof. $\square$

Example

Let $A = \{a, b\}$ and $B = \{b, c, d\}$. Their offspring O obtained by geometric crossover under symmetric distance are $\{b\} \subseteq O \subseteq \{a, b, c, d\}$. Dually, for the set A the vector of the values of the indicator function is $V_A = (1, 1, 0, 0)$ and for B is $V_B = (0, 1, 1, 1)$. The set of their offspring under traditional crossover is the schema $(*, 1, *, *)$. For the duality, these offspring vectors correspond to the offspring sets above via their indicator functions as it is easy to verify.

Fixed-size sets

In chapter 5, we have seen that permutations with repetitions are permutations in with each element is repeated a fixed number of times. The numbers of occurrences of each element define the repetition class of the permutation. *Binary permutations with repetitions* have only two elements occurring multiple times. For example, (01001) and (00011) are binary permutations with repetitions belonging to the same repetition class, with repetition classes $n_0 = 3$ and $n_1 = 2$.

We have seen, in section 6.3, that the substitution edit distance between fixed-size sets can be thought as a (rescaled) restriction of the symmetric distance between variable-size sets to fixed-size sets. The isometry between the metric space of variable-size set with symmetric distance and the metric space of binary strings with Hamming distance via the indicator function (theorem 6.4.1) can be restricted to map the metric space of fixed-size sets with substitution distance, with a restriction of the metric space of binary strings with Hamming distance. The corresponding indicator vectors to fixed-size sets of size n (subsets of the universal set U of size $|U|$) are binary permutations with repetitions of the same repetition class, with repetition classes $n_0 = |U| - n$ and $n_1 = n$. The restricted Hamming distance isometric to the substitution edit distance is the swap distance, that, as we have seen in chapter 5, it is a repetition-class preserving distance when applied to permutations with repetitions. Hence, the intersection/union crossover restricted to fixed-size sets is equivalent to sorting crossover by swap for binary permutations with repetitions.

6.4.2 Equivalence of crossovers for multisets and ordinal integer vectors

Variable size multisets

The isometry between the metric space of (variable-size) set with symmetric distance and the metric space of binary strings with Hamming distance via the indicator function (theorem 6.4.1) can be extended to the case of the isometry between (variable-size) multisets endowed with the ins/del edit distance and the space of (ordinal) integer vectors endowed with the absolute value distance (see chapter 9) via the multiplicity function. This isometry holds because the absolute value distance is, in fact, the same as the symmetric distance defined on the multiplicity vectors of the corresponding multisets, which in section 6.2.2, we have seen to be equivalent with the ins/del edit distance for multisets. Hence, the intersection/union crossover for multisets is equivalent to the integer blending crossover for integer vectors which is geometric crossover under absolute value distance (see chapter 9).

Fixed-size multisets

Analogously to the case of fixed-size sets, the isometry for fixed-size multisets can be seen as a restriction of the isometry associated with variable-size multisets. The multiplicity vectors of fixed-size multisets are integer vectors with constant sum. So, the space of fixed-size multisets endowed with the substitution edit distance is isometric via the multiplicity function to the space of integer vectors with constant sum endowed with the absolute value distance. Hence, the intersection/union crossover restricted to fixed-size multisets is equivalent to the integer blending crossover restricted to integer vectors with constant sum.

6.4.3 Equivalence of crossovers for partitions and cardinal integer vectors

Variable size partitions

The isometry between the metric space of (variable-size) set with symmetric distance and the metric space of binary strings with Hamming distance via the indicator function (theorem 6.4.1) can be extended to the case of the isometry between (variable-size) labeled partitions endowed with the partition move edit distance and the space of (cardinal) integer vectors endowed with the Hamming distance via the *partition label function*. This function is a generalization of the indicator function that returns, for each element in the universal set U, the label of the partition it belongs to. Hence, the partitionwise intersection/union crossover restricted to mutual exclusion and complete covering, which is geometric crossover under partition move edit distance (see section 6.2.3), is equivalent to the homologous crossover for (cardinal) integer vectors, which is geometric crossover under Hamming distance (see chapter 9).

Fixed-size partitions

Analogously to the case of fixed-size sets and fixed-size mutisets, the isometry for fixed-size partitions can be seen as a restriction of the isometry associated with variable-size partitions. The partition-label vectors, which generalize indicator vectors by using the partition-label function instead of the indicator function, of fixed-size partitions are permutations with repetitions belonging to the same repetition class. This is a straightforward generalization of the case of fixed-size sets which are associated with binary permutations with repetitions. So, the space of fixed-size partitions endowed with the partition swap edit distance is isometric via the partition label function to the space of permutations with repetitions endowed with swap distance. Hence, the partitionwise intersection/union crossover restricted to mutual exclusion, complete covering and same partition structure is equivalent to sorting crossover by swaps for permutations with repetitions.

Interestingly, the special case of the space of fixed-size n-partitions of sets of size n endowed with the partition swap edit distance is isometric via the partition label function to the space of simple permutations endowed with swap distance. Hence, the partitionwise intersection/union crossover restricted to mutual exclusion, complete covering and same partition structure is equivalent to sorting crossover by swaps for simple permutations such as PMX and cycle crossover.

6.4.4 Interesting uses of the duality

Thanks to these results we can use the two representations interchangeably. In particular, we can use the most convenient representation, knowing that the search done in one space is equivalent to the search in the other. For example, it is more convenient to work with partitions of both variable structure or fixed structure in their dual spaces based on permutations with repetitions because the constraints of mutual exclusion, full covering and structure preserving are much easier to deal with in operators defined on this space. We have exploited this property in chapter 11 on the graph partitioning problem. On the other hand, it may be more convenient to work with sets of small size (small compared to the size of the universal set) rather than on their dual vectors of fixed size (all the same size of the universal set).

6.5 Summary

We have considered three related representations – sets, multisets and partitions – in their variable size and fixed size variants.

For the variable size case, we have considered the ins/del edit distance. We have seen that for sets, this distance corresponds to the symmetric distance. Its extensions to the case of multi-sets and partitions correspond to the move edit distances for these two representations.

We have shown that the geometric crossover associated to the ins/del edit distance for sets is a crossover that requires offspring to be supersets of the intersection of the two parent sets and subsets of their union. The geometric crossovers associated to the ins/del distance for multisets and partitions are simple extensions of this intersection/union crossover for sets.

For the case of fixed-size sets, multi-sets and partitions, we have considered the substitution edit distance, that is equivalent to the ins/del edit distance when restricted to sets of fixed size. Therefore, the geometric crossover under substitution edit distance is a restricted version of the intersection/union crossover where all offspring are required to have fixed size. The geometric crossover associated to multisets and partitions for the fixed size case are analogous to the restricted intersection/union crossover for sets.

We have proved a duality between geometric crossover for sets, multisets and partitions on the one hand, and binary strings, integer vectors, and permutations with repetitions on the other. Interestingly, this allows the interchangeable use of representations and operators, being equivalent in terms of search, but exploiting their differences in terms of expressing constraints.

Table 6.1: Crossovers dual equivalence

		PRIMAL		DUAL
Representation:		Sets - variable size		Binary vectors
Map:		Indication function		
Distance:		Ins/del edit distance		Hamming distance
Crossover:		Inter/union crossover		Traditional crossover
Representation:		Sets - fixed size		Binary permutations with repetitions
Map:		Indication function		
Distance:		Substitution edit distance		Permutation swap edit distance
Crossover:		Inter/union crossover		Sorting crossover by swap
		restricted to fixed size		
Representation:		Multisets - variable size		Integer vectors
Map:		Multiplicity function		
Distance:		Ins/del edit distance		Absolute value distance
Crossover:		Inter/union crossover		Integer blending crossover
Representation:		Multisets - fixed size		Integer distributions
Map:		Multiplicity function		
Distance:		Substitution edit distance		Absolute value distance
Crossover:		Inter/union crossover		Integer blending crossover
		restricted to fixed size		restricted to constant sum
Representation:		Partitions - variable structure		Multary vectors
Map:		Partition label function		
Distance:		Partition move edit distance		Hamming distance
Crossover:		Partitionwise inter/union crossover		Traditional crossover
		restricted to mutual exclusion		
		restricted to complete covering		
Representation:		Partitions - fixed structure		Permutations with repetitions
Map:		Partition label function		
Distance:		Partition swap edit distance		Permutation swap edit distance
Crossover:		Partitionwise inter/union crossover		Sorting crossover by swaps
		restricted to mutual exclusion		
		restricted to complete covering		
		restricted to same structure		
Representation:		n-partitions of set size n		Permutations
Map:		Partition label function		
Distance:		Partition swap edit distance		Permutation swap edit distance
Crossover:		Partitionwise inter/union crossover		Sorting crossover by swaps,
		restricted to mutual exclusion		PMX, Cycle crossover
		restricted to complete covering		
		restricted to single element subsets		

Chapter 7

Syntactic Trees

The relationship between search space, distances and genetic operators for syntactic trees such as those used in genetic programming to represent computer programs is little understood. In this chapter we apply the geometric framework to the syntactic tree representation and show how a variation of the well-known structural distance for trees [35] is naturally associated with homologous crossover and sub-tree mutation.

7.1 Introduction

The fitness landscape associated with genetic operators for syntactic trees is little understood. In this chapter we provide the following contributions: a) Application of the geometric framework to the syntactic tree representation discussing the difference with other representations; b) Proof that the family of homologous crossovers [79] for syntactic trees are geometric crossover under a family of structural distances; c) Clarification of the structure of the search space associated with structural distances; d) Proof that the natural mutation operator associated with homologous crossover and structural distances is the sub-tree mutation operator; f) Corroboration that homologous crossover based on structural distances between syntactic trees is a meaningful genetic operator for genetic programming.

7.2 Crossovers and distances for trees

Let us now consider the tree representation and the class of homologous crossovers for trees.

7.2.1 Subtree swap crossover and homologous crossover

The *common region* is the largest rooted region where two parent trees have the same topology. In *homologous crossover* [79] parent trees are aligned at the root and recombined using a crossover mask over the common region. If a node belongs to the boundary of the common region and is a function then the entire sub-tree rooted in that node is swapped with it if the crossover mask says it should be swapped. One special case of homologous crossover is *one-point crossover* in which a common crossover point is picked randomly from the nodes belonging to the common region and then the two sub-trees rooted at the crossover point are swapped. In *subtree swap crossover* [77] any subtree of one parent can be exchanged with any subtree of the other.

7.2.2 Non-existence geometric crossover theorems

Theorem 7.2.1. *Subtree swap crossover is not a geometric crossover.*

See chapter 14 for the proof.

Theorem 7.2.2. *Homologous crossover is not a geometric crossover under graphic distance.*

Proof. For a graphic metric space, any segment of length 1 includes only the two end-points of the segment. So, the edges of the unique graph associated with the graphic metric space coincide with the segments of length 1. If by absurd homologous crossover was geometric under some graphic distance, then the previous property must hold for the metric space associated with homologous crossover. Let us consider the image sets under homologous crossover. The set of all possible offspring (image set) obtained by crossing over any tree with a tree consisting of only one node by homologous crossover is the set including the two parent trees, or, in case the two parent trees coincide, it is the set comprising the single node tree. This means that if the homologous crossover were associated to a graphic distance in the associated graph there would be an edge connecting any tree to a tree with any single node tree. In terms of associated distance we have only four possible cases: (i) distance zero, coinciding extremes, segment containing only the extreme; (ii) distance one, one extreme is single node tree and the other any other tree, segment containing only these two trees; (iii) distance one, the two extremes are not single node trees, segment containing only these two trees; (iv) distance two, the two extremes are not single node trees, segment may contain any trees but must contain all single node trees. This is because when the distance between two trees is two there is a shortest path between the two trees passing on a single node tree. Notice that when the distance between two trees is two, to be graphic, the segment between the two trees must contain a tree that differs from the extreme trees. Distance two is the maximum distance between two trees because is the maximum length of the shortest path connecting any two trees passing through a single node tree.

The image set obtained by crossing over two trees using homologous crossover may contain, beside the two parent trees, one or more offspring trees and does not need to contain any single node tree. In this case the distance between the two tree parents must be two, but there is no single node tree on the shortest path between these two trees, hence there is incongruence with condition (iv) above and the homologous crossover cannot be associated with a graphic distance □

Theorem 7.2.1 tells us that there is no distance naturally associable with the search space of subtree swap crossover. (See [56] for a different notion of distance for this operator.)

Because of Theorem 7.2.2 either homologous crossover is not a geometric crossover or homologous crossover is a geometric crossover based on a non-graphic metric space. If we find at least one distance that matches homologous crossover, then we know that homologous crossover is a geometric crossover and that the distance we found is a non-graphic distance.

7.2.3 Structural distance and hyperschemata

Ekart and Nemeth [35] defined an edit distance specific to genetic programming syntactic trees, adapted from [115]. Two trees are brought to the same tree structure by adding null nodes to each tree. The cost of changing one node into another can be specified for each pair of nodes or for classes of nodes. Differences near the root have more weight.

Definition 7.2.1. (Normalized structural hamming distance (SHD) for trees)
$dist(T_1, T_2) = \delta(p \neq q)$ if $arity(p) = arity(q) = 0$
$dist(T_1, T_2) = 1$ if $arity(p) \neq arity(q)$
$dist(T_1, T_2) = \frac{1}{m+1}(hd(p, q) + \sum_{i=1,m} dist(s_i, t_i))$ if $arity(p) = arity(q) = m$

With SHD when two subtrees are not comparable (roots of different arities) they are considered to be at a maximal distance. When two subtrees are comparable their distance is at most 1.

Theorem 7.2.3. *SHD is a metric strictly bounded by 1.*

Proof. **SHD bounded by 1**: we prove it by induction. It is clear that $dist(S, T) \leq 1$ when the arities of root nodes p and q of T and S are either both 0 (p and q are leaves) or different. Now suppose T and S have equal non-zero arities: $dist(S, T) = \frac{1}{m+1}(hd(p, q) + \sum_{i=1,m} dist(s_i, t_i))$ and suppose $dist(s_i, t_i) \leq 1, \forall i$ (induction hypothesis). Then since $hd(p, q) \leq 1$ we have $dist(S, T) \leq \frac{1}{m+1}(1 + \sum_{i=1,m} 1) = \frac{m+1}{m+1} = 1$
SHD is a metric:

identity: $dist(S,T) = 0 \leftrightarrow S = T$. (i) if $S = T$ then $dist(S,T) = 0$. This is true because recursively the distance between all coupled subtrees of S and T is 0. (ii) $dist(S,T) = 0$ implies that the item 2 in the definition of *dist* must not apply to any paired nodes otherwise the distance among two nodes becomes non-zero and consequently the distance of the whole trees becomes non- zero as well. Since for every paired nodes the trees S and T have the same arity then S and T have the same structure. It is easy to see that two trees with the same structure have $dist(S,T) = 0$ if and only if $hd(p,q) = 0$ for any paired nodes p and q i.e. $p = q$.

symmetry: $dist(S,T) = dist(T,S)$ is trivially true because dist is defined using symmetric functions.

triangular inequality: $dist(R,S) + dist(S,T) \geq dist(R,T)$. We prove it by induction on the depth of the tree.

Base case: suppose $depth(R) = depth(S) = depth(T) = 0$, so R, S and T have roots of arity zero. The triangular inequality holds in this case because *dist* degenerates to the hamming distance between roots for which the triangular inequality holds.

Induction hypothesis: suppose the triangular inequality is true if the depth of R, S and T is at most k. Verify induction implication: we now assume the tree among R, S and T that has the greatest depth, has depth $k + 1$. Let us consider in the following all possible cases.

- $arity(root(R)) \neq arity(root(T))$: in this case $dist(R,T) = 1$. We have two sub-cases: (i) $arity(root(R)) \neq arity(root(S)) = arity(root(T))$ in which case $dist(R,S) = 1$ and the triangular inequality holds; (ii) $arity(root(R)) \neq arity(root(S))$ and $arity(root(S)) \neq arity(root(T))$ in which case $dist(R,S) = 1$ and $dist(S,T) = 1$ so that the triangular inequality holds.

- $arity(root(R)) = arity(root(T)) = 0$: in this case $dist(R,T) = hd(R,T) \leq 1$. We have two sub-cases: (i) $arity(root(R)) = arity(root(S)) = arity(root(T)) = 0$, in which case dist degenerates to hamming distance and the triangular inequality holds; (ii) $arity(root(S)) > 0$, in which case $dist(R,S) = dist(S,T) = 1$, hence the triangular inequality holds.

- $arity(root(R)) = arity(root(T)) = m > 0$ and $arity(root(S)) \neq m$: in this case $dist(R,T) \leq 1$ and $dist(R,S) = dist(S,T) = 1$ because the root node of S in diverse in arity hence not comparable with R and T. Hence the triangular inequality holds.

- $arity(root(R)) = arity(root(S)) = arity(root(T)) = m$: $dist(R,S) + dist(S,T) = \frac{1}{m+1}(hd(R,S) + \sum_{i=1,m} dist(r_i,s_i)) + \frac{1}{m+1}(hd(S,T) + \sum_{i=1,m} dist(s_i,t_i)) = \frac{1}{m+1}(hd(R,S) + hd(S,T) + \sum_{i=1,m}(dist(r_i,s_i) + dist(s_i,t_i)))$ since $hd(R,S) + hd(S,T) \geq hd(R,T)$ and for induction hypothesis $hd(r_i,s_i) + hd(s_i,t_i) \geq hd(r_i,t_i)$ then $dist(r_i,s_i) + dist(s_i,t_i) \geq \frac{1}{m+1}(hd(R,T) + \sum_{i=1,m} dist(r_i,t_i)) = dist(R,T)$

□

The hyperschema [79] associated with two trees is the tree structure that has the topology of the common region of the two trees; its nodes are '=' when two matched nodes differ in the content, or '#' replacing two subtrees whose roots are matched but their arities differ, or the node itself when the two matched nodes coincide. Figure 7.1 illustrates the relation between parent trees, hyperschema and offspring trees and shows: at the top, two parent trees $P1$ and $P2$; at the bottom on the left, their associated hyperschema $H(P1, P2)$; at the bottom on the right, all the potential offspring obtained by applying homologous crossover to parents $P1$ and $P2$ (the part in bold means alternative content of the tree; in this case there are 5 independent binary alternatives, resulting in 32 possible offspring).

The SHD distance between two trees is a function of only the hyperschema associated with the two trees and not directly of the two trees. This is because it can be calculated using only the information contained in the hyperschema associated with the two trees without requiring the explicit knowledge of the trees themselves. The examples in figure 7.2 shows how to calculate SHD from an hyperschema. The contributions to the distance SHD of the wildcards '=' and $'\#'$ is 1, the contributions of the the other nodes is 0. The contributions are aggregated recursively, starting from the leaves of the hypershema, and weighted according to the branching factor of the node under consideration. In the example, starting from the bottom left, d gives contribution 0 because it is not a wildcard. The node next to it is $'\#'$ which gives a contribution of 1 being a wildcard. The parent node of the previous two nodes is '=', which also gives a contribution of 1 being a wildcard. The contribution of the subtree including the previous three nodes is the average of their contributions, which is $\frac{0+1+1}{3} = \frac{2}{3}$. This subtree now is considered as a single node with contribution $\frac{2}{3}$. The parent node of the subtree is a which gives contribution 0. The contribution of the subtree rooted in a is the average of the contribution of a and the contribution of its offspring node. So, it is $\frac{0+\frac{2}{3}}{2} = \frac{1}{3}$. And, so on. The final contribution of the overall hyperschema equals the distance between the two trees.

7.2.4 Geometric crossover theorems

Theorem 7.2.4. *Homologous crossover is a geometric crossover under SHD.*

Proof. As shown in figure 7.2, the distance between two trees $P1$ and $P2$ is function, d, of the hyper-schema $H(P1, P2)$ identified by the two trees: $SHD(P1, P2) = d(H)$.

Every offspring of two trees is obtained by substituting each wildcard characters in the hyper-schema with a node (in the case of '=') or a sub-tree (in the case of $'\#'$) coming from either

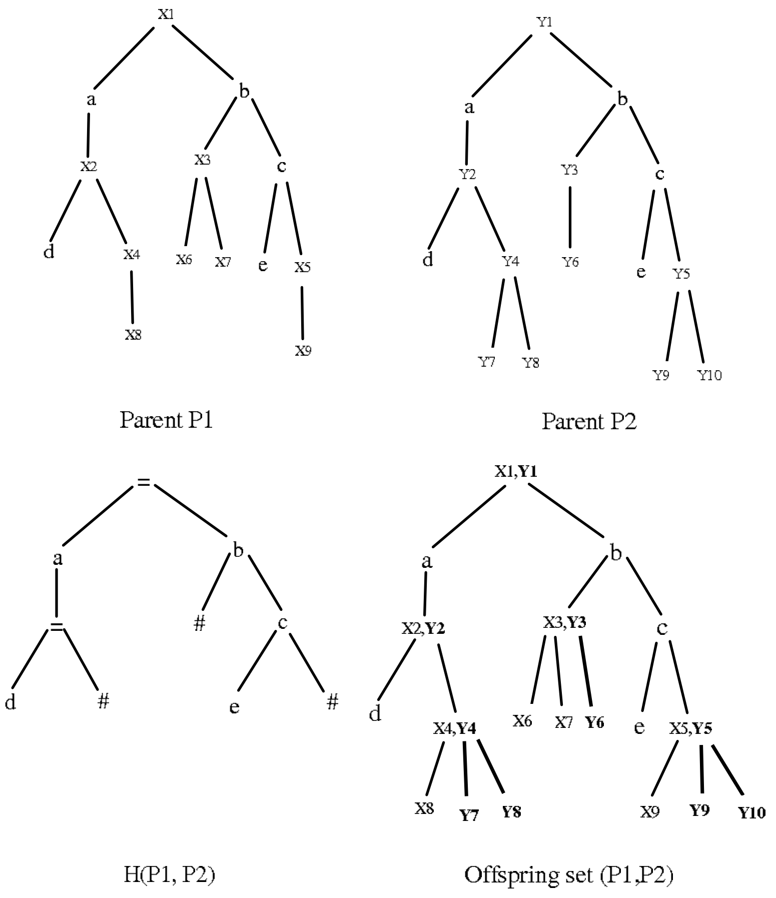


Figure 7.1: Hyperschema and offspring set

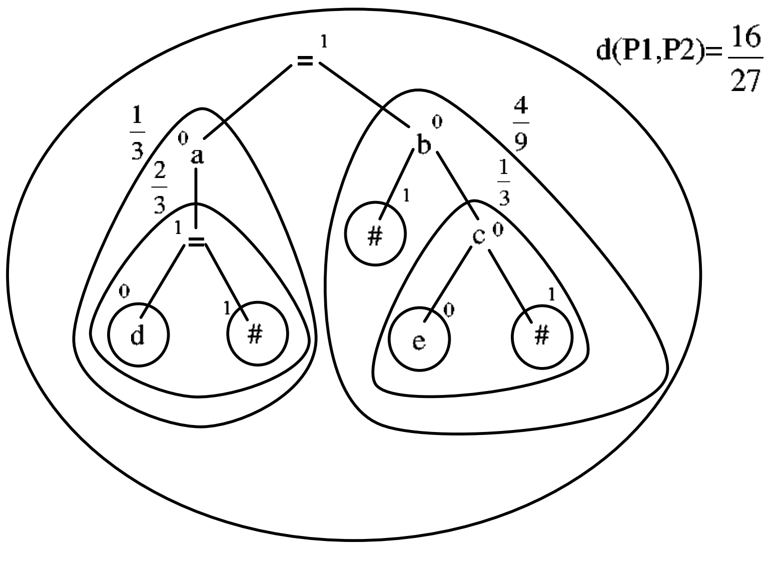


Figure 7.2: Hyperschema and structural distance

parent at that specific position.

Let $p_1, ..., p_n$ be the positions in the structure of H of the wildcard characters. Then the distance $d(H)$ can be decomposed into a sum of distances that are only functions of the positions of the wildcard characters in the tree: $d(H) = d(p_1) + ... + d(p_n)$.

Let O be the offspring of $P1$ and $P2$. Then, the hyper-schema $H(P1, O)$ is obtainable by turning some wildcard characters in $H(P1, P2)$ to corresponding nodes/sub-trees from parent $P1$. The hyper-schema $H(O, P2)$ is obtainable by turning the wildcard characters in $H(P1, P2)$ left untouched into corresponding nodes/sub-trees from parent $P2$.

The positions of wildcard characters in $H(P1, O)$, say $\{p_i\}$, and in $H(O, P2)$, say $\{p_j\}$, are complementary, which is there is no i and j such as $p_i = p_j$, and taken all together are the same as in $H(P1, P2)$, which is $\{p_i\} \cup \{p_j\} = \{p_1, ..., p_n\}$.

Therefore, we have that $d(H(P1, O)) + d(H(O, P2)) = d(\{p_i\}) + d(\{p_j\}) = \sum d(p_i) + \sum d(p_j) = d(p_1) + ... + d(p_n) = d(\{p_1, ..., p_n\}) = d(P1, P2)$.

This means that every offspring O of $P1$ and $P2$ is in the segment between $P1$ and $P2$ under *dist*. □

Theorem 7.2.5. *The image set of the class of homologous crossovers is the segment between the two parent trees under SHD.*

Proof. We need to prove that O in $[P1, P2]$ implies O in the image set of homologous crossover $R(P1, P2)$. Let us assume by absurdum that O in $[P1, P2]$ but not in $R(P1, P2)$. Then $d(P1, O) + d(O, P2) = d(P1, P2)$ and either (i) O matches $H(P1, P2)$ but does not take the node/sub-tree of either parents at (at least) one position in H or (ii) O does not match $H(P1, P2)$. In case (i) the positions of wildcard characters in $H(P1, O)$ and $H(O, P2)$ are not complementary but their union still equals the positions of the wildcards in $H(P1, P2)$. This means that some of the wildcard positions are present in both $\{p_i\}$ (which are the positions of wildcard characters in $H(P1, O)$) and $\{p_j\}$ (which are the positions of wildcard characters in $H(O, P2)$) implying that the sum of the associated distances is greater than the distance associated with their union; hence $d(P1, O) + d(O, P2) > d(P1, P2)$. In case (ii) there are two sub-cases: (a) O does not match $H(P1, P2)$ but matches its structure; this happens when some nodes of O do not match the corresponding non-wildcard characters in H. (b) O does not match the structure of $H(P1, P2)$; this happens when some nodes of O do not have the same arity of the corresponding node in H. In sub-case (a) the positions $\{p_i\}$ and $\{p_j\}$ of the wildcard characters in $H(P1, O)$ and $H(O, P2)$ respectively, both contain the positions of the mismatch with O plus, each one, a complementary bipartition of the set of positions $\{p_1, ..., p_n\}$. Since the distance associated with $H(P1, P2)$ is additive function of the set of positions $\{p_1, ..., p_n\}$ and the union of $\{p_i\}$ and $\{p_j\}$ is a proper subset of $\{p_1, ..., p_n\}$ then the sum of the distances associated with $H(P1, O)$ and $H(O, P2)$ is greater than the one associated with $H(P1, P2)$. In the sub-case (b) O differs (in arity) at certain position form both parents hence $H(P1, O)$ and $H(O, P2)$ are obtained by, first, pruning $H(P1, P2)$ at the node in which O differs in arity and put a wildcard character and, then, substituting some of the wildcards with some nodes/sub-trees at the corresponding position from $P1$ and $P2$ respectively. The pruning of $H(P1, P2)$ always produces an hyper-schema the associated distance of which is greater or equal to the one associated to the $H(P1, P2)$ un-pruned. This is because the weight associated with a wildcard substituting a sub-tree is an upper-bound of the contributions of the sum of the weights of any possible sub-tree put in that position. The positions of the wildcards in $H(P1, O)$ and

$H(O, P2)$ are complementary except for the wildcard attached at the position of the pruned tree, that appears in both trees. This wildcard, therefore, contributes to both the distances associated with $H(P1, O)$ and $H(O, P2)$ and being un upper bound of the contributions in the sub-tree of $H(P1, P2)$ it replaces, we have $H(P1, O) + H(O, P2) > H(P1, P2)$ also in this last case. □

7.2.5 Analysis of the normalization coefficient

The normalizing value $n = \frac{1}{m+1}$ in SHD has been chosen to have a strict upper bound at 1. It also provides consistency in the distance between two fully different subtrees in the following two senses: (i) any two sub-trees that have the same structure but differ in all nodes must have distance 1 (ii) any two subtrees that are incomparable must have distance 1. For smaller values of n, SHD is still a metric but the upper bound 1 is never reached. This gives greater distance to subtrees that are non-comparable than to subtrees that are fully-comparable but differ in all nodes. For n slightly bigger than $\frac{1}{m+1}$, SHD is still a metric, not upper bounded by 1, but still upper bounded. Between $\frac{1}{m+1}$ and 1 there is a critical value of n where SHD ceases to be a metric. In the following we consider SHD with $n = 1$, that we call *Hamming distance* (HD) *between syntactic trees.* (HD can be also seen as the number of mismatching nodes at corresponding positions within the common region.)

Theorem 7.2.6. *HD is not a metric*

Proof. Let us consider three syntactic trees, $T1$, $T2$ and $T3$. $T1$ consists only of a single terminal node. $T2$ and $T3$ have the same shape and size k but they differ in all matching nodes. If HD is a distance the triangular inequality must hold for any choice of $T1$, $T2$ and $T3$. In the specific case of our example the following must hold $HD(T2, T1) + HD(T1, T3) \geq HD(T2, T3)$. Since we have $HD(T2, T1) = 1$, $HD(T1, T3) = 1$ and $HD(T2, T3) = k$, it is immediate to see that for $k > 2$ the triangular inequality fails to hold. Hence, HD is not a metric □

Looking at the proofs of theorem 7.2.4 and 7.2.5, it is easy to see that they work for $0 < n < \frac{1}{m+1}$. So there is a whole family of distances that match homologous crossover.

7.3 Graphic space vs non-graphic space

In the previous section we have shown that homologous crossover is geometric but non-graphic. The uniqueness results of geometric crossover presented in chapter 3 are for graphic distances, so

Table 7.1: Graphic space and graphic operators.

Graphic	representation	structure	distance	mutation	crossover
Representation	-	many	many	many	many
Structure	many	-	1	1	1
Distance	many	1	-	1	1
Mutation	many	1	1	-	1
Crossover	many	1	1	1	-

Table 7.2: Non-graphic space and non-graphic operators.

Non-graphic	representation	**weighted** structure	distance	mutation	crossover
Representation	-	many	many	many	many
Weighted structure	many	-	1	1	1
Distance	many	**many**	-	1	1
Mutation	many	**many**	**many**	-	**many?**
Crossover	many	**many**	**many**	**many?**	-

they do not necessarily hold for tree homologous crossover. In the following, we make a digression and consider some general, fundamental differences between geometric operators defined over graphic metric spaces and geometric operator defined over non-graphic metric spaces, which are relevant also for the specific case of tree homologous crossover.

Tables 7.1 and 7.2 summarise the cardinality of the relations between solution representation, neighbourhood structure, distance, geometric mutation and geometric crossover in the case of graphic and non-graphic spaces, respectively. The rows labelled "representation" in the two tables are identical and tell us that to a solution representation may be associated: (i) more than one neighbourhood structure; (ii) more than one distance because each neighbourhood structure induces a path metric; and consequently (iii) more than one type of geometric mutation operator and (iv) more than one type of geometric crossover operator because geometric operators are functions of the distance, and, so, there are as many types available as the distances for the same representation.

The rows labelled (neighbourhood) "structure" have the same contents in the two tables.

This tells us that there can be different representations that induce the same neighbourhood structure; the three 1's in this row are due to the fact that distance, mutation and crossover are functions of the neighbourhood structure and this does not depend on the type of underlying structure.

The row labelled "distance" says that there may be more than one representation associated to the same distance for both graphic and non-graphic spaces. The first "1" in that row, in table 7.1, tells us that a graphic distance has a unique graphic representation. In table 7.2 in the same cell we find 'many', meaning that for any non-graphic distance there is no simple graph representation but instead there are many possible weighted graphs representations. *So, the notion of unique and discrete search space structure, like the hyper-cube for the Hamming distance for binary strings for example, in the case of non-graphic distances is lost.* The following two 1s in the row, in both tables, are there because mutation and crossover are function of the distance, so they are unique to it.

The row labelled "mutation" tells us that the same mutation operator, graphic or non-graphic, may arise from different solution representations. Given a graphic mutation operator is always possible to determine the full structure of its underlying graphic space and, hence, its associated graphic distance and its associated graphic crossover. This uniqueness result is possible because of the graphic character of graphic mutation and it is not valid in general (for details see chapter 3). The situation for non-graphic mutation is quite different: passing from a weighted graph (structure of the search space) to its induced non-graphic metric space, there is a loss of information; there is a further loss of information when passing from the distance to its induced non-graphic mutation. The same reasoning applies to crossover (last row in the tables). *Hence, for non-graphic operators, the theorem of uniqueness of distance and space structure for crossover and mutation, which holds in the graphic case, ceases to hold, opening up to the possibility of more than one distance and space structure associated with the same non-graphic operator.*

In essence table 7.1 tells us that no matter what graphic element one knows - space structure, distance, mutation or crossover - one can always determine any other. Table 7.2 tells us that for non-graphic spaces the weighted structure has more information than the induced distance that, in turn, embeds more information than the induced mutation and crossover operators. Since the mutation-crossover isomorphism theorem for graphic operators relies on the uniqueness of their underlying distance, the one-to-one mapping between non-graphic mutation and non-graphic crossover is not provable in this way.

The fact that homologous crossover for syntactic trees is non-graphic does not preclude the possibility of a graphic crossover for syntactic trees based on a graphic distance between trees. [119] proposed a simple extension of Levinsthein distance for sequences to syntactic trees, that is indeed a graphic distance. The geometric crossover based on such a distance is, therefore, an example of graphic crossover for syntactic trees (however, the geometric crossover based on such a distance is allowed to generate infeasible offspring and this may be undesirable). *So, the non-graphic attribute is attached to the distance and to the genetic operators based on it; it is not part of the underlying representation nor the geometric operators for a given representation.*

7.4 SHD mutation

What is the mutation operator associated to homologous crossover? We have seen in section 7.3 that is not clear weather or not the one-to-one mapping existing between graphic crossover and graphic mutation extends to non-graphic operators. However, since both mutation and crossover are defined as functions of a distance, we will consider one mutation operator that is connected to the homologous crossover through the SHD metric. We should bear in mind, though, that there may be other mutation operators connected to it through other distances.

Vanneschi [156] introduced structural mutation operators for syntactic trees and proved that their operators are consistent in some sense with the structural distance. In the following we discuss the geometric mutation operator defined over the SHD, that is a variation on the structural distance. *That is, we consider the potential mutated offspring of a tree as those trees that are within the ball of radius ϵ centred on the parent tree.*

Unlike geometric crossover that partitions the set of all binary genetic operators into two clear-cut categories, crossovers and non-crossovers (see chapter 14), geometric mutation has a continuous character and any unary operator is a geometric mutation under some distance. The point is to understand how a syntactic change affects the "amount" of mutation (i.e., the distance between the parent and the offspring) under a given distance. So the questions to ask are: what syntactic change is a "micro"-mutation under SHD? And what other syntactic change is a "macro"-mutation? How much a specific syntactic change affects the amount of mutation?

To understand the peculiarity of SHD mutation we compare it with mutation for binary strings. For binary strings the mutation is:

- *non-positional*: mutating any locus results in the same amount of mutation
- *proportional to the syntactic change*: lots of bit changed = lots of mutation
- *based on single-type mutation*: bit-flip only
- *additive*: two bit changed add up in terms of contribution (provided you do not flip the same bit)

For trees the amount of mutation associated with SHD is:

- *positional*: the amount of mutation depends on the depth at which the mutation occurs: the deeper the level, the smaller the mutation (smaller SHD between parent and mutated parent); it depends also on the branching factor of the path from the root node to the node at which mutation takes place: the bigger the branching factor, the smaller the mutation. If we want to restrict the mutation to be within a certain distance from the parent tree, this can be done approximately by picking mutation sites below a certain level in the tree. If we take as a mutation site every node in the parent tree with uniform probability on the node of the tree, we allow for maximal macro-mutation (changing the root of the tree produces a tree a maximal distance (distance 1)) with low probability and micro-mutation with higher probability since the number of nodes increases geometrically with the depth of the node in the tree.

- *Non-proportional to syntactic change*: a big mutation at a big depth may be smaller than a small mutation closer to the root
- *Based on various types of mutation* [8]:
  - Point mutation [79]: node substitution at a specified position in the tree
  - Subtree-prune mutation [5]: a sub-tree is substituted by a terminal node
  - Subtree-grow mutation [5]: a terminal node of the tree is replaced with a sub-tree
  - Subtree mutation [5]: a sub-tree is substituted by another sub-tree

    All edit moves considered above are degenerated forms of sub-tree edit move
- *Weighted additive and coherent*: the unrestricted sub-tree edit move, which degenerates to specific *coherently and additively weighted edit moves* as special cases.

7.5 Tree interpretation and smooth landscape

In previous sections we have described the search space associated with genetic operators for syntactic trees. In the following we discuss how such a search space and its fitness connect together giving a picture of the fitness landscape in its entirety.

In chapter 3 we mentioned that what is really important for an algorithm to perform better than random search is how problem and algorithm are connected via distance. In chapter 5 we have suggested that if one picks a distance that makes sense for the entity represented (phenotype) by the solution (genotype), then the geometric crossover defined over this distance is likely to perform well. The logic is the following: closer genotypes imply closer phenotypes that in turns imply closer fitness. This allows for a smooth fitness landscape that is good for most meta-heuristics based on neighbourhood search [48]. Naturally, this is a rule of thumb and not a proven theorem. A question comes to mind: does the SHD metric associated with homologous crossover make sense when syntactic trees are interpreted as GP programs? Is it a meaningful distance in terms of GP programs?

Because of the way solutions are encoded in GP and since often information propagates in the tree from the leaves (some of which might never be reached during evaluation of a solution) to the root node (that is always considered), the nodes near the root of the tree are typically much more used, hence important, than nodes at lower levels. Such an interpretation of a syntactic tree is very different from that given to other types of tree-like structures. For example, to find the minimum spanning tree of a graph one can encode spanning trees of a graph as tree structures. In this case every node of the tree has presumably the same importance because of the way the problem is encoded. The syntax of the two representations above is similar, but the part of their syntax having an impact on the phenotype (interpretation) is completely different.

In section 7.4, we have seen that the distance associated to homologous crossover assigns a greater weight, for the same amount of syntactic change, to the top of the tree and smaller weight to the bottom of the tree. This goes well with the previously mentioned landscape design principle in that, when the tree is interpreted as a GP program, changes at upper levels of the tree have a much higher impact on the behaviour of a program than changes at lower levels. In turn, the impact on the behaviour is reflected on the fitness. So, programs that are modified at an upper level have much higher probability to behave completely differently and, therefore, to have very different fitnesses than programs that are neighbours for a modification at a lower level in the tree.

Homologous crossover for syntactic trees is therefore a very natural choice when the trees are interpreted as GP programs as it induces a smoother landscape that is likely to facilitate the search.

7.6 Summary

We have shown that the geometric framework naturally connects the notions of homologous crossover, subtree mutation, hyperschema and structural distance for syntactic trees. We have also described the structure of the space of syntactic trees associated with these elements and argued that, when using the standard interpretation of syntactic trees as programs, the associated landscape is smoother, hence the homologous crossover is a good choice.

To conclude we want to emphasise the significance of the present results in the larger context of our on-going programme of evolutionary algorithms unification: most of the pre-existing genetic operators for binary strings, permutations, real vectors and now also some operators for syntactic trees, all fit naturally the geometric framework hence implying a profound geometric unity of all major flavours of existing evolutionary algorithms.

Chapter 8

Sequences

This chapter extends the geometric framework for interpreting crossover and mutation to the case of sequences. This representation is important because it is a link between artificial evolution and biological evolution. We define and theoretically study geometric crossover for sequences under edit distance and show its intimate connection with the biological notion of sequence homology.

8.1 Introduction

EAs mimic, in a simplified manner, natural evolution. However, very few theoretical results are available which apply equally well to both forms of evolutionary search.

One important cause of the lack of connection between evolutionary computation theory and evolutionary biology is that they focus on different kinds of genotypes (different solution representations), namely DNA strands (variable-length strings or sequences) and binary strings. Most importantly, even if DNA strands and binary strings appear to be very similar at a first sight, the crossover operator for binary strings is just a caricature of the biological recombination acting on DNA strands. The main difference is that DNA strands align on the basis of their contents (during meiosis) before exchanging genetic material. They do not align only positionally as it is the case for binary strings. The DNA's alignment is flexible in that two DNA strands can stretch and fold to better align with each other. Moreover, DNA strands do not need to be aligned at the extremities. After alignment, the two DNA strands split in one or more regions in which they match well and exchange DNA segments. This last phase is present in crossovers for EAs, in which, however, typically no alignment process based on content takes place.

Is biological recombination geometric? In this chapter we are able to answer this question in the affirmative by extending the geometric framework mentioned above to sequences under edit distance. This has the remarkable consequence that the theory of geometric crossover applies to biological crossover as well, bridging the gap between biological evolution and artificial evolution. Our results reveal a deep connection between crossover for binary strings and biological recombination, showing that standard EA crossover is less of a caricature than it appears at first sight.

The chapter is organised as follows. In section 8.2, we show that, in the case of sequences endowed with edit distances, geometric crossover is a form of homologous crossover which performs the alignment on sequence contents before mixing genetic material. We prove various properties of this crossover and, in section 8.3, extend it to weighted alignments and alignment with gaps. In section 8.4, we argue that biological recombination is geometric and discuss the consequences of this.

8.2 Geometric crossover for sequences

In this section, we extend the geometric framework to the case of sequences. In particular we will focus on edit distances that associate with sequence homology.

8.2.1 Preliminaries: sequences, edit distance and alignments

A sequence is a variable length string of characters. In particular, DNA strands are sequences of characters from the alphabet $\Sigma_{dna} = \{a, c, t, g\}$. The *edit distance* between two sequences is defined as the minimum number of edit operations – insertions, deletions, and substitutions – needed to transform the first string into the second. The edit distance is a metric in that it respects all the metric axioms (chapter 3). Hence, the space of sequences endowed with edit distance is a metric space. There are a number of extensions to the simple edit distance such as weighted edit distance, block-edit distance, reversals and transpositions distances (see Sections 8.3 and 8.4 for a discussion on their use). The edit distance between two sequences is a measure of their syntactic dissimilarity. This syntactic dissimilarity is intimately connected with the notion of sequence alignment.

In the following, we introduce some preliminary concepts about sequence alignments that are well-known in bioinformatics. The book by Gusfield [55] is a good introduction to sequence alignments. An *alignment* of two sequences is obtained by first appropriately inserting spaces (which we represent with dashes), either into or at the ends of the two sequences, and then placing the two resulting sequences one above the other so that every character or space in one sequence is aligned with a character or space in the other sequence. The *score of an alignment* is the number of aligned characters that are different in the two sequences. There may be more that one optimum alignment between two sequences. The score of an optimum alignment of two sequences equals their edit distance. Changing the scoring system, one can obtain optimal alignments associated to weighted edit distances and block-edit distances. Edit distances and optimal alignments can be computed efficiently using dynamic programming.

The (edit) *transcript* T associated to an alignment q is a vector that specifies what edit move to apply at each position to transform parent 1 into parent 2 . For each alignment, q, there is only one transcript T and *vice versa*. For example, $T = (\texttt{RIMDMDMMI})$ and $q = \begin{pmatrix} \texttt{v-intner-} \\ \texttt{wri-t-ers} \end{pmatrix}$, where `R`, `I` and `D` stand for replace, insert, delete, respectively, while `M`, which stands for match, is just a place holder (a no-op instruction).

8.2.2 Homologous crossover and geometric crossover

Homologous crossover for sequences has been introduced by [123] in the context of linear GP. We formalise and generalise it, we prove that it is geometric crossover and then we list some of its properties.

Definition 8.2.1. (Alignment-based homologous crossover operators)

1. Let Q be the set of all optimal alignments of two sequences S_1 and S_2 under simple edit distance. Homologous crossover picks a random optimal alignment $q = \binom{\bar{S_1}}{\bar{S_2}} \in Q$ with a

given probability distribution over Q where $\overline{S}_1$ and $\overline{S}_2$ are the two sequences aligned with gaps according to q.

2. Let l be the length of q and m be a mask drawn from $\{0,1\}^l$ with a given probability distribution. m specifies for each position of q from which parent to copy the corresponding character to produce an aligned offspring $\overline{S}_3$

3. The actual offspring S_3 is obtained by remove the dashes from $\overline{S}_3$.

For example, if $S_1 = \texttt{agcacaca}$ and $S_2 = \texttt{acacacta}$ and the chosen optimal alignment is $q = \begin{pmatrix} \texttt{agcacac-a} \\ \texttt{a-cacacta} \end{pmatrix}$ then $l = 9$, $\overline{S}_1 = \texttt{agcacac-a}$ and $\overline{S}_2 = \texttt{a-cacacta}$. If $m = \texttt{111100000}$ we obtain the offspring $\overline{S}_3 = \texttt{a-cacac-a}$. After gap removal we obtain $S_3 = \texttt{acacaca}$.

Theorem 8.2.1. *All alignment-based homologous crossover operators are geometric crossovers under edit distance.*

Proof. An optimal edit transcript T contains a smallest set E of edit moves to transform u in v. Note that $\texttt{M}$ is not an edit move and it is not included in E. $|E| = d(u,v)$. The edit moves in E are independent because they can be applied in any order to transform u into v. Any intermediate sequence z obtained by applying a subset $E' \subseteq E$ of edit moves to u is on a shortest path between u and v because z is $d(u,z) = |E'|$ moves away from u and $d(z,v) = |E \setminus E'|$ moves to v. Hence, $d(u,z) + d(z,v) = d(u,v)$. A mask m selects a subset of edit moves $E_m \subseteq E$ from the transcript T to apply to u and produce the offspring z. Hence z is on the shortest path. □

Theorem 8.2.2. *Every sequence O in the segment between two sequences P_1 and P_2 under edit distance is reachable by homologous alignment-based crossover applied to the parent sequences P_1 and P_2.*

Proof. We need to prove that for each $O \in [P_1, P_2]_{ed}$ there exists an optimal alignment q of P_1 and P_2 and a mask m that applied to q gives O. We prove it by constructing q and m given any O.

If $O \in [P_1, P_2]_{ed}$ then there exists a shortest path sp between P_1 and P_2 in the search space of sequences endowed with the edit distance such that $O \in sp$. Then there exists a transcript T such as all the edit moves in T are the same of the set of edit moves that generate sp. The transcript T may comprise also one or more M characters that do not correspond to any edit move. The transcript T is optimal by construction because the number of edit moves in T (non-M characters) is exactly $ed(P_1, P_2)$.

Given T, P_1 and P_2, it is possible to build the unique alignment q of P_1 and P_2 associated with T. The alignment q is optimal because T is optimal. Consider now the crossover mask m of the same length of the transcript T obtained by setting at 1 the loci corresponding to those edit moves in the transcript T that transform P_1 into O. The crossover mask m applied to the optimal alignment q produces O in the path sp. □

Theorems 8.2.1 and 8.2.2 establish that a crossover for sequences is an alignment-based homologous crossover if and only if it is a geometric crossover under simple edit distance.

8.2.3 Optimal alignments and segment subsets

The family of crossovers introduced in the previous section can be seen as an extension to sequences of the family of alignment-based crossovers for fixed-length binary strings. In chapter 3 we proved that for binary strings, uniform crossover, where crossover masks are obtained by flipping n times a unbiased coin, picks offspring with uniform probability distribution on the line segment between parents under Hamming distance. In this section we introduce a generalisation of uniform crossover based on masks for sequences and show that, unlike the binary string case, this crossover, in general, does not pick offspring uniformly in the segment between parents under edit distances.

Definition 8.2.2. (Uniform alignment-based homologous crossover) Uniform homologous crossover is an alignment-based crossover operator that chooses optimal alignments and crossover masks with uniform probability.

As an example, in Table 8.1, we enumerate all possible offspring under homologous crossover of the sequences “vint” and “writ”. For these sequences there are three possible optimal alignments. The edit distance between the sequences is 3. This can be seen also from the edit transcript associated to each optimal alignment in which there are 3 non-`M` characters. These characters describe the edit operations and the location of their application on the alignment to transform the first sequence into the second one. In the first column, all the possible crossover masks are shown. We report only the bits corresponding to the three non-`M` symbols, thereby obtaining 8 effective crossover masks (out of the 32 possible masks). The entry at the intersection of a row (effective crossover mask) and a column (optimal alignment) contains the offspring obtained by the application of the mask to the alignment. Alignment-based uniform crossover returns any of the offspring in the table at random with uniform probability ($\frac{1}{24}$). However, some offspring can be generated by more than one alignment, and, so, they have higher chances to be picked. “vint” and “writ”, for example, are produced with a probability $\frac{1}{8}$, while “vit”, “wrint”, “vrit” and “wint” are returned with probability $\frac{1}{12}$.

Definition 8.2.3. The image set of an optimal alignment q is the set of offspring that can be generated by homologous crossover using all possible masks m over q.

Table 8.1: Possible offspring under uniform alignment-based homologous crossover.

	Alignment 1	Alignment 2	Alignment 3
mask	`mm*m*`	`mm*m*`	`mmm*`
transcript	`IRMDM`	`RIMDM`	`RRRM`
parent 1	`-vint`	`v-int`	`vint`
parent 2	`wri-t`	`wri-t`	`writ`
000	`-vint`	`v-int`	`vint`
001	`-vi-t`	`v-i-t`	`viit`
010	`-rint`	`vrint`	`vrnt`
011	`-ri-t`	`vri-t`	`vrit`
100	`wvint`	`w-int`	`wint`
101	`wvi-t`	`w-i-t`	`wiit`
110	`wrint`	`wrint`	`wrnt`
111	`wri-t`	`wri-t`	`writ`

Theorem 8.2.3. *Consider the image sets $Im(q_1)\dots Im(q_n)$ of homologous crossover applied to all optimal alignment $q_1 \dots q_n$ of the sequences P_1 and P_2. The union of $Im(q_1)\dots Im(q_n)$ is $[P_1, P_2]$ but they do not form a partition of $[P_1, P_2]$.*

Proof. For theorem 8.2.1, the image set of any optimal alignment is subset of the segment. For theorem 8.2.2, any sequence z in the segment $[P_1, P_2]$ can be generated by homologous crossover. Hence, there must exist at least one alignment such that its image set includes z. This means that every point in the segment is at least in some $Im(q_i)$. Hence, the union of all $Im(q_i)$ is the segment $[P_1, P_2]$. Proof by counterexample: table 8.1 shows that the $Im(q_i)$ do not form a partition of the segment $[P_1, P_2]$ because there are some pairs whose intersection is non-empty. □

Theorem 8.2.4. *Uniform alignment-based homologous crossover is not the uniform geometric crossover under edit distance.*

Proof. Proof by counterexample: table 8.1 shows that the frequency of some offspring sequences under uniform homologous crossover is higher than others. So, the probability is not uniformly distributed over the segment. □

The non-uniformity of this crossover is the result of the same offspring sequence being generated by multiple different optimal alignments. Parent sequences, for example, are in this category because they can be generated by all optimal alignments using masks 0...0 and 1...1. Other offspring sequences can be generated more than once when two optimal transcripts share non-`M` characters at the same positions. For example, if two transcripts have a `D` at position 1, then the mask 0X...X where X...X is either 0...0 or 1...1 will produce the same offspring with both alignments. The mask 1X...X will have the same effect.

8.2.4 Bounds on offspring size

In this section we explore how offspring and parent sizes are related in homologous crossover.

Theorem 8.2.5. *Given two parent sequences P_1 and P_2 of length l_1 and l_2 with $l_1 \leq l_2$ and edit distance $d_p = d(P_1, P_2)$, the length l_3 of any offspring sequence O obtained by homologous recombination is bounded as follows:*

1. *Edit distance d_p known: $(l_1 + l_2 - d_p)/2 \leq l_3 \leq (l_1 + l_2 + d_p)/2$*
2. *Edit distance d_p not known: $l_1/2 \leq l_3 \leq l_1/2 + l_2$*
3. *Parents of same length $l_1 = l_2 = l$: $l/2 \leq l_3 \leq 3l/2$*
4. *Non-empty parents imply non-empty offspring*

Proof. Trivial edit distance bounds: (i) $d(a, b) \geq |l(a) - l(b)|$ and (ii) $d(a, b) \leq max(l(a), l(b))$. From bound (i) applied to P_1 and O: $d(P_1, O) \geq |l_1 - l_3|$ that breaks into two cases: (1) $l_1 - l_3 \leq 0 \rightarrow l_1 \leq l_3 \leq d(P_1, O) + l_1$ (worst case upper bound) (2) $l_1 - l_3 \geq 0 \rightarrow l_1 - d(P_1, O) \leq l_3 \leq l_1$ (worst case lower bound). Analogously, applying bound (i) to P_2 and O we obtain other two alternative cases: (3) $l_2 - l_3 \leq 0 \rightarrow l_2 \leq l_3 \leq d(P_2, O) + l_2$ (worst case upper bound) (4) $l_2 - l_3 \geq 0 \rightarrow l_2 - d(P_2, O) \leq l_3 \leq l_2$ (worst case lower bound).

Let us consider the upper bound for l_3. Both the conditions (1) and (3) must hold true, so $2l_3 \leq d(P_1, O) + d(P_2, O) + l_1 + l_2$. For all O: $d(P_1, O) + d(P_2, O) = d(P_1, P_2) = d_p$. Hence for all O: $l_3 \leq (l_1 + l_2 + d_p)/2$. If the distance d_p between parents P_1 and P_2 is unknown we can use bound (ii) to bound it: $d_p \leq max(l_1, l_2) \rightarrow d_p \leq l_2$. Hence for all O in the worst case we have: $l_3 \leq l_1/2 + l_2$. In case $l_1 = l_2 = l$ we have for all O: $l_3 \leq 3l/2$.

Let us consider the lower bound for l_3. Both the conditions (2) and (4) must hold true, so $l_1 + l_2 - (d(P_1, O) + d(P_2, O)) \leq 2l_3$. For all O: $d(P_1, O) + d(P_2, O) = d(P_1, P_2) = d_p$. Hence for all O: $(l_1 + l_2 - d_p)/2 \leq l_3$. If the distance d_p between parents P_1 and P_2 is unknown we can use bound (ii) to bound it: $d_p \leq max(l_1, l_2) \rightarrow d_p \leq l_2$. Hence for all O in the worst case we have: $l_1/2 \leq l_3$. In case $l_1 = l_2 = l$ we have for all O: $l/2 \leq l_3$.

Homologous crossover cannot produce empty offspring from non-empty parents. This can be shown by using the second inequality: $l_1/2 \leq l_3 \leq l_1/2 + l_2$. Independently from the distance between parents the minimum lower bound of the length of any offspring is half of the length of the shortest parent. When such parent is not empty ($l_1 \geq 1$) then $l_3 \geq 1/2$. Since the length is an integer we have $l_3 \geq 1$. So even for parents of length 1 the offspring are non-empty. □

Under geometric crossover, the more the parents differ, the more "unrelated" or "innovative" the offspring are. From the previous theorem, the size of the offspring is bounded by: $(l_1 + l_2 - d_p)/2 \leq l_3 \leq (l_1 + l_2 + d_p)/2$. Hence, the bigger the difference between the parents the bigger the range of offspring sizes. Note, however, that when using weighted edit distances (see next section) it is possible to create situations were an empty offspring can be returned.

8.3 Extensions of homologous crossover

8.3.1 Weighted edit distances and geometric crossover

Extending homologous crossover to the case of weighted edit distances is crucial to capture more realistic details of real biological sequences. Weighted edit distances allow to specify relative preferences in the alignment before recombination such as character mismatches vs. sequence interruptions (spaces), positional preferences (for example, matches at the extremities vs. matches at the centre of the sequences) or preferences on the mismatching pairs (for example, preferring a mismatch (a, t) to a mismatch (a, c)).

The following theorem is a very general and useful result that connects weighted edit moves for any solution representation and metric spaces.[1]

Theorem 8.3.1. *Any weighted edit distance with strictly positive weights on edit moves is a metric.*

Proof. A space of configurations endowed with an edit distance with strictly positive weights can be represented by a weighted graph in which nodes are syntactic configurations and weighted edges represent (reversible) weighted edit moves transforming one configuration into neighbour configuration. Any graph with strictly positive weights on edges is a metric space (see chapter 3). Hence, an edit distance with strictly positive weights on edit moves, that is isomorphic to such a graph, is a metric. □

The cost of a weighted alignment is the sum of the weights associated to each character-alignment pair. The weight of each pair of aligned characters is symmetric. Matching characters have weight 0. An optimal alignment is an alignment with minimal cost. The cost of the optimal weighted alignment between two sequences equals their weighted edit distance. The weighted edit distance is based on weighted edit moves whose weights are the same as those of their corresponding character-alignment pairs.

The following theorem extends the geometricity result of homologous crossover to weighted edit distances and weighted alignments.

Theorem 8.3.2. *Alignment-based homologous crossover on the optimal alignments under weighted edit distance d_w is a geometric crossover under d_w.*

[1]This is a fairly simple result. However, it appears that this has not been proved in published literature, leading to significant confusion, particularly in the bio-informatics literature, in which edit distances and scoring matrices are extensively used.

Proof. An optimal edit transcript T contains a set E of edit moves to transform u in v whose cost $w(E) = \sum_{e \in E} w_e$ is minimal. The weighted edit distance is $d_w(u, v) = w(E)$. The edit moves in E are independent because they can be applied in any order and transform u into v. Any intermediate sequence z obtained by applying a subset $E' \subseteq E$ of edit moves to u is on a shortest weighted path between u and v because $d_w(u, z) = w(E')$ and $d(z, v) = w(E \setminus E')$=w(E)-w(E') hence $d(u, z) + d(z, v) = d(u, v)$. A mask m selects a subset of edit moves $E_m \subseteq E$ from the transcript T to apply to u and produce the offspring z. Hence z is on the shortest path.

□

8.3.2 Gaps and geometric crossover

Gaps help create alignments that better conform to underlying biological models: the insertion or deletion of an entire substring often occurs as a single mutational event. Also, gaps allow to model loops in the alignments before recombination.

In this section we extend homologous crossover to the case of edit distance based on substitutions and block ins/del moves. This edit distance allows to specify preference for few big gaps against many small gaps in the alignment before recombination.

The block ins/del move is an operation that inserts or deletes a substring (a consecutive run of elements) of the sequence at once. The block ins/del move is weighted (with a strictly positive weight) according to the length of the substring involved in the operation: shorter substrings have smaller weight. However, each additional element in the substring contributes less to the weight of the move than the preceding elements. So, the weight of the block ins/del move to insert/delete the substring a is smaller than the sum of the weights of two or more consecutive block ins/del moves that do the same transformation (convex weights gap model).

Let us consider a modification of the aligned-based homologous crossover for the case of gaps in which (i) the optimal alignment is based on the convex weights gap model, and (ii) gaps cannot be broken while being exchanged between aligned parents (crossover points never happen within a gap in either parents).

Theorem 8.3.3. *The modified alignment-based homologous crossover for gaps with alignment based on the weighted edit distance d_{bw} is geometric crossover under d_{bw}.*

Proof. Let us consider a weighted block ins/del edit move such as its weights depends only on the length of the block in a way that shorter blocks have smaller cost per length unit: $l_1 < l_2 \rightarrow w(l_1)/l_1 > w(l_2)/l_2$. An optimal edit transcript must necessarily comprise the largest block ins/del edit move. Since the crossover mask treats each block ins/del edit move as a unity, the rest follows from theorem 8.3.2. □

8.4 Bridging natural and artificial evolution

From an engineering viewpoint living creatures are marvelously complex devices perfectly fit to their environment. How could evolution be able to design such complex devices in a relatively short time contrasted with the magnitude of the space of all possible designs? Current biological theories of evolution such as population genetics and molecular evolution are not able to explain this. Casting a computational perspective on biological evolution could be the key to understand why biological evolution is able to do “intelligent design” so effectively and efficiently without intelligence.

There are important differences between evolutionary algorithms and biological evolution. We mention three of them in the following.

A first important difference is that evolutionary algorithms have a pre-specified target fitness function to optimize, whereas biological evolution is an open-ended process without an objective function. The natural fitness landscape of a population is not static but changes over time in response to environmental changes and in response to the change in the population composition adapting to the new environment due to evolutionary forces. Evolution (adaptation) happens when the fitness landscape is non-flat [147].

A second important difference is that, whereas in evolutionary algorithms the genotype-phenotype map is fixed and provided by the designer of the algorithm, the biological genotype-phenotype map is itself subject to evolution and existing species acquired a genotype-phenotype map which allows improvements by mutation and selection [159].

A third important difference is that most evolutionary algorithms use rather simple, fixed-length genotypic representations together with mutation and crossover operators that are only a caricature of real biological genetic operators.

In the following we argue that homologous crossover, or some extension of it, is a plausible model of biological recombination and discuss the implication of this.

8.4.1 Is biological recombination geometric?

Most of pre-existing recombination operators for the most-frequently used representations are geometric. So, we could say that this geometric property unifies by abstraction across representations the notion of “crossoverness” emerged experimentally over the years. Evolutionary algorithms are only caricatures of biological evolution. Yet, they present a surprising design ability. This ability may have origin in the same algorithmic reason in both cases. Is biological recombination geometric as well? This question, if answer affirmatively, would allow us to study biological recombination within the same geometric framework.

Many details of real biological recombination are unknown and they are the focus of active research. There are various models for studying different aspects of biological evolution at different levels of granularity.

The model of homologous recombination based on fixed-size strings used in population genetics [36], is a simple extension of the traditional crossover for binary strings to the multi-valued case and it is geometric under Hamming distance.

Another, more realistic, model of recombination is studied in molecular evolution [75] and used also in bio-informatics [55]. In this model, DNA strands are modeled by (variable-length) sequences and they align on the basis of their contents before exchanging genetic material. Unequal crossover [2] happens when the homologous alignment of the strands is not perfect. The misalignment can be either the result of an error in the alignment due to environmental noise (this can be considered as a mutation), or it can be one of the possible best inexact alignments under edit distance at the level of genes. In this case unequal crossover would be geometric.

The reason why strands tend to align according to the edit distance can be understood at a molecular level. Our working hypothesis is that an edit distance, weighted and based on edit moves such as insertion/deletion (to model frame-shift), replacement (to model base mismatch), block-insertion/deletion (to model folds/loops), block-reversal (to model subsequence inversion) and block-transposition (to model subsequence transposition), is expressive enough to model the resulting configuration obtained at the equilibrium of all the forces that lead to the inexact homologous alignment of two chromosomes at a molecular level (before crossing over). The notion of minimum distance connects naturally with the notion of optimal alignment (best trade-off among all forces involved, or chemical equilibrium) of two macromolecules (chromosomes) that, as any other chemical reaction, tends to evolve toward the state of "minimum free energy". In summary:

1. the geometric crossovers associated with edit distances naturally capture the notion of homology, or inexact alignment based on the sequences contents.

2. there is a natural parallel between weighted edit distances and DNA pairing up at the molecular level because the weights on edit moves can be interpreted in chemical terms as attraction and repulsion forces.

3. there are a variety of edit distances that allow to show that other possible models of biological crossovers and many variants are still geometric. This suggests that assuming that biological recombination is geometric is reasonable even in the absence of full-knowledge about all its details.

8.4.2 Is the natural fitness landscape smooth?

According to the neutral theory of molecular evolution the majority of mutations are neutral and therefore have no effect on fitness[75]. So, genetic variation is simply the result of lots of neutral mutations unaffected by natural selection.

Although the neutral theory can explain the variation in the natural populations, some of its predictions are only partially confirmed by experimental evidence. This gave rise to the *nearly neutral model of molecular evolution*[116]. According to this theory, mutations in non-coding DNA are still strictly neutral. However, in coding DNA *mutations are regarded as being nearly neutral*, being either slightly deleterious or slightly advantageous.

According to the nearly neutral model of molecular evolution, the fitness landscape based on ins/del edit distance is therefore smooth in the sense that closer DNA strands tend to have closer fitness: DNA strands one mutation away have either identical fitness (neutral mutation) or very similar fitness (nearly neutral mutation); hence, the more mutations away, the more the fitness of the DNA strands can differ. So, despite the inherent dynamical character, the natural fitness landscape may be smooth, under edit distance, in the statistical sense introduced in chapter 3.

From another viewpoint, smoothness of the natural fitness landscape is the principle on which bio-informatics is firmly based upon: similarity of genotypes allows us to infer similarity in the phenotype (hence, in fitness) without doing any experimental work except searching databases of known genotypes by homology [55].

8.4.3 Geometric biological operator + smooth natural landscape = quick adaptation?

What is the fitness function optimised in natural evolution? Natural evolution, seen as a search algorithm, is trying to optimise the fitness function that is obtained from the fitness landscape by removing the space structure (by definition of fitness landscape, see chapter 3). While doing this optimisation, the fitness function is constantly changing, because the fitness landscape is constantly changing under the effect of population change due to evolution (optimisation) itself. The evolution (optimisation) ends when the fitness landscape becomes flat and the fitness function becomes constant. This means that the population is completely adapted to the environment. Hence, the performance of biological evolution, seen as a search algorithm, is in terms of speed of adaptation.

Now, we have all ingredients to put together a tentative argument that could explain the computational power of natural evolution, at least partially. The argument goes as follows. If one accepts that (i) biological recombination is geometric under edit distance and that (ii) the natural landscape is smooth and that (iii) smoothness of the landscape is the condition we need to enforce to the landscape to be well-searched by geometric crossover and geometric mutation, then the logical conclusion would be that biological recombination and mutation are well-matched with the natural fitness landscape. So, their performance in terms of adaptation would be expected to be much better than pure random search. This is to say that biological evolution would be efficient at doing adaptation.

8.5 Summary

In this chapter we have extended the geometric framework to the important case of sequences. We have given a number of theoretical results and started investigating the hypothesis that biological recombination is geometric and discussed its consequences.

Chapter 9

Product Crossover

In this chapter, we introduce the important notion of product geometric crossover that allows to build new geometric crossovers combining pre-existing geometric crossovers in a simple way. We then use product geometric crossover to design an evolutionary algorithm to solve the Sudoku puzzle.

9.1 Introduction

Theoretical results for metric spaces can naturally lead to interesting results for geometric crossover. In particular, in this chapter we focus on the notion of *metric transformation*. A metric transformation is an operator that constructs new metric spaces from pre-existing metric spaces: it takes one or more metric spaces as input and outputs a new metric space. The notion of metric transformation becomes extremely interesting when considered together with distances firmly rooted in the syntactic structure of the underlaying solution representation (e.g., edit distances). In these cases it gives rise to a simple and *natural interpretation in terms of syntactic transformations*.

In this chapter we extend the geometric framework introducing the important notion of Cartesian product of geometric crossovers, that allows to build new geometric crossovers combining preexisting geometric crossovers in a very simple way. The metric transformation considered is a simple product of metric spaces and the corresponding induced crossover transformation is the *product geometric crossover*. This may sound very abstract and impractical. However, it actually is not. Indeed, we put these ideas to the test and use product geometric crossover to design an evolutionary algorithm to solve the Sudoku puzzle. The different types of constraints make Sudoku an interesting study case for crossover design. We conducted extensive experimental testing and found that, on medium and hard problems, the new geometric crossovers perform significantly better than hill-climbers and mutations alone.

The chapter is organised as follows. In section 9.2, we extend the geometric framework with the notion of geometricity-preserving transformation and focus on the notion of product geometric crossover. In section 9.3, we discuss extensions of product geometric crossover to more complex structural compositions. In section 9.4, we design new geometric crossovers tailored to Sudoku using the product geometric crossover. In section 9.5, we test experimentally our new search operators.

9.2 Product geometric crossover

We first introduce the general notion of geometricity-preserving transformations. Then we consider a specific geometricity-preserving transformation associated with the product metric. We introduce product metrics for vector spaces, that are metric preserving transformations, and stress how they can be seen as natural generalization of simple metrics for vector spaces. We then introduce the notion of interval space that naturally bridges the metric and representation aspects of geometric crossover, and summarise some key results from interval spaces theory. We use these results to prove our main result of this chapter on the product of geometric crossovers. We then give a number of examples of applications. In section 9.3, we will discuss how to further generalize the product of geometric crossovers to a general structural composition of geometric crossovers.

9.2.1 Geometricity-preserving transformations

In previous chapters we have proven that a number of important pre-existing recombination operators for the most frequently used representations are geometric crossovers. We have also applied the abstract definition of geometric crossover to distances firmly rooted in a specific solution representation and designed brand-new crossovers. An appealing way to build new geometric crossovers is starting from recombination operators that are known to be geometric and derive new geometric crossovers by *geometricity-preserving transformations/combinations* that, when applied to geometric crossovers, return geometric crossovers.

The definition of geometric crossover is based on the notion of metric. Therefore, a natural starting point to seek geometricity-preserving transformations is to consider transformations of the underlying metrics that are known to return metric spaces and study how the geometric crossover associated to the transformed metric space relates with the geometric crossover associated with the original metric space.

There are a number of metric space transformations [30] [152] that are potentially of interest for geometric crossover: sub-metric spaces, product spaces, quotient metric space, gluing metric space, combinatorial transformation, non-negative combinations of metric spaces, Hausdorf transformation, concave transformation, and biotope transform.

Geometric crossover is well-defined once a metric space is defined. Let us consider the geometric crossover X associated to the original metric space M, and the geometric crossover X' associated to the transformed metric space $M' = mt(M)$ where mt is the metric transformation. The functional relationship among metric spaces and geometric crossovers can be nicely expressed through a commutative diagram (Figure 9.1). gx means application of the formal definition of geometric crossover and gt means *induced geometricity-preserving* crossover transformation associated to the metric transformation mt. This diagram becomes particularly interesting when the metric transformation mt is associated to an induced geometricity-preserving crossover transformation gt that has a simple interpretation in terms of syntactic manipulation. This indeed allows one to get new geometric crossovers starting from recombination operators that are known to be geometric by simple *geometricity-preserving syntax manipulation.*

We study those metric-preserving transformations whose associated geometricity-preserving transformations have a simple and natural interpretation on the solution representation.

$$\begin{tikzcd} M \arrow[r, "mt"] \arrow[d, "gx"] & M' \arrow[d, "gx"] \\ X \arrow[r, dashed, "gt"] & X' \end{tikzcd}$$

Figure 9.1: Commutative diagram linking metric and crossover transformations.

9.2.2 N-dimensional real spaces and product metric spaces

Metric spaces on $\mathbb{R}^2$. Let $S = \mathbb{R}^2$, and $x = (x', x'')$, $y = (y', y'')$. The following are metrics on S [152]:

$d_1(x, y) = |x' - y'| + |x'' - y''|$ (Manhattan distance)

$d_2(x, y) = \sqrt{|x' - y'|^2 + |x'' - y''|^2}$ (Euclidean distance)

$d_\infty(x, y) = \max\{|x' - y'|, |x'' - y''|\}$ (Chessboard distance)

These definitions may be extended to n-dimensional real spaces.

Product metric spaces. Given two metric spaces $M' = (S', d')$ and $M'' = (S'', d'')$, we may define several metrics on $S' \times S''$. For example, if $x = (x', x'')$ and $y = (y', y'')$ are in $S' \times S''$, let

$d_1(x, y) = d'(x', y') + d''(x'', y'')$ (Manhattan product)

$d_2(x, y) = \sqrt{d'(x', y')^2 + d''(x'', y'')^2}$ (Euclidean product)

$d_\infty(x, y) = \max\{d'(x', y'), d''(x'', y'')\}$ (Chessboard product)

These may be proved to be metrics [152]. These definitions may be extended to the product of any finite number of metric spaces.

It is interesting to notice that product spaces can be considered as generalization of n-dimensional real spaces, where the absolute value metric $|a - b|$ at each dimension (absolute values of the difference of paired coordinates) is replaced by a generic metric $d(a, b)$. So, for example the formula of the Manhattan product of two metric spaces d' and d'' can be obtained from the formula of the Manhattan distance between two bi-dimensional vectors by replacing the absolute values of the difference of the paired coordinates of the two vectors with the metric d' and d'' respectively. This is important because the generalization involves two different types of objects: a *simple metric* for a structured space and a structural *metric transformation* of generic metric spaces. More on this in the section 9.3.

9.2.3 Product interval spaces

Metric spaces can be associated to geometric interval spaces. The latter are a more natural setting for geometric crossover than the former. We review the notion of interval space and present results that draw a parallel between metric spaces and interval spaces. Then we use them to prove specific results for geometric crossover.

Interval space and geometric interval space

Let X be a set and let $I : X \times X \to 2^X$ be a function with the following properties:

- *Extensive Law*: $a, b \in I(a, b)$
- *Symmetry Law*: $I(a, b) = I(b, a)$

Then I is called an *interval operator on* X, and $I(a, b)$ is the *interval between* a *and* b. The resulting pair (X, I) is called an *interval space*.
An interval operator I on a set X is *geometric* provided the following hold:

- *Idempotent Law*: $\forall b \in X : I(b, b) = \{b\}$
- *Monotone Law*: if $a, b, c \in X$ and $c \in I(a, b)$, then $I(a, c) \subseteq I(a, b)$
- *Inversion Law*: if $a, b \in X$ and $c, d \in I(a, b)$, then $c \in I(a, d)$ implies $d \in I(c, b)$

A set with a geometric interval operator is called a *geometric interval space*.

Interval space associated to a metric space

The geodesic operator $[\bullet, \bullet]_d$ that associates extremes of a metric segment to all the points that constitute it is a geometric interval operator [154].

Definition 9.2.1. *Product segment.* Let us define the product segment as $[a, b]_{d\times d'} = \{(x_1, x_2)|x_1 \in [a_1, b_1]_d, x_2 \in [a_2, b_2]_{d'}\}$ where $a = (a_1, a_2), b = (b_1, b_2)$

Theorem 9.2.1. *Product segment theorem. The product segment corresponds to the segment of the Manhattan product space: $[(a_1, a_2), (b_1, b_2)]_{d\times d'} = [(a_1, a_2), (b_1, b_2)]_\delta$ where $\delta((a_1, a_2), (b_1, b_2)) = d(a_1, b_1) + d'(a_2, b_2)$ [154]*

This result may be extended to the product segment of any finite number of metric spaces.

Interval spaces connect very naturally with the notion of geometric crossover. There is a wealth of results for geometric interval spaces [154] that can easily be transferred to geometric crossover.

9.2.4 Product geometric crossover

A *product geometric crossover* of the geometric crossovers X_i based on the metric spaces (S_i, d_i) is a recombination operator defined over the Cartesian product set $\prod_i S_i$ that applies the geometric crossover X_i to the projection S_i.

For example, let us consider two geometric crossovers $X_1 : S_1 \times S_1 \to S_1$ and $X_2 : S_2 \times S_2 \to S_2$. A product geometric crossover of X_1 and X_2 is a recombination operator $X_3 : (S_1, S_2) \times (S_1, S_2) \to (S_1, S_2)$ that applies the geometric crossover X_1 to elements in the first position and crossover X_2 to elements in the second position.

From the results in the previous section we have the following

Theorem 9.2.2. *Any product geometric crossover is a geometric crossover under the distance given by the sum of the distances of the crossovers composing it.*

Proof. This follows immediately from the definition of geometric crossover and the product segment theorem. □

Corollary 9.2.3. *The geometric crossovers in each projection of the product geometric crossover need not to be independent for the product crossover to be geometric.*

Proof. This is because casting any form of dependency between geometric crossovers in different projections results in a reduction of the pool of offspring allowed to be created by the product geometric crossover. From the definition of geometric crossover, such a restriction does not affect its geometricity. □

Theorem 9.2.2 and corollary 9.2.3 are useful because they allow one to build new geometric crossovers combining crossovers that are known to be geometric. In particular, this applies to crossovers for mixed representations. Examples of application of product geometric crossovers include:

- Multi-crossover: product of the same crossover on the same representation n times.
- Hybrid crossover: product of different crossovers on the same representation for each projection.
- Hybrid representation crossover: product of different crossovers on different representations for each projection.
- Dependent crossover: different projections represent a single entity and they are mutually constrained. This occurs very often in real-world problems. E.g., for neural networks one projection might be a variable-size graph representing the structural part, while a second projection could be a variable-length sequence of reals representing the weights. Clearly recombination of the first projection imposes constraints on the recombination of the second projection to obtain a feasible offspring.

9.2.5 Geometric crossover for integer vectors

Integer vectors come in two types, those whose elements are ordinal attributes that can be thought as quantities, and those whose elements are cardinal attributes that can be thought as symbols without a natural ordering defined among them, such as, for example, the set $\{0, 1, 2, 3\}$ representing $\{North, East, South, West\}$.

These two types of vector are naturally associated with different notions of distance derived from distances that are meaningful for their elements. A meaningful distance for ordinal attributes is the following.

Definition 9.2.2. (Absolute value distance) Let A be a set of integers, the absolute value distance between $a, b \in A$ is the absolute value of their difference: $d(a, b) = |a - b|$. This is a metric because it is a special case of the Manhattan metric.

Instead, a meaningful distance for cardinal attributes is the following.

Definition 9.2.3. (Discrete metric space) Let A be any non-empty set and

$$d(x,y) = \begin{cases} 1, & x \neq y \\ 0, & x = y \end{cases} \quad \forall x, y \in A$$

This is a metric and is called the *discrete metric* of A.

Let us, first, consider the metric for cardinal vectors that derives from the discrete metric on its elements, and associated geometric crossover.

The discrete metric space is the path metric of a fully-connected graph with A as node set. The interval operator associated with the discrete metric space is: $\forall a, b \in A : [a, b] = \{a, b\}$ (all segments are edges).

Definition 9.2.4. (Discrete metric geometric crossover) We call the geometric crossover associated to the discrete metric, the *discrete metric geometric crossover* (DM-GX). Clearly, the only possible offspring of DM-GX are the parents.

Definition 9.2.5. (Hamming metric space) Let us consider a set A^n whose elements are all vectors of length n over some alphabet A of size $|A|$. The Hamming distance between two vectors is the number of coordinates where they differ. The Hamming space is denoted by $H(n, |A|)$.

Theorem 9.2.4. *The Hamming space can be seen as the product metric of n discrete metric spaces of the alphabet A is the Hamming space $H(n, |A|)$.*

Theorem 9.2.5. *Any traditional mask-based crossover for cardinal vectors taking values on the alphabet A is geometric under Hamming distance.*

Proof. This follows from the fact that any traditional mask-based crossover is the product crossover of n DM-GX, one for each projection. □

Let us now quickly consider the more intuitive case of ordinal vectors. It is easy to see that the geometric crossover associated with the absolute value distance produces integers in the interval between the parent integers. The product metric associated with the ordinal vectors is the Manhattan distance restricted to integers. The product geometric crossover is, in this case, a blend-type crossover: it is the box recombination for real-coded vectors restricted to integer vectors. For theorem 9.2.2, this operator is a geometric crossover under Manhattan distance restricted to integers.

9.3 Towards a structural composition of geometric crossovers

The previous results could be generalized in a very interesting way, extending the geometric framework to complex representations. In the following we discuss this.

Basic representations such as vectors, permutations, sequences, trees, graphs and sets, to mention only the most common, can all be seen as structures containing generic objects. These objects do not need to be necessarily numbers or atomic symbols from a given alphabet. Such objects can be structures themselves. So we can consider derived structures obtained by structural composition, for example sets of trees. The composition can be repeated recursively with different types of representations thus obtaining a wealth of derived representations, potentially suited to any problem conceivable.

Given geometric crossovers X_A and X_B for the structure A and B associated to the metric spaces M_A and M_B, what is the derived geometric crossover for some derived structure obtained by the structural composition of A and B? What is the derived metric space associated to the derived geometric crossover and the derived structure?

With the product geometric crossover, we have seen that when the structure is a vector, the structural composition with any other representations is connected to a natural derived geometric crossover consisting of a simple geometric crossover for each position in the vector, and associated to a derived metric that is simply the sum of the metric for each position.

In the case of vectors, we have a number of possible structural compositions (a number of metric product operators) but only one notion of metric product, the Manhattan product, that has a natural interpretation on the representation, making it the only one actually useful. In the case of other structures, there could be more than one (or even no) structural composition that has a natural interpretation on the representation (i.e., corresponding exactly to a simple syntactic manipulation). Furthermore, in the case of structures other than vectors, we do not have standard metric transformations such as the metric product that naturally suit them. So, where can we start our generalization from?

There seems to be a way suggested by the case of vectors: we have seen that simple metrics on the vector space (structured objects) can be easily generalized to structural metric transformations of generic metric spaces retaining the overall structure of the original object (vector of metric spaces). We could do the same to generalize metrics for other type of structures to structural metric transformations. The starting point is noticing that distances for structured representations are naturally expressed as some aggregating function of marginal contributions due to the difference in the structural subcomponents. The way of measuring the difference between two components is normally a very simple notion of metric, discrete metric for difference between symbols, or just absolute value for numeric components. In the case of vectors, the aggregating function is a simple sum, and the distance between components is the absolute value. So, the way to pass from metric distance to metric transformation is to replace the component metric with generic metrics, exactly how it was done for the case of vectors.

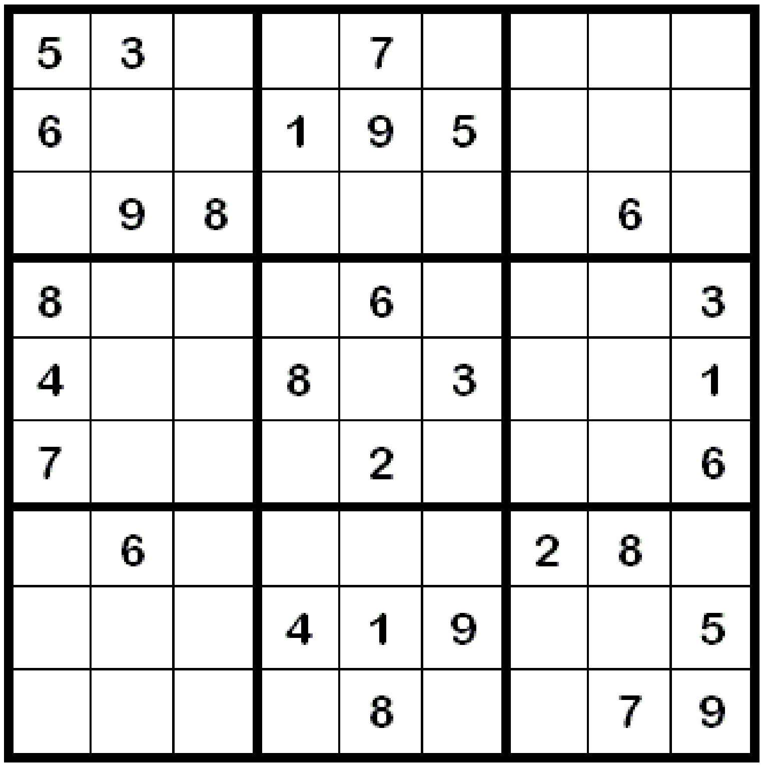
This seems to be a general and very promising starting point to extend simple metrics on any type of structured object to structural metric transformations naturally associated with its shape.

9.4 Geometric design for Sudoku

In this section, we use product geometric crossover to design an evolutionary algorithm to solve the Sudoku puzzle. The different types of constraints make Sudoku an interesting study case for crossover design. In section 9.5, we test experimentally our new search operators.

9.4.1 The Sudoku puzzle

Sudoku is a logic-based placement puzzle. The aim of the puzzle is to enter a digit from 1 through 9 in each cell of a 9×9 grid made up of 3×3 subgrids (called "regions"), starting with various digits given in some cells (the "givens"). Each row, column, and region must contain only one instance of each digit. In figure 9.2 we show an example of Sudoku puzzle. Sudoku puzzles with a unique solution are called proper sudoku. The majority of published grids are of



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 9.2: Example of Sudoku puzzle.

this type.

Published puzzles often are ranked in terms of difficulty. Perhaps surprisingly, the number of givens has limited bearing on a puzzle's difficulty which is based on the relevance and the positioning of the numbers rather than the quantity of the numbers.

The 9×9 Sudoku puzzle of any difficulty can be solved very quickly by a computer. The simplest way is to use some brute force trial-and-error search employing back-tracking. Constraint programming is a more efficient method that applies the constraints successively to narrow down the solution space until a solution is found or until alternate values cannot otherwise be excluded, in which case backtracking is applied. A highly efficient way of solving such constraint problems is the Dancing Links Algorithm, by Donald Knuth [76].

The general problem of solving Sudoku puzzles on $n^2 \times n^2$ boards of $n \times n$ blocks is known to be NP-complete [168]. This means that, unless P=NP, the exact solution methods that solve very quickly the 9×9 boards take exponential time in the board size in the worst case. However, it is unknown whether the general Sudoku problem restricted to puzzles with unique solutions remains NP-complete or becomes polynomial.

Solving Sudoku puzzles can be expressed as a graph coloring problem. The aim of the puzzle in its standard form is to construct a proper 9-coloring of a particular graph, given a partial 9-coloring.

A valid Sudoku solution grid is also a Latin square [19]. Sudoku imposes the additional regional constraint. Latin square completion is known to be NP-complete. A further relaxation of the problem allowing repetitions on columns (or rows) makes it polynomially solvable.

Admittedly evolutionary algorithms are not the best technique to solve Sudoku because they do not exploit systematically the problem constraints to narrow down the search. However, Sudoku is an interesting case study because it is a relatively simple but non trivial problem (since is NP-complete), and the different types of constraints make Sudoku an interesting playground for crossover design. In particular, its structure makes it a perfect candidate for product geometric crossover.

9.4.2 Sudoku constraints, search spaces and genetic operators

Sudoku is a constraint satisfaction problem with 4 types of constraints:

1. Fixed elements.
2. Rows are permutations.
3. Columns are permutations.
4. Boxes are permutations.

It can be cast as an optimization problem by choosing some of the constraints as hard constraints that all solutions have to respect, and the remaining constraints as soft constraints that can be only partially fulfilled and the level of fulfillment is the fitness of the solution. Notice that there is more than one way of deciding which are hard constraints and which are soft constraints. We have considered two search spaces: the first respecting one constraint, the other respecting two constraints at the same time. Trying to further restrict the search space, i.e., requiring to respect three constraints at the same time, seems unfeasible: even only generating a random solution within the space of grids respecting the first three constraints is a NP-complete problem being equivalent to finding a solution to a Latin square.

Restricted Hamming space

The first space we consider is the restricted hamming space, which presents the following characteristics:

- *Hard constraints*: fixed positions.
- *Soft constraints*: permutations on rows, columns and boxes.
- *Distance*: Hamming distance between grids.
- *Feasible geometric mutation*: change any non-fixed elements.
- *Feasible geometric crossover*: traditional crossover over the vector obtained by joining the rows of the grid.

It is easy to see that these mutation and crossover preserve fixed positions from parent grids to offspring grids. The mutation is geometric under Hamming distance (the offspring is in the Hamming ball of radius 1 centered in the parent). The crossover, being the traditional crossover, is geometric under Hamming distance.

To restrict the search to the space of grids with fixed positions, the initial population must be seeded with feasible random solutions taken from the feasible space, which is, by solutions respecting the fixed position constraints.

Row-swap space

The second space we consider is the row-swap space, which presents the following characteristics:

- *Hard constraints*: fixed positions and permutations on rows.

- *Soft constraints*: permutations on columns and boxes.
- *Distance*: sum of swap distances between paired rows (row-swap distance).
- *Feasible geometric mutation*: swap two non-fixed elements in a row.
- *Feasible geometric crossover*: row-wise PMX and row-wise cycle crossover.

This mutation preserves both fixed positions and permutations on rows because swapping elements within a row that is a permutation returns a permutation. The mutation is geometric under row-swap distance (the offspring is in the row-swap ball of radius 1 centered in the parent).

Row-wise PMX and row-wise cycle crossover recombine parent grids applying respectively PMX and cycle crossover to each pair of corresponding rows. In case of PMX the crossover points can be selected to be the same for all rows, or random for each row. In terms of offspring that can be generated, the second version of row-wise PMX includes all the offspring of the first version.

Simple PMX and simple cycle crossover applied to parent permutations return always permutations. They also preserve fixed positions. This is because both are geometric under swap distance and in order to generate offspring on a minimal sorting path between parents using swaps (sorting one parent into the order of the other parent) they have to avoid swaps that change common elements in both parents (elements that are already sorted). Therefore, also row-wise PMX and row-wise cycle crossover preserve both hard constraints.

Using the product geometric crossover theorem (section 9.2.4), it is immediate to see that both row-wise PMX and row-wise cycle crossover are geometric under row-swap distance, since simple PMX and simple cycle crossover are geometric under swap distance. Since simple cycle crossover is also geometric under Hamming distance (restricted to permutations), row-wise cycle crossover is also geometric under Hamming distance.

To restrict the search to the space of grids with fixed positions and permutations on rows, the initial population must be seeded with feasible random solutions taken from this space. Generating such solutions can be done still very efficiently.

The row-swap space presents two advantages to the restricted Hamming space: (i) the search space of feasible solutions is much smaller and (ii) this restriction includes the optimum grid and prunes only grids with lower fitness.

9.4.3 Sudoku fitness landscapes

In chapter 3, we have seen that, as a rule-of-thumb, search operators associated with a smooth fitness landscape are likely to perform well. In the following, we will show that the fitness landscapes associated with both spaces introduced in the previous section are smooth, making the search operators associated with them a good choice for Sudoku. As a measure of smoothness, we calculate the normalized Lipschitz constant (chapter 3) for the two landscapes.

We use as fitness function (to maximize) the sum of number of unique elements in each row, plus the sum of number of unique elements in each column, plus the sum of number of unique elements in each box. So, for a 9×9 grid we have a maximum fitness of $9 \cdot 9 + 9 \cdot 9 + 9 \cdot 9 = 243$ for a completely correct Sudoku grid and a minimum fitness little more than $9 \cdot 1 + 9 \cdot 1 + 9 \cdot 1 = 27$ because for each row, column and square there is at least one unique element type. The maximum global fitness variation is, therefore, little less than $\Delta F = 216$.

Under Hamming distance, a single element change affects the current fitness of -1, 0 or +1 for the row, for the column and for the box the element belongs to. So, the total change in fitness due to a single element change is between -3 and +3. Therefore, the maximum local fitness variation between two neighboring solutions in the Hamming space is $\Delta f_H = 3$.

For the row-swap space, since the rows are always permutations, the minimum fitness is $9 \cdot 9 + 9 \cdot 3 + 9 \cdot 1 = 117$, which gives a global variation of 126. A single swap in a row affects the current fitness of 0 for the row (permutation is preserved), in the range between -2 and +2 for the columns touched (a single swap can fix or unfix at most one element in two columns at the same time) and between -2 and +2 for the boxes touched. So, the total change in fitness due to a single swap in a row is between -4 and 4. Therefore, the maximum local fitness variation between two neighboring solutions in the row-swap space is $\Delta f_{rs} = 4$.

As seen in chapter 3, the normalized Lipschitz constant of the landscape is maximum local fitness variation over maximum global fitness variation. The smaller this measure, the smoother the landscape. The Lipschitz constants of the Sudoku fitness landscapes with Hamming distance and row-swap distance are $S_H = 3/216 = 0.0139$ and $S_{rs} = 4/126 = 0.0317$, respectively. Therefore, *the fitness landscapes are both very smooth* when compared with the two extreme of smoothness 0 (no change between neighbor solutions) and 1 (maximum and minimum are neighbors).

Interestingly, the landscape based on the Hamming space is a little smoother than the one based on the row-swap space. So, in terms of smoothness the Hamming space is a better choice than the row-swap space.

It is also interesting to notice that the maximum global fitness variation increases (linearly) with the grid size, whereas the the maximum local fitness variation is constant with the grid size. So, this may counteract the increasing difficulty of the search due to a larger space.

9.5 Experiments

9.5.1 Genetic framework

We tested the various operators with a steady state evolutionary algorithm. In the following, we describe the framework used in our experiments. The parameters were subject to extensive systematic tuning.

- *Encoding*: Each individual was encoded as an array of 81 integers in the range [0..9], which was interpreted as a vector of 9 rows.
- *Initialization*: For the Hamming space experiments, each candidate solution of the first generation had the contents all non-fixed positions set to random integers in the range [1..9]. For the swap space experiments, each row of each candidate solution in the first generation was a random valid permutation of all numbers in the range [1..9], respecting fixed positions. See section 9.4 for more information.
- *Population*: The population size was set to 5,000.

- *Selection and replacement*: Each generation, all candidate solutions were evaluated using the fitness function described in section 9.4. The whole population was then sorted based on fitness, so that the 2,500 best candidates formed the new elite. The other half of the population was replaced with 2,500 new candidates, created by crossover and mutation (using the current operators) from two candidates randomly selected from the elite.

- *Crossover*: We used the crossover operators described in section 9.4. All crossovers operate within the bounds of single rows (e.g., PMX is applied row-wise), except two-point crossover, which is traditional two-point crossover applied to the whole grid seen as a vector, and the whole-row crossover, which randomly selects whole rows from either of the two parents, and is geometric under both distances. Cycle crossover takes two forms: onecycle, the form proposed by Goldberg [51], which considers only one cycle, and multicycle, which considers all possible cycles. Uniform swap is a sorting crossover based on edit-swap-move.

- *Mutation*: For each newly formed candidate in a population, the current mutation operator was applied with probability 0.8. A number of mutation operators were used. For the Hamming space, we used point mutation, meaning that one non-fixed position was set to a new random value in the range [1..9] (Table 9.2); uniform swap mutation, where the values of two randomly selected positions of the whole grid are exchanged (Table 9.3); and exponential probability point mutation, after each point mutation there is a 0.8 probability of a new point mutation, leading to a theoretically infinite number of mutations, but in practice only a handful (Table 9.4). We also tried the algorithm without any mutation at all (Table 9.1).

  For the swap space, we used the row swap mutator, where two non-fixed positions in a row were exchanged (Table 9.6). We also used an exponential probability row swap mutator (Table 9.7), and tried the various crossover operators without any mutation (Table 9.5).

- *Stopping criterion*: The stopping criterion was that no progress had been made (no fitter candidates discovered) for 20 generations.

We further tested all the mutation operators as local search operators. In other words, we used these operators as bases for hill-climbers. Initialization and fitness function for the hill-climbers were the same as for the population-based algorithms, and the stopping criterion was that no progress had been made for 100,000 fitness evaluations.

9.5.2 Test environment

Five initial Sudoku grids were taken from the sudoku problem generator at *www.sudoku.com*. Three of them were classified (by the generator program) as easy, one as medium hard and one as hard. We then ran 30 evolutionary runs of each combination of crossover and mutation operator on each initial grid. As time efficiency was not a main concern, we did not measure the time taken for each run.

9.5.3 Results

The result tables are organized by mutation type.

Hamming space

In Tables 9.1, 9.2, 9.3 and 9.4, we report the results for mutations and crossovers associated with the Hamming space. The best crossover, independent of mutation operator, is whole-row crossover, followed by two-point crossover. But even with row-wise crossover, we only relatively rarely manage to reach the optimal solution. We have never evolved an optimal solution without crossover, or when using uniform crossover.

Swap space

In Tables 9.5, 9.6 and 9.7, we report the results for mutations and crossovers associated with the Swap space. In general, the final fitnesses are a lot higher for the swap space experiments than for the Hamming space experiments. For the three easy instances, all the crossovers generally perform very well, when supplied with some mutation. PMX, uniform swap and multicycle crossover perform surprisingly well even in the absence of mutation. For the medium difficulty problem and for the hard problem either uniform swap or multicycle crossover seems to perform best.

Hill-climbers

In table 9.8, we report the results for the hill-climbers. Two mutations that performed reasonably well were uniform swap and exponential probability row swap mutation. No mutations managed to get even close to optimal fitness for the medium and hard problem instances.

9.5.4 Discussion

We have seen in section 9.4.2, that the row-swap space presents two advantages to the restricted Hamming space: (i) the search space of feasible solutions is much smaller and (ii) this restriction includes the optimum grid and prunes only grids with lower fitness. In section 9.4.3, we have seen that both spaces are associated with smooth fitness landscapes, with the landscape based on the Hamming space being a little smoother than the row-swap space. So, we expected geometric operators based on both spaces being good operators for the Sudoku problem (because both are associated with smooth landscapes), with those based on row-swap space being better (because it is a much smaller search space).

The experiments seem to corroborate the theoretical analysis. The evolutionary algorithms based on both spaces find easily near-optimal solutions. Furthermore, the search operators associated with the row-space are better than those associated with the Hamming space, and some of them find consistently optimal solutions[1] for easy grids, and more than half of the times optimal solutions for medium and hard grids.

Interestingly, for both spaces, the evolutionary algorithms with only crossover performed better than the corresponding evolutionary algorithms with only mutation, and even better than the corresponding hill-climbers. So, it seems that the fitness landscapes proposed are particularly well-suited to crossover.

[1]To the end of solving the Sudoku problem, sub-optimal solutions are useless because they correspond to invalid grids.

Table 9.1: No mutation, hamming space crossovers. Each cell contains the number of evolutionary runs out of 30 that produced the optimal solution, and the mean fitness of the best candidate of the last generation of all 30 runs (the max fitness is 243).

Grid	None		Uniform		Twopoint		Whole-row	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	211	0	212	0	240	1	239
Easy 2	0	212	0	212	1	239	7	241
Easy 3	0	215	0	214	4	239	1	239
Med	0	210	0	210	0	236	1	238
Hard	0	210	0	209	0	237	0	239

Table 9.2: Point mutation, hamming space crossovers.

Grid	None		Uniform		Twopoint		Whole-row	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	231	0	212	4	240	1	240
Easy 2	0	231	0	212	2	239	8	241
Easy 3	0	232	0	213	5	241	9	241
Med	0	230	0	210	0	238	1	238
Hard	0	230	0	210	0	237	1	239

9.6 Summary

In this chapter we have extended the geometric framework introducing the notion of product crossover. This is a very general result that allows one to build new geometric crossovers customized to problems with mixed representations by combining pre-existing geometric crossovers in a straightforward way. We have presented this notion in the more general setting of metric transformations. Using the product geometric crossover theorem, we have also shown that traditional crossovers for symbolic vectors and blend crossovers for integer and real vectors are geometric crossover.

We have then designed new geometric crossovers for the Sudoku puzzle that deal in a natural way with its constraints. We have demonstrated the usage of the important notion of product geometric crossover to straightforwardly derive (i) new geometric crossovers for the entire grid obtained by employing simple geometric crossovers for each row and (ii) the distance functions

Table 9.3: Uniform swap mutation, hamming space crossovers.

Grid	None		Uniform		Twopoint		Whole-row	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	231	0	212	2	240	5	241
Easy 2	0	231	0	212	4	238	8	241
Easy 3	0	231	0	213	4	240	14	242
Med	0	228	0	211	0	235	0	238
Hard	0	229	0	210	0	236	0	238

Table 9.4: Exponentional probability point mutation, Hamming space crossovers.

Grid	None		Uniform		Twopoint		Whole-row	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	222	0	212	1	238	2	240
Easy 2	0	221	0	212	1	237	8	241
Easy 3	0	221	0	214	3	238	3	240
Med	0	223	0	210	0	234	0	237
Hard	0	220	0	210	0	234	0	238

associated with them. This has allowed us to analyze the geometric fitness landscape associated to the new geometric crossovers and tell *a priori*, by the way the fitness landscape is constructed, that the new crossovers are very well-suited to the Sudoku puzzle hence likely to perform well. We extensively tested a number of geometric crossovers, mutations and hill-climbers and found that the operators associated with the row-swap distance are the best and produce consistently (near) optimal Sudoku grids.

Table 9.5: No mutation, swap space crossovers.

Grid	None		Whole-row		PMX		Uniform swap		Onecycle		Multicycle	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	212	0	239	26	243	28	243	0	237	24	243
Easy 2	0	212	2	241	21	242	21	242	0	234	22	242
Easy 3	0	214	1	240	29	243	30	243	0	237	30	243
Med	0	210	0	238	12	241	19	242	0	233	11	241
Hard	0	210	0	238	9	242	15	242	0	234	12	242

Table 9.6: Row swap mutation, swap space crossovers.

Grid	None		Whole-row		PMX		Uniform swap		Onecycle		Multicycle	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	5	241	15	242	28	243	29	243	22	242	27	243
Easy 2	3	240	16	242	24	243	27	243	9	241	23	243
Easy 3	8	241	26	243	30	243	30	243	25	243	30	243
Med	0	239	3	241	8	240	10	237	1	240	7	241
Hard	0	239	5	241	4	241	9	239	1	240	10	242

Table 9.7: Exponential probability row swap mutation, swap space crossovers.

Grid	None		Whole-row		PMX		Uniform swap		Onecycle		Multicycle	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	239	14	242	27	243	24	242	20	242	29	243
Easy 2	2	239	16	242	19	242	21	241	11	242	23	243
Easy 3	5	240	27	243	29	243	30	243	29	243	29	243
Med	0	237	4	241	0	235	3	230	0	239	7	240
Hard	0	237	2	241	0	236	1	234	2	239	7	241

Table 9.8: Hill-climbers. Mutation operators along the y-axis.

Grid	None		Point		Uniform swap		Exp. p. point		Row swap		Exp. p. row swap	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	190	0	232	12	241	0	233	1	238	7	241
Easy 2	0	190	0	232	9	241	0	235	0	237	2	239
Easy 3	0	188	0	232	21	242	0	233	6	239	15	241
Med	0	182	0	232	1	239	0	232	0	239	0	239
Hard	0	171	0	232	1	240	0	233	0	238	0	239

Chapter 10

Quotient Crossover

In chapter 9 we have started studying how metric transformations of the distance associated with geometric crossover affect the original geometric crossover. In particular, we focused on the product of metric spaces. This metric transformation gives rise to the notion of product geometric crossover that allows to build one new geometric crossovers combining pre-existing geometric crossovers in a simple way.

In this chapter, we study another metric transformation, the quotient metric space, that gives rise to the notion of *quotient geometric crossover*. This turns out to be a very versatile notion. We give many examples of application of the quotient geometric crossover.

10.1 Introduction

The metric transformation associated with *quotient geometric crossover* is the quotient metric space. Quotient geometric crossover reduces the search space actually searched by geometric crossover, introduces problem knowledge in the search by using a distance better tailored to the specific solution interpretation and it can be used to remove the inherent search bias of the original crossover operator. The notion of quotient geometric crossover is extremely versatile.

The chapter is organized as follows. In Section 10.2, we introduce the notion of quotient geometric crossover. In the following sections, we study several useful applications related with quotient geometric crossover. In Section 10.3, we show how work on a grouping problem presented in chapter 11 can be understood in terms of quotient geometric crossover. Here, quotient geometric crossover is used to filter out inherent redundancy in the solution representation. In section 10.4, we show that the same technique can be generalised form groupings to graphs. In Section 10.5, we apply the quotient geometric crossover in a completely different way: we show how homologous crossover for variable-length sequences can be understood as a quotient geometric crossover. In Section 10.6, we recast the results on geometric crossover on glued real-valued representations presented in chapter 4 in terms of quotient geometric crossover. In Section 10.7, we consider functional representation and shows how the concept of quotient geometric crossover is connected to the search of spaces of functions. Genetic programming is shown as an example. In Section 10.8, we explain that quotient geometric crossover can be used to understand how crossover and neutral code interact.

10.2 Quotient geometric crossover

10.2.1 Definition of quotient geometric crossover

In the following, we recall the definition of quotient metric space given in chapter 4, then we give the definition of quotient geometric crossover. Let (S, d) be a metric space and $\sim$ be an equivalence relation on S. Consider the *quotient set* $S/\sim$, which is the set consisting of all equivalence classes of $\sim$. The metric on $S/\sim$ induced by the original metric d on S is: $d_\sim(\bar{x}, \bar{y}) := \inf_{x \in \bar{x}, y \in \bar{y}} d(x, y)$ for $\bar{x}, \bar{y} \in S/\sim$. This metric space $(S/\sim, d_\sim)$ is called *quotient metric space.*

In a metric space (S, d) a *quotient line segment* is the set of the form $[x; y]_{d_\sim} = \{z \in S \mid d_\sim(\bar{x}, \bar{z}) + d_\sim(\bar{z}, \bar{y}) = d_\sim(\bar{x}, \bar{y}), \bar{z} \in S/\sim\}$ where $\bar{x}, \bar{y} \in S/\sim$. Now we can define quotient geometric crossover.

Definition 10.2.1 (Quotient geometric crossover).
A binary operator GX_q is a *quotient geometric crossover* under the metric d and the equivalence relation $\sim$ if all offspring are in the quotient line segment between its parents x and y, i.e., $GX_q(x, y) \subseteq [x; y]_{d_\sim}$.

10.2.2 Genotype-phenotype map

The notion of quotient geometric crossover is important because it lies at the heart of the relation between geometric crossover and genotype-phenotype map as we illustrate in the following.

The term "Genotype" means solution representation, i.e., some structure that can be stored in a computer and manipulated. "Phenotype" means solution itself without any reference to how it is represented. Sometimes it is possible to have a one-to-one mapping between genotypes and phenotypes, so the distinction between genotype and phenotype becomes purely formal. However, in many interesting cases phenotypes cannot be represented by unique genotypes. So, the same phenotype is represented by more than one genotype. In this case, we say that we have a *redundant representation*. For example, to represent a graph we need to label its nodes and then we can represent it using its adjacency matrix. This representation is redundant: the same graph can be represented with more than one adjacency matrix by relabeling its nodes.

There are quite a few problems in which it is hard to represent one phenotype by just one genotype using traditional representations. Roughly speaking, redundant representations for many combinatorial optimization problems lead to severe loss of search power in genetic algorithms, in particular, with respect to traditional crossovers [16]. To alleviate the problems caused by redundant representation, a number of methods have been proposed [32] [80] [112] [155]. Among them, a technique called *normalization*[1] is a particularly interesting representative. It transforms the genotype of a parent to another genotype to be consistent with the other parent so that the genotype contexts of the parents are as similar as possible in crossover. There have been a number of successful studies using normalization. An extensive survey about normalization is in [16].

Although many studies about normalization did not use the concept of distance, once a distance d_G on the genotypes G is defined, we can formally redefine the result of the normalization of the second parent p_2 to the first p_1 as:

$$p_2' := \operatorname*{argmin}_{s \in w(p_2)} d_G(p_1, s),$$

[1]The term of *normalization* is firstly appeared in [66]. However, it is based on the adaptive crossovers proposed in [80] [112].

where $w(s)$ is the set of all the genotypes with the same phenotype as genotype s. The use of distance to define normalization is important because it generalizes and makes rigorous the notion of normalization for *any solution representation*.

Now we formally present the general relation between geometric crossover and genotype-phenotype map. The concept of normalization defined by distance is closely related with the quotient geometric crossover. Let us consider genotype-phenotype maps $g : G \rightarrow P$ that are non injective (redundant representation). A map g induces a natural equivalence relation $\sim$ on the set of genotypes: *genotypes with the same phenotype belong to the same class*. Since there is a one-to-one map between the classes in $G/\sim$ and the set of phenotypes P, we can think of phenotypes in P as coinciding with the classes of genotypes in $G/\sim$. Given a distance d_G on the space of genotypes G, the quotient with the relation $\sim$ produces a distance d_P on the phenotypes P. So, $P = G/\sim$ and $d_P(\bar{x}, \bar{y}) = \inf_{x \in \bar{x}, y \in \bar{y}} d_G(x, y)$.

By applying the formal definition of geometric crossover to the metric spaces (G, d_G) and (P, d_P), we obtain the geometric crossovers X_G and X_P, respectively. X_G searches the space of genotypes and X_P searches the space of phenotypes. Searching the space of phenotypes has a number of advantages: (i) X_P is smaller than the space of genotypes, hence quicker to search; (ii) the phenotypic distance is better tailored to the underlying problem, hence the corresponding geometric crossover is likely to work better; (iii) the space of phenotypes has different geometric characteristics from the genotypic space which can be used to remove unwanted bias from geometric crossover.

However, the crossover X_P cannot be directly used itself because it recombines phenotypes which are formal objects that can be represented and manipulated only via their genotypes. The quotient geometric crossover allows us to search the space of phenotypes with the crossover X_P *indirectly* by manipulating the genotypes G. This is possible because for the commutative diagram in Figure 9.1 (chapter 9) there exists an induced geometricity-preserving transformation gt of the genotypic crossover X_G that allows us to use the genotypic representation to implement a geometric crossover in the space of phenotypes $gt(X_G)$, without making explicit use of phenotypes. The type of the transformation gt depends on the type of the equivalence relation $\sim$ used in the quotient of the underlying metric space that, in turns, depends on the underlying syntax of the solution representation. It may happen that the induced geometricity-preserving transformation may be difficult to implement and/or computationally intractable. In these cases, it may not be feasible to manipulate genotypes to induce an *exact* equivalent of the phenotypic geometric crossover. So, a genotypic transformation which approximates the induced phenotypic geometric crossover may be preferable and still retaining most of the advantages of the exact equivalent.

In the following sections we consider a number of equivalence classes for the quotient operation and their related induced genotypic crossover transformations.

10.3 Groupings

Now we introduce an example class of problems using the normalization method. The problem that we consider is the *grouping* problem [37]. Grouping problems are commonly concerned with partitioning given item set into mutually disjoint subsets. Examples belonging to this class of problems are multiway graph partitioning, graph coloring, bin packing, and so on. Grouping representations are also used to solve the joint replenishment problem, which is a well-known problem in the field of industrial engineering [118]. In this class of problems, the normalization decreased the problem difficulty and led to notable improvement in performance.

Most normalization studies for grouping problems were focused on the k-way partitioning problem. In this problem, the k-ary representation, in which k subsets are represented by the integers from 0 to $k-1$, has been generally used. In this case, a phenotype (a k-way partition) is represented by $k!$ different genotypes, because there are $k!$ ways of labelling the same (unordered) partition. So, genotypes are ordered partitions represented as permutations with repetitions and phenotypes are unordered partitions. In the problem, a normalization method was used in [66]. Other studies for the k-way partitioning problem used the same technique [16] [73]. Normalization pursues the minimization of genotypic differences among aligned chromosomes before recombination by relabeling the subsets in the partition. In chapter 11, we see that geometric crossover based on a specific distance measure for unordered partitions, the *labeling-independent distance*, is a recombination that requires a normalization phase before recombination. In the following we briefly illustrate how this recombination can be understood as a quotient geometric crossover.

Given two k-ary encodings $\mathfrak{a}, \mathfrak{b} \in U = \{1, 2, \ldots, k\}^n$ (fixed-length vectors on a k-ary alphabet) and the Hamming distance H in U, we define the *labeling-independent distance* LI associated to H as follows:

$$LI(\mathfrak{a}, \mathfrak{b}) := \min_{\sigma, \sigma' \in \Sigma_k} H(\mathfrak{a}_\sigma, \mathfrak{b}_{\sigma'})$$

where Σ_k is the set of all permutations of length k and $\mathfrak{a}_\sigma$ is a permuted encoding of $\mathfrak{a}$ by a permutation σ, i.e., the i^{th} element a_i of $\mathfrak{a}$ is transformed into $\sigma(a_i)$. Then, labeling-independent distance LI is a *pseudo-metric*[2] on U (see chapter 11).

Given an element $\mathfrak{a} \in U$, since H is a metric, there are $k!$ elements such that the labeling-independent distance LI to $\mathfrak{a}$ is zero. If the labeling-independent distance LI between two elements is equal to zero, we define them to be *in relation* $\sim$. Then, the relation $\sim$ is an equivalence relation (see chapter 11). $(U/\sim, LI)$ is a metric space, i.e., the labeling-independent distance[3] LI is a metric on $U/\sim$ (see chapter 11).

We can design a quotient geometric crossover based on the labeling-independent metric.

Definition 10.3.1 (Labeling-independent crossover). Normalize the second parent to the first under the Hamming distance H. Do the normal crossover using the first parent and the normalized second parent.

Theorem 10.3.1. *The labeling-independent crossover is geometric under the metric LI (see chapter 11).*

In the following we summarise, for the specific case of groupings, what the elements of the commutative diagram of the quotient geometric crossover are.

[2]As explained in chapter 3, the term *distance* or *metric* denotes any real valued function that conforms to the axioms of identity, symmetry and triangular inequality. A distance for which the axiom of identity is relaxed so that distance zero does not necessarily implies equality (but equality still implies distance zero) is called *pseudo-metric*.

[3]Strictly speaking, this labeling-independent distance is not exactly the same as LI on U because it is a metric on $U/\sim$. However, the definition of LI on $U/\sim$ can be naturally induced from LI on U, i.e., $LI(\bar{\mathfrak{a}}, \bar{\mathfrak{b}}) := \min_{\sigma,\sigma' \in \Sigma_k} H(\mathfrak{a}_\sigma, \mathfrak{b}_{\sigma'})$.

- *Genotypes G*: labeled partitions represented as vectors of symbols
- *Phenotypes P*: unlabeled partitions
- *Equivalence relation $\sim$*: labeled partitions with the same partition structure
- *Distance on genotypes d_G*: Hamming distance
- *Distance on phenotypes d_P*: labeling-independent distance
- *Crossover on genotypes X_G*: traditional crossover for vectors
- *Crossover of phenotypes X_P*: label normalization before traditional crossover
- *Induced crossover transformation gt*: label normalization

The benefit of understanding normalization for grouping problems in terms of quotient geometric crossover is the possibility of understanding the benefit of normalization in terms of landscape analysis (see chapter 11).

10.4 Graphs

In this section, we consider any problem naturally defined over a graph in which the fitness of the solution does not depend on the labels on the nodes but only on the structural relationship, i.e., edge between nodes.

Formally, let $A \in \mathfrak{M}_n$ be the adjacency matrix of a labeled graph using labels of n nodes and let P be an $n \times n$ permutation matrix[4]. Then the matrix PA means the labeled graph obtained by relabeling A according to the permutation represented by P. The fitness $f : \mathfrak{M}_n \to \mathbb{R}$ satisfies that for every $A \in \mathfrak{M}_n$ and every permutation matrix P, $f(A) = f(PA)$.

Let $(\mathfrak{M}_n, H)$ be a metric space on the labeled graphs under the Hamming distance H. Notice that this metric is labeling-dependent. In particular, $H(A, PA)$ may not be zero although A and PA represent the same structure. If A is equal to PA' for some permutation matrix P, we define A and A' to be *in relation* $\sim$, i.e., $A \sim A'$. Then, the relation $\sim$ is an equivalence relation.

An *unlabeled graph* $\mathfrak{g}$ is the equivalence class of all its labeled graphs, i.e., $\mathfrak{g}(A) =$

[4]Permutation matrix is a $(0, 1)$-matrix with exactly one 1 in every row and column.

$\{PA \mid P \text{ is a permutation matrix}\}$. The *unlabeled-graph space* $\mathfrak{M}_n/\sim$ is the set of all equivalence classes partitioning the set $\mathfrak{M}_n$.

We define an *induced distance measure* LI on $M_n/\sim$ as follows. For each $\mathfrak{g}, \mathfrak{g}' \in \mathfrak{M}_n/\sim$,

$$LI(\mathfrak{g}, \mathfrak{g}') := \min_{A \in \mathfrak{g}, A' \in \mathfrak{g}'} H(A, A').$$

Then, $(\mathfrak{M}_n/\sim, LI)$ is a metric space, i.e., LI is a metric on $\mathfrak{M}_n/\sim$. This shows that the metric space $(\mathfrak{M}_n, H)$ induces a quotient metric space $(\mathfrak{M}_n/\sim, LI)$.

Definition 10.4.1 (Labeling-independent crossover). Do the graph matching of the second parent p_2 to the first p_1 under the Hamming distance H obtaining the graph-matched second parent

$$p_2' := \operatorname*{argmin}_{A \in \mathfrak{g}(p_2)} H(p_1, A).$$

Do the normal crossover using the first parent p_1 and p_2'.

The following theorem shows that the labeled-graph geometric crossover for $(\mathfrak{M}_n, H)$ induces the unlabeled-graph geometric crossover for $(\mathfrak{M}_n/\sim, LI)$.

Theorem 10.4.1. *The labeling-independent crossover is geometric under the metric LI.*

Proof. Let p_1 and p_2 be parents and c be offspring after the labeled-structure crossover GX. It is enough to show that $d_\sim(w(p_1), w(p_2)) = d_\sim(w(p_1), w(c)) + d_\sim(w(c), w(p_2))$. Since $d_\sim$ is metric, by triangle inequality, it is trivial to see that $d_\sim(w(p_1), w(p_2)) \le d_\sim(w(p_1), w(c)) + d_\sim(w(c), w(p_2))$. Now, we will show that $d_\sim(w(p_1), w(p_2)) \ge d_\sim(w(p_1), w(c)) + d_\sim(w(c), w(p_2))$.

Let $\sigma' = \arg\min_{\sigma \in \Sigma} d(p_1, r(p_2, \sigma))$. Then, we let $p_2' = r(p_2, \sigma')$.

$$\begin{aligned} d_\sim(w(p_1), w(p_2)) &= d(p_1, p_2') \\ &= d(p_1, c) + d(c, p_2') \\ &\qquad (\because GX \text{ is geometric under } d) \\ &\ge d_\sim(w(p_1), w(c)) + d_\sim(w(c), w(p_2')) \\ &= d_\sim(w(p_1), w(c)) + d_\sim(w(c), w(p_2)). \end{aligned}$$

□

The labeling-independent crossover is defined over unlabeled graphs $\mathfrak{M}_n/\sim$. This space is much smaller than that of labeled graphs $\mathfrak{M}_n$. More precisely, $|\mathfrak{M}_n/\sim| = |\mathfrak{M}_n|/n!$. This means that the more the labels are, the smaller the unlabeled-graph space is compared with the labeled-graph space. A smaller space means better performance given the same number of fitness evaluations.

The previous theorem tells us that we can implement the geometric crossover over unlabeled graphs using graph matching. As a matter of fact, to implement the geometric crossover over unlabeled graphs we *need* to use labeled graphs. The labeling results are necessary to represent and handle the solution (phenotype), even if in fact the representation (genotype) is only an auxiliary function and can be considered as not being part of the problem to solve. Graph matching before crossover allows to implement the geometric crossover on the unlabeled-graph space using the corresponding geometric crossover over the auxiliary space of the labeled graph after graph matching.

In the following we summarise, for the specific case of graphs, what the elements of the commutative diagram of the quotient geometric crossover are:

• *Genotypes G*: labeled graphs with the same number of nodes represented as adjacency matrices of the same size.

• *Phenotypes P*: unlabeled graphs.

• *Equivalence relation $\sim$*: adjacency matrices with the same underlying unlabeled graph.

• *Distance on genotypes d_G*: Hamming distance between adjacency matrices.

• *Distance on phenotypes d_P*: labeling-independent distance between unlabeled graphs. This equals the *edge edit distance*.

• *Crossover on genotypes X_G*: traditional crossover on adjacency matrices seen as vectors.

• *Crossover of phenotypes X_P*: graph matching before traditional crossover on adjacency matrices.

• *Induced crossover transformation gt*: graph matching.

The benefit of applying the quotient geometric crossover on graphs is the design of a crossover better tailored to graphs. The notion of graph matching before crossover arises directly from the definition of quotient geometric crossover. Graphs are very important because they are ubiquitous. Graphs and groupings can be seen as particular cases of labeled structures in which the fitness of a solution depends only on the structure and not on the specific labeling. So, it may be possible to study the general class of labeled structures in combination with quotient geometric crossover.

10.5 Sequences

In this section we recast alignment before recombination in variable-length sequences as a consequence of quotient geometric crossover. In chapter 8 we have applied geometric crossover to variable-length sequences. The distance for variable-length sequences we used there is the *edit distance* LD[5]: the minimum number of insertion, deletion and replacement of single character to transform one sequence into the other. The geometric crossover associated with this distance is the *homologous geometric crossover*: two sequences are aligned optimally before recombination. Alignment here means allowing parent sequences to be *stretched* to match better with each other.

After the optimal alignment, one does the normal crossover and produce a new *stretched* sequence. The offspring is obtained by unstretching the sequence. How does quotient geometric crossover fit in here? We can define a relation $\sim$ on stretched sequences: *each stretched sequence belongs to the class of its unstretched version*. Then, we can easily check that the relation $\sim$ is an equivalence relation. Let $\langle s \rangle$ be the set of all stretched sequences of sequence s. We define the induced distance measure $d_\sim$. Let s_1, s_2 be variable-length sequences. If H is the Hamming distance for stretched sequences,

$$d_\sim(s_1, s_2) := \min_{s_1' \in \langle s_1 \rangle, s_2' \in \langle s_2 \rangle} H(s_1', s_2').$$

Then, by the definition of edit distance, $d_\sim$ is equal to LD. Hence $d_\sim$ is a metric on variable-length sequences.

In the following we summarise, for the specific case of sequences, what the elements of the commutative diagram of the quotient geometric crossover are:

- *Genotypes G*: variable-length stretched sequences
- *Phenotypes P*: variable-length (unstretched) sequences
- *Equivalence relation $\sim$*: stretched sequences with the same unstretched sequence
- *Distance on genotypes d_G*: If the two stretched sequences have different length, add as many '-' as

[5]The notation LD comes from *Levenshtein distance* that is another name of edit distance.

necessary at the right end of the shorter sequence to make it become equal in length to the longer sequence. Their genotypic distance is then their Hamming distance.

• *Distance on phenotypes* d_P: edit distance between sequences

• *Crossover on genotypes* X_G: traditional crossover on stretched sequences. If the two stretched sequences have different length, add as many '-' as necessary at the right end of the shorter sequence to make it become equal in length to the longer sequence.

• *Crossover of phenotypes* X_P: homologous crossover for sequences

• *Induced crossover transformation* gt: optimal alignment

Phenotypes are variable-length sequences that are directly representable. So in this case the quotient geometric crossover is not used to search a non-directly representable space (phenotypes) through an auxiliary directly representable space (genotypes). The benefit of applying the quotient geometric crossover on variable-length sequences is that the homologous crossover over sequences X_P is naturally understood as a transformation gt of the geometric crossover X_G over stretched sequences G rather than a crossover acting directly on sequences P. This is because the notion of optimal alignment is inherently defined on stretched sequences and not on simple sequences.

Normalization before recombination and alignment before recombination arise from quotient crossovers based on two complementary types of equivalence relations in the following sense. The former is associated with an equivalence relation that considers as equivalent two solutions with the same structure independently from their labeling. Normalization does in fact adapt the labeling to the structure before recombination. Alignment is associated with an equivalence relation that considers as equivalent two solutions with the same labeling independently from how stretched their structures are. So, opposite to normalization, alignment adapts the structure to the labeling before recombination.

In chapter 11, we study the effect of the quotient transformation associated with normalization before recombination on the fitness landscape. Seeing alignment as a quotient crossover allows us to study how alignment affects the fitness landscape associated with geometric crossover and to compare the differences of these two complementary quotient transformations of the fitness landscape.

10.6 Glued space

In chapter 4, we have considered geometric crossover for glued spaces for real domains. In this section we illustrate that gluing as a topological transformation is associated with quotient geometric crossover.

In general, the solution space of real problems is a range. As we have seen in chapter 4, the geometric crossover on this bounded domain with Euclidean distance is biased toward the center of the space. One method to eliminate this bias is gluing opposite boundaries together. Figure 4.7 shows a glued space in $\mathbb{R}^2$.

The interesting fact is that this glued space can be considered as a quotient space.

In Figure 4.9 (a) shows the segment on Euclidean space and (b) shows the segment on quotient space. In quotient space, segments may cross the boundaries.

In the following we summarise, for the specific case of real vectors, what the elements of the commutative diagram of the quotient geometric crossover are:

- *Genotypes G*: bounded continuous space
- *Phenotypes P*: glued continuous space
- *Equivalence relation $\sim$*: points between which, for each coordinate, the difference is a multiple of the range size
- *Distance on genotypes d_G*: Euclidean distance
- *Distance on phenotypes d_P*: glued distance (Pacman-world distance)
- *Crossover on genotypes X_G*: segment crossover (in the Euclidean space geometric crossover becomes the line recombination.)
- *Crossover of phenotypes X_P*: segment crossover in Pacman-world spaces (the 2D Euclidean space becomes the surface of a torus. Geometric crossover is defined on segments in this space.)
- *Induced crossover transformation gt*: points that were distant and close to opposite boundaries

become close.

The original crossover X_G on genotypes is biased toward the center of the space. The benefit of quotient geometric crossover in this case is to transform the topology of the space and search the space of phenotypes that is isotropic (every point is the center, or there is no center) whose associated geometric crossover X_P has no such a bias. An unbiased space gives the same a priori opportunity to each point of the space to be searched without giving arbitrary preference to points close to the center. The search is therefore guided by the fitness values encountered only.

10.7 Functions

Here we consider functional representations. These include any representation that encodes a function. Examples of this type of representation are genetic programming trees, finite state automata, neural networks, and so on. Among them, we will see an example for genetic programming (GP). We can define an equivalence relation: *all symbolic expressions that represent the same function*. We can also define a weaker equivalence relation: *consider as equivalent those syntactic trees that differ in the order of the operands in nodes with commutative operations.* For example, the multiplication operation '$*$' is commutative and two different trees represent the same function. The quotient geometric crossover corresponds to homologous crossover for GP trees with reordering of commutative subtrees to have minimum structural Hamming distance. This quotient geometric crossover is based on the weaker equivalence relation. However, this quotient geometric crossover cannot be implemented efficiently because the complexity of computing the structural Hamming distance between rooted unordered trees grows exponentially with the number of nodes in the trees.

In the following we describe the elements of the commutative diagram of the quotient geometric crossover for the specific case of functions:

• *Genotypes G*: syntax trees that are a compact (shorter than extensive form), redundant (the same function can be represented by more than one syntax tree) and biased (some functions have more associated syntax trees than other functions) representation of functions.

• *Phenotypes P*: computed functions. A generic function can be directly represented in an extensive form as a vector in which for every combinations of the input values there is an element that contains the output of the function for those values. We call this vector *the output vector representation* of the function. Clearly this direct representation in practice is not used because it is too long.

• *Equivalence relation $\sim$*: parse trees that correspond to the same function or equivalently with the same output vector.

• *Distance on genotypes d_G*: structural Hamming distance between syntax trees

• *Distance on phenotypes d_P*: (weighted) Hamming distance on output vectors

• *Crossover on genotypes X_G*: homologous crossover for syntax trees

• *Crossover of phenotypes X_P*: traditional crossover on the output vectors of the functions

• *Induced crossover transformation gt*: expand/reduce/change syntactic trees before crossover without changing the underlying computed functions in such a way that they have minimal structural Hamming distance

The benefit of applying quotient geometric crossover to syntax trees is to search the space of the functions represented by the syntax trees rather than the space of syntax trees. This is done indirectly by manipulating syntax trees. In principle a function can be represented directly using its output vector representation. So, recurring to an auxiliary genotypic representation and using the quotient geometric crossover to search this space is not strictly necessary. However, such direct representation is simply too large for any practical purpose, and a concise genotypic representation is needed.

The implementation of the phenotypic geometric crossover using the transformation gt on the genotypic crossover X_G presents a problem: it is simply not possible to compute efficiently the transformation gt because one needs to compute the smallest structural Hamming distance between all possible transformations of the syntactic trees that keep their underlying functions invariant. We could then consider the weaker equivalence relation, mentioned before, in which two parse trees are equivalent if exchanging subtrees of nodes with commutative operations they become equal. In this case d_P becomes the distance between rooted (partially) unordered trees. This distance could be still hard to compute and consequently the same would hold for the associated geometric crossover X_P. However, there are quick approximating algorithms to compute this distance.

We can apply the concept of quotient geometric crossover to other examples with functional representations such as finite state machines and neural networks in a similar way.

10.8 Neutrality

Neutrality is wide-spread in nature, so studying neutrality is important. The role of neutrality is little understood. Notice that *neutrality* is a synonym of redundancy. As a rule of thumb one would like to filter out redundancy as in normalization for structural problem to improve performance. However, neutrality may have some beneficial aspect on variable-length representations. For example it could give rise to a self-adaptive mutation rate at a phenotypic level on variable-length genotypes, as explained in the following. Imagine one has a constant number of mutations at a genotype level. If the part of the genotype used to code for the phenotype (coding part) is small compared with the non-coding part of the genotype (neutral part), mutation at genotype level have a small chance to affect the phenotype. So the same mutation rate at genotype level can correspond to a smaller or equivalent mutation at a phenotype level depending on the amount of neutral code in the genotype. Since the mutation inserts or deletes neutral code, lengthening or shortening the neutral part of the genotype, this combined with selection may develop a self-adaptive mechanism that selects genotypes with the right amount of neutral code to be more evolvable [43].

Quotient geometric crossover can be used to understand how crossover and neutrality interact. In fact the induced geometricity-preserving transformation tells what *trick* to use to remove redundancy for crossover but still keep it there for mutation to obtain the self-adaptive mutation rate *trick*, so to take advantage of both genotypic and phenotypic spaces.

In the following we describe the elements of the commutative diagram of the quotient geometric crossover for the specific case of nutrality.

• *Genotypes G*: sequence with neutral code (part of the sequence that if removed would not affect the phenotype)

• *Phenotypes P*: sequence without neutral code. There is a one-to-one mapping between these sequences and the phenotypes. So this is a direct representation of the phenotypes, rather than the phenotype itself.

• *Equivalence relation $\sim$*: two sequences with neutral code are equivalent if when the neutral code is removed they become the same phenotypic sequence

• *Distance on genotypes d_G*: edit distance on sequences with neutral code

• *Distance on phenotypes d_P*: edit distance on sequences without neutral code

• *Crossover on genotypes X_G*: homologous crossover for sequences

• *Crossover of phenotypes X_P*: homologous crossover for sequences

• *Induced crossover transformation gt*: identity transformation

The benefit of the quotient geometric crossover is to show how crossover and neutral code interact. We have seen that neutrality may be beneficial in terms of adaptive mutation rate. Since the induced crossover transformation is the identity transformation, this means that the same crossover that searches the genotypes space can be understood as a crossover searching the phenotype space indirectly using the genotypes. In other words, the neutral code is completely transparent to the search done by crossover and it does not affect its search or performance. So, neutral code retains the advantage of an adaptive mutation rate together with being transparent to the action of crossover.

10.9 Summary

In this chapter we have extended the geometric framework introducing the notion of quotient geometric crossover. This could be clearly understood using the concept of geometricity-preserving transformation. Quotient geometric crossover is a very general and versatile tool. We have given a number of interesting examples as its applications. As shown in applications, quotient geometric crossover is not only theoretically significant but also has a practical effect of making search more effective by reducing the search space or removing the inherent bias.

Chapter 11

Graph Partitioning

The standard encoding for multiway graph partitioning is highly redundant: each solution has a number of representations, one for each way of labeling the represented partition. Traditional crossover does not perform well on redundant encodings. We propose a geometric crossover for graph partitioning based on a labeling-independent distance that filters out the redundancy of the encoding. A correlation analysis of the fitness landscape based on this distance shows that it is well suited to graph partitioning.

A second difficulty with designing a crossover for multiway graph partitioning is that of feasibility: in general recombining feasible partitions does not lead to feasible offspring partitions. The generalization of cycle crossover for permutations with repetitions presented in chapter 5 naturally suits partition problems. We adapt it to the graph partitioning problem and test it experimentally. We then combine it with the labeling-independent crossover and obtain a much superior geometric crossover inheriting both advantages.

11.1 Introduction

Grouping problems are interesting and NP-hard [44]. In this chapter, we focus on the multiway graph partitioning problem. When applying evolutionary algorithms to grouping problems, the standard solution encoding is highly redundant. This affects badly the performance of traditional crossover. In [16], a highly effective procedure to compensate for redundancy was introduced that requires a labeling-normalization phase before the actual exchange of genetic material between parents.

The Hamming distance is a naïve distance for grouping problems because it does not take the inherent redundancy of the solution encoding into account. In chapter 10, we introduced a natural distance for grouping problems, the labeling-independent distance. In this chapter we prove that this distance satisfies the metric axioms and that it can be efficiently computed using the Hungarian method [78].

We use the labeling-independent distance as basis of geometric crossover to design a new crossover for redundant encodings. Interestingly, this distance gives rise to a crossover that requires a labeling-normalization phase before exchanging genetic material between parents in the traditional way. The new crossover can be implemented exactly and efficiently using the Hungarian method that, unlike previous normalization heuristics, allows obtaining a perfect normalization in an efficient way. We compare the landscapes under Hamming distance and labeling-independent distance and find that the latter landscape presents characteristics that make it better matched with the corresponding geometric crossover; so we expect that the new geometric crossover would achieve better performance. We compare experimentally the traditional crossover (geometric under the Hamming distance), the new geometric crossover based on labeling-independent distance, and a third one employing a heuristic normalization that is known to be very good [73]. The new geometric crossover shows remarkable performance improvement.

Beside redundancy of the encoding, a second difficulty with grouping problems is that traditional recombination for multary strings does not preserve offspring feasibility: recombining parents with the same grouping structures does not lead in general to offspring with the same structure, requiring a repairing mechanism to be applied to the offspring. The repairing mechanism is not necessarily negative *per se* as it can be interpreted as a form of mutation. However, when the extent of change in the offspring is large, the repairing mechanism degenerates into macro-mutation with deleterious effect on performance.

In general, a much preferred way to deal with this problem is to design a recombination operator that naturally transmits parent feasibility to offspring. We show that the generalization of cycle crossover for permutation with repetitions presented in chapter 5 allows to search only the space of feasible solutions without the need of any repairing mechanism. We then combine this cycle crossover and labeling-independent crossover obtaining a new geometric crossover with both advantages that suits very well grouping problems with redundant encodings. We test experimentally the new geometric crossovers on graph partitioning obtaining remarkable performance improvement.

The remainder of this chapter is organized as follows. In Section 11.2, we introduce the multiway graph partitioning problem. In Section 11.3, we introduce a new geometric crossover based on labeling-independent distance. In Section 11.4, we present a correlation analysis of the fitness landscapes associated with the Hamming and labeling-independent distances. In Section 11.5, we recast the graph partitioning problem in terms of permutations with repetitions and motivate the use of cycle crossover. We then combine cycle crossover and labeling-independent crossover into a new geometric crossover with both characteristics. In Section 11.6, we present experimental results.

11.2 Multiway graph partitioning

Graph partitioning is an important problem that arises in various fields of computer science, such as sparse matrix factorization, VLSI circuit placement, network partitioning, and so on. Good partitioning of a system not only significantly reduces the complexity involved in the design process, but can also improve the timing performance as well as its reliability [60].

Let $G = (V, E)$ be an unweighted undirected graph, where V is the set of vertices and E is the set of edges. *K-way partition* is a partitioning of the vertex set V into K disjoint subsets $\{C_1, C_2, \ldots, C_K\}$. A K-way partition is said to be *balanced* if the difference of cardinalities between the largest and the smallest subsets is at most one. The *cut size* of a partition is defined to be the number of edges with endpoints in different subsets of the partition. The *K-way partitioning problem* is the problem of finding a K-way balanced partition with minimum cut size.

Since the K-way partitioning problem is NP-hard [44], attempts to solve partitioning problems have focused on finding heuristics which yield approximate solutions in polynomial time. Among such methods, the Kernighan-Lin algorithm [71] [33] and the Fiduccia-Mattheyses algorithm (FM) [38] are representative. They are local search heuristics for bipartitioning (2-way partitioning).

Traditional multiway partitioning algorithms have been based on bipartitioning. Two representative approaches, which were suggested by [71], are the pairwise approach [66] and the recursive one [141] [14]. The pairwise approach starts with an arbitrary K-way partition. It picks two subsets at a time among the K subsets and performs bipartitioning between them. The other recursively bipartitions the target graph until we have K subsets.

Another method is a direct extension of the 2-way FM [141] [20]. The algorithm starts with an arbitrary K-way partition. It considers all possible moves of each vertex from its home set to any of the others. [141] showed that this direct multiway partitioning approach obtained better solutions compared to the recursive approach for random networks. An improved variation appeared in [73]. We will use this version as local optimization engine in our genetic algorithms.

In addition, there have been some studies about constructive (called *multi-level*) methods for multiway partitioning [67] [21]. There have also been several methods using genetic algorithms [80] [58] [66] [73].

11.3 Labeling-independent crossover

11.3.1 Labeling-independent distance

The standard representation of a solution for K-way graph partitioning is a vector x of size $|V|$ such as $x_i = j \Rightarrow v_i \in P_j$. Since the specific mapping of indices to partitions does not change how the graph is partitioned, each solution has $K!$ representations. For this encoding, the Hamming distance between two solutions is unnatural because it depends on the specific mapping between indices and partitions that is completely arbitrary. We propose a distance measure, the *labeling-independent distance*, that eliminates this dependency completely.

Formally, the term *distance* or *metric* denotes any real valued function that conforms to the axioms of identity, symmetry, and triangular inequality. A distance for which the axiom of identity is relaxed so that distance zero does not necessarily imply equality (but equality still implies distance zero) is called *pseudo-metric*.

Definition 11.3.1 (Labeling-independent distance).
Given two K-ary encodings $\mathfrak{a}, \mathfrak{b} \in U = \{1, 2, \dots, K\}^{|V|}$ (fixed-length vectors on a K-ary alphabet) and a metric $\mathfrak{d}$ in U, we define the *labeling-independent distance* LI associated with $\mathfrak{d}$ as follows:

$$LI(\mathfrak{a}, \mathfrak{b}) := \min_{\sigma \in \Sigma_K} \mathfrak{d}(\mathfrak{a}, \mathfrak{b}_\sigma)$$

where Σ_K is the set of all permutations of length K and $\mathfrak{b}_\sigma$ is a permuted encoding of $\mathfrak{b}$ by a permutation σ, i.e., the i^{th} element b_i of $\mathfrak{b}$ is transformed into $\sigma(b_i)$.

Theorem 11.3.1. *Labeling-independent distance LI is a pseudo-metric in U.*

Proof. It is enough to show that LI satisfies the triangle inequality.

$$\begin{aligned}
LI(\mathfrak{x}, \mathfrak{y}) + LI(\mathfrak{y}, \mathfrak{z}) &=_{(WLOG)} \mathfrak{d}(\mathfrak{x}, \mathfrak{y}_\sigma) + \mathfrak{d}(\mathfrak{y}, \mathfrak{z}_{\sigma'}) \\
&= \mathfrak{d}(\mathfrak{x}, \mathfrak{y}_\sigma) + \mathfrak{d}(\mathfrak{y}_\sigma, (\mathfrak{z}_{\sigma'})_\sigma) \\
&= \mathfrak{d}(\mathfrak{x}, \mathfrak{y}_\sigma) + \mathfrak{d}(\mathfrak{y}_\sigma, \mathfrak{z}_{\sigma\cdot\sigma'}) \\
&\geq \mathfrak{d}(\mathfrak{x}, \mathfrak{z}_{\sigma\cdot\sigma'}) \quad (\because \mathfrak{d} \text{ is a metric}) \\
&\geq LI(\mathfrak{x}, \mathfrak{z}) \quad (\because \sigma \cdot \sigma' \in \Sigma_K).
\end{aligned}$$

□

Given an element $\mathfrak{a} \in U$, since $\mathfrak{d}$ is a metric, there are $K!$ elements such that the labeling-independent distance LI to $\mathfrak{a}$ is zero. If the labeling-independent distance LI between two elements is equal to zero, we define them to be *in relation* $\sim$. Then, the following proposition holds.

Proposition 11.3.2. *The relation $\sim$ is an equivalence relation.*

Proof. It is obvious that the relation $\sim$ is reflexive and symmetric. It is transitive as in the following.

$$\begin{aligned}
\mathfrak{a} \sim \mathfrak{b}, \mathfrak{b} \sim \mathfrak{c} &\Rightarrow_{(WLOG)} \mathfrak{a} = \mathfrak{b}_\sigma, \mathfrak{b} = \mathfrak{c}_{\sigma'} \\
&\Rightarrow \mathfrak{a} = (\mathfrak{c}_{\sigma'})_\sigma = \mathfrak{c}_{\sigma\cdot\sigma'} \\
&\Rightarrow \mathfrak{a} \sim \mathfrak{c} \quad (\because \sigma \cdot \sigma' \in \Sigma_K).
\end{aligned}$$

□

Theorem 11.3.3. *Suppose that Q is the quotient set of U by relation $\sim$ (i.e., $Q = U/\sim$). Then, (Q, LI) is a metric space, i.e., the labeling-independent distance LI is a metric in Q.* [1]

Proof. By Proposition 11.3.2, Q is well defined. Since LI is a pseudo-metric by Theorem 11.3.1, it is clear that LI is a metric in Q. $\square$

So, U is the set of all *labeled partitions* and $\mathfrak{d}$ is a metric on this set. Q is the set of all *unlabeled partitions* associated to U and LI is the corresponding metric on Q associated to $\mathfrak{d}$.

11.3.2 Efficient normalization and labeling-independent distance

We say that $\mathfrak{b}_{\sigma^*}$ is normalized to $\mathfrak{a}$ when $LI(\mathfrak{a}, \mathfrak{b}_{\sigma^*}) = \mathfrak{d}(\mathfrak{a}, \mathfrak{b}_{\sigma^*})$ and we call σ^* a normalizing relabeling.

We show that, in the special case of $\mathfrak{d}$ being Hamming distance H, the problem of computing LI can be formulated as the optimal assignment problem and it can be solved efficiently by the Hungarian method (see section 11.3.3). The problem of finding a normalizing relabeling σ^* is equivalent to computing LI. Let $M = (m_{ij})$ be the $K \times K$ assignment weight matrix between two chromosomes X and Y. Each element m_{ij} means $\sum_{k=1}^{|V|} I(X_k = i, Y_k \neq j)$ or $\sum_{k=1}^{|V|} I(X_k \neq i, Y_k = j)$, where $I(\cdot)$ is the indicator function, i.e., $I(true) = 1$ and $I(false) = 0$. The problem of computing LI is exactly the problem of finding an assignment (permutation) with minimum summation.

Theorem 11.3.4. *If the metric $\mathfrak{d}$ is Hamming distance H, then*

$$LI(X, Y) = \min_{\sigma \in \Sigma_K} \sum_{i=1}^{K} \sum_{k=1}^{|V|} I(X_k = i, Y_k \neq \sigma(i)).$$

[1] *Strictly speaking, this labeling-independent distance is not exactly the same as LI on U because it is a metric on Q. However, the definition of LI on Q can be naturally induced from LI on U, i.e., for $\bar{\mathfrak{a}}, \bar{\mathfrak{b}} \in Q$, $LI(\bar{\mathfrak{a}}, \bar{\mathfrak{b}}) := \min_{\sigma \in \Sigma_k} \mathfrak{d}(\mathfrak{a}, \mathfrak{b}_\sigma)$.*

Proof.

$$\begin{aligned}
LI(X,Y) &= \min_{\sigma\in\Sigma_K} H(X, Y_\sigma) \\
&= \min_{\sigma\in\Sigma_K} \left(\sum_{k=1}^{|V|} I(X_k \neq \sigma(Y_k)) \right) \\
&= \min_{\sigma\in\Sigma_K} \left(\sum_{k=1}^{|V|} \sum_{i=1}^{K} I(X_k = i, \sigma(Y_k) \neq i) \right) \\
&= \min_{\sigma\in\Sigma_K} \left(\sum_{k=1}^{|V|} \sum_{i=1}^{K} I(X_k = i, \sigma^{-1} \cdot \sigma(Y_k) \neq \sigma^{-1}(i)) \right) \\
&= \min_{\sigma\in\Sigma_K} \left(\sum_{k=1}^{|V|} \sum_{i=1}^{K} I(X_k = i, Y_k \neq \sigma^{-1}(i)) \right) \\
&= \min_{\sigma\in\Sigma_K} \left(\sum_{i=1}^{K} \sum_{k=1}^{|V|} I(X_k = i, Y_k \neq \sigma^{-1}(i)) \right) \\
&= \min_{\sigma\in\Sigma_K} \left(\sum_{i=1}^{K} \sum_{k=1}^{|V|} I(X_k = i, Y_k \neq \sigma(i)) \right) \quad (\because \sigma \in \Sigma_K \Leftrightarrow \sigma^{-1} \in \Sigma_K).
\end{aligned}$$

□

11.3.3 Optimal assignment problem

Consider a weighted complete bipartite graph with bipartition (X, Y), where $X = \{x_1, x_2, \ldots, x_K\}$, $Y = \{y_1, y_2, \ldots, y_K\}$, and each edge $(x_i, y_j) \in X \times Y$ has a weight w_{ij}. The *optimal assignment problem* is the problem of finding a maximum-weight (or minimum-weight) perfect matching in this weighted graph as follows:

$$\max_{\sigma\in\Sigma_K} \left(\sum_{i=1}^{K} w_{i\sigma(i)} \right) \quad \text{or} \quad \min_{\sigma\in\Sigma_K} \left(\sum_{i=1}^{K} w_{i\sigma(i)} \right)$$

where σ is a permutation. It is also known as the bipartite weighted matching problem.

To solve the optimal assignment problem, it is possible to enumerate all $K!$ permutations in Σ_K and find an optimal one among them. However, for a large K, such a procedure is intractable. Fortunately, [78] proposed an efficient way to solve the problem. It is called the *Hungarian method*. Roughly speaking, starting from the initial unweighted bipartite graph with no edge, the method iteratively modifies edge weights, adds new edges into the bipartite graph, and applies the *maximum matching* (or *minimum covering*) in the resulting bipartite graph. It continues the above process until a perfect match is found. The Hungarian method gives an optimum assignment and it can be implemented in $O(K^3)$ time [121]. There are also some fast heuristic algorithms for solving the assignment problem approximately [7]. They can be implemented in $O(K^2)$ and experimental studies indicate that these methods give solutions very close to the optimum and are faster than the Hungarian method.

11.3.4 Geometric crossover under labeling-independent metric

We design a new geometric crossover based on the labeling-independent metric. We call it n-point LI-GX.

Definition 11.3.2 (n-point LI-GX).
Normalize the second parent to the first under Hamming distance. Do the normal n-point crossover using the first parent and the normalized second parent.

Theorem 11.3.5. *The n-point LI-GX is geometric under the labeling-independent metric.*

Proof. Let p_1 and p_2 be parents and c be offspring after n-point LI-GX. It is enough to show that $LI(p_1, p_2) = LI(p_1, c) + LI(c, p_2)$. Since LI is a metric, by triangular inequality, it is trivial that $LI(p_1, p_2) \leq LI(p_1, c) + LI(c, p_2)$. Now, we will show that $LI(p_1, p_2) \geq LI(p_1, c) + LI(c, p_2)$. Let σ' be $\text{argmin}_{\sigma \in \Sigma_K} H(p_1, (p_2)_\sigma)$, where $(p_2)_\sigma$ is a permuted encoding of p_2 by σ, i.e., the i^{th} element p_{2i} of p_2 is transformed into $\sigma(p_{2i})$. Then, let $p_2' = (p_2)_{\sigma'}$.

$$\begin{aligned}
LI(p_1, p_2) &= H(p_1, p_2') \\
&= H(p_1, c) + H(c, p_2') \\
&\geq LI(p_1, c) + LI(c, p_2') \\
&= LI(p_1, c) + LI(c, p_2).
\end{aligned}$$

□

To implement n-point LI-GX, we use the Hungarian method to find a maximum matching of the labels of the second parent to the first parent and then we apply a traditional n-point crossover. The time complexity of n-point LI-GX is $O(|V| + K^3)$.[2]

Here, labeling-normalization itself is not our original idea. There have been some studies dealing with relabeling of partitions [80] [58] [66]. They used approximate labeling-normalization

[2]We consider only the optimal method. However, there is the possibility of a faster implementation ($O(|V| + K^2)$) thanks to the algorithm for normalizing parents approximately mentioned in section 11.3.3.

to alleviate difficulties caused by redundant encodings in genetic algorithms for multiway graph partitioning. These methods are greedy heuristics for finding a good relabeling. Instead, the proposed algorithm based on the Hungarian method always finds the optimal normalization efficiently.

11.4 Landscape of labeling-independent crossover

11.4.1 Smoothness

Measuring smoothness

To measure the smoothness of a fitness landscape, we will plot its autocorrelation function (or *correlogram*): a graph that on the abscissas has the distance between solutions and on the ordinates has the correlation between their fitness. The autocorrelation function for fitness landscapes was introduced by [160].

To have a meaningful comparison between correlation functions of fitness landscapes based on different search space structures with different distance distributions, we normalize distances by dividing them by the average distance $E(d)$ between any two points of the search space. Let $r_d(\delta)$ be the normalized correlogram of a fitness landscape based on a metric d, where δ is the distance between points in the search space. Then $r_d(1)$ is the average distance correlation and is a measure of global smoothness of the landscape. If $r_d(1)$ is positive, we say that the landscape is globally smooth. If it is negative, the landscape is said to be globally discontinuous. If the normalized correlogram of a landscape is above the normalized correlogram of a second landscape, then the former is smoother than the latter.

Correlogram of local-optimum space

Our evolutionary algorithm uses crossover together with hill-climbing (see Section 11.6.1). Therefore, the space searched is the local-optimum space[3] rather than the whole space. So, we have to consider the correlogram of such a restricted space. However, it is not easy to plot it

[3]The local-optimum space is a metric space with the same distance function of the all-partition space. Its domain set is the domain set of all-partition space restricted to the set of local optima.

Algorithm 4 Approximated correlogram routine

```
Get randomly generated $M$ local optima $s_i$'s
for each $p \in (0, 100]$ do
    Get corresponding $M$ local optima $t_i$'s after random $p\%$ perturbation + local optimization
    for each pair of $M$ pairs $(s_i, t_i)$'s do
        Compute Hamming distance ($h_i$) and labeling-independent distance ($l_i$)
    end for
    Compute the correlation coefficient ($c$) between the fitness of $M$ pairs
    Plot a point at position $(\frac{1}{M}\sum h_i/E(H),\ c)$
    Plot a point at position $(\frac{1}{M}\sum l_i/E(LI),\ c)$
end for
```

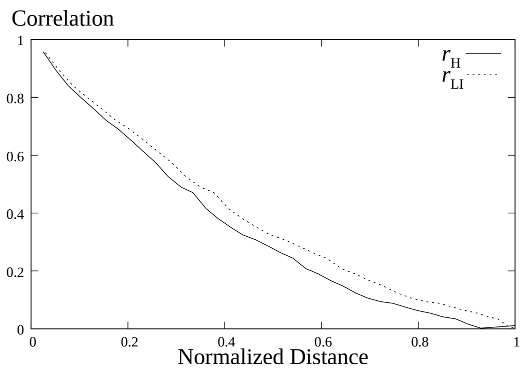
exactly. Given a distance value δ, it is not easy to find pairs of local optima such that their distance is equal to δ by randomly sampled local optima. However, an approximated correlogram can be obtained using Algorithm 4.

First, we get M local optima by applying the hill-climber[4] to M randomly generated solutions. In our experiments, we set M to be 3,000. For each perturbation rate p (see Algorithm 4), we change p percent of the bits of each local optimum at random and apply the local optimizer to the perturbed solution to obtain a new local optimum. Then, we compute the Hamming distance and the labeling-independent distance between the new local optimum and the previous one. After computing the correlation coefficient between the fitness of M pairs, we plot the relation between the distances and the correlation coefficient. Notice that the average of h_i's and the average of l_i's will increase as the perturbation rate p increases.

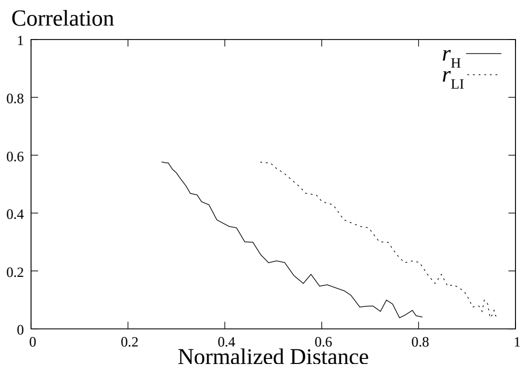
Distance distributions

Let us consider the search space for 32-way partitions on a graph with 500 vertices (problem instance G500.2.5 of the testbed described in section 11.6.2). We can compute the expected value $E(H)$ in all-partition space approximately. In all-partition space, the probability that the i^{th} positions of two solutions are the same is $1/32$. Hence, $E(H)$ is approximately $500 \times (1 - 1/32) = 484.375$. This fact hints that most pairs of points are nearly at a maximum distance. This is interesting and counterintuitive. Table 11.1 shows the average and the maximum distance for

[4]We used the local optimization algorithm described in Section 11.6.1.



(a) All-partition space



(b) Local-optimum space*

Figure 11.1: Normalized correlogram (G500.2.5)

$*$ There were some missing points in (b) because we could hardly obtain close pairs of local optima.

each space. It shows that the problem space becomes much narrower in the labeling-independent space (LI-space).

Table 11.1: Average and maximum distance of search spaces

Metric space	$E(d)$	$\max(d)$
(all-partition, H)	484.375	500
(local-optimum, H)	484.369	$\leq \max(H)$ in all-partition space
(all-partition, LI)	429.010	$\leq 484^{\ddagger}$
(local-optimum, LI)	274.301	$\leq \max(LI)$ in all-partition space

The values of $E(d)$ were empirically computed from a pool of 500 randomly sampled solutions. The value marked with $\ddagger$ is a simple upper bound of $\max(LI)$ of the all-partition space found as follows. Let a and b be 32-way partitions of 500 vertices. The size of the largest subset for a or b is $\lceil 500 \times 1/32 \rceil = 16$. Let i and j be the indices of the largest subsets of a and b, respectively. Then, $LI(a, b) \leq H(a, b_\sigma)$, where σ is a permutation such that $\sigma(j) = i$. Since $H(a, b_\sigma) \leq 500 - 16 = 484$, $LI(a, b) \leq 484$.

Comparison

Figure 11.1 shows normalized correlograms for the all-partition space and the local-optimum space. The abscissa means normalized distance between solutions and the ordinate is the correlation coefficient between their fitness. First, we compare the H-space and the LI-space based on all partitions. The normalized correlograms look similar (see Figure 11.1a). The LI-correlogram is always above the H-correlogram, but not by much. So, we can say that the LI-landscape is slightly smoother than the H-landscape. For both correlograms, points that are at a smaller than average distance have increasingly strong correlation; this makes both landscapes globally smooth and, so, suitable for geometric crossover.

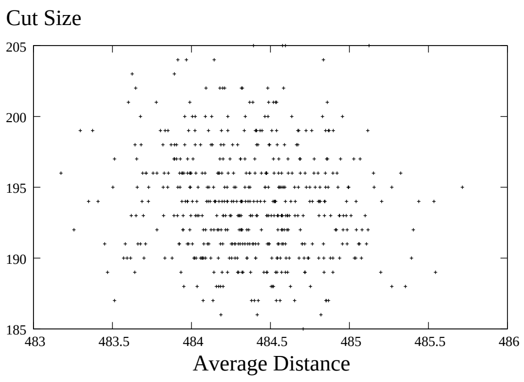
We expect the geometric crossover based on LI-landscape to perform better than the one based on H-landscape. This is only partially due to the stronger correlation. Mostly it is due to the fact that the sizes of the spaces that are actually searched by geometric crossover are different: H-crossover searches the space of all labeled-partitions and LI-crossover searches the much smaller space of all unlabeled-partitions.

Next, we compare the H-space and the LI-space on the local-optimum space. It is interesting to note that the average distance in the local-optimum LI-space is much smaller than the average distance in the all-partition LI-space, whereas for the H-space there is almost no difference between the average distances in the whole space and the local-optimum space. The LI-space is a quotient space of the H-space induced by normalization. Evidently this topological transformation remaps all the peaks and clusters them. The reason why normalization has this effect on the fitness landscape remains an open question.

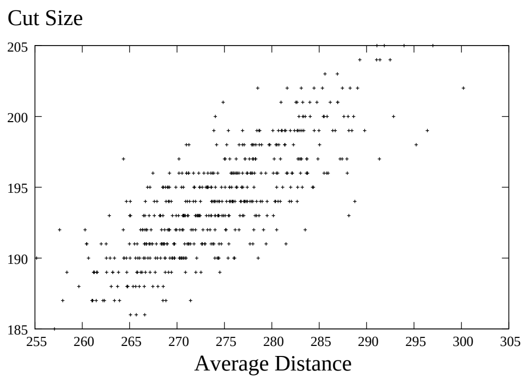
Since the average distance between local optima for LI is so much smaller than that for H, the respective normalized correlograms are strongly affected (see Figure 11.1b): the LI-landscape is really much smoother than the H-landscape (LI-correlogram is much above H-correlogram). So, from the rule-of-thumb that smoother landscape has better geometric crossover performance, we expect that, on the local-optimum space, the geometric crossover based on labeling-independent distance (LI-GX) performs much better than the geometric crossover based on the Hamming distance (H-GX).

11.4.2 Global convexity

Given a set of local optima, [12] plotted, for each local optimum, the relationship between the cost and the average distance from all the other local optima. They performed experiments for the graph bisection and the traveling salesman problem, and showed that both problems have



(a) Hamming distance



(b) Labeling-independent distance

Figure 11.2: The relationship between cost and distance (G500.2.5)

strong positive correlations. This fact hints that the best local optimum is located near the center of the local-optimum space and, roughly speaking, the local-optimum space is globally convex. In this section, we repeat their experiments for multiway graph partitioning with different distances.

The solution space for the experiment is selected as follows. First, we choose 500 random solutions and obtain the corresponding set of local optima by locally optimizing them[5]. Table 11.2 shows the results for 32-way partitioning. Each value represents the cost-distance correlation for each instance and distance. Figure 11.2 shows a sample plotting of an instance (G500.2.5). The result with the labeling-independent metric (figure 11.2b) was consistent with the results of Boese *et al.* [12] with strong positive cost-distance correlation. On the other hand, the result with Hamming distance (figure 11.2a) showed little correlation.

Table 11.2: The relationship between cost and distance

Graph	Correlation coefficient		Graph	Correlation coefficient	
	H	LI		H	LI
G500.2.5	−0.11	0.79	U500.05	−0.00	0.78
G500.05	0.03	0.79	U500.10	−0.09	0.66
G500.10	−0.00	0.78	U500.20	0.05	0.77
G500.20	−0.01	0.78	U500.40	0.00	0.62

[5]We used the local optimization algorithm described in Section 11.6.1.

In summary, there are three reasons we expect LI-GX to perform better than H-GX: (i) global convexity (by cost-distance correlation [12]), (ii) the smaller size of problem space (from average distance), and (iii) smoothness (by autocorrelation [160]). Properties (ii) and (iii) are more obvious in the local-optimum space.

11.5 Cycle crossovers for graph partitioning

11.5.1 Searching balanced partitions

The multiway graph partitioning problem is a constrained optimization problem. Feasible solutions are those in which the sizes of the partitions are balanced. A way to dealt with such a constraint is using a crossover that searches the space of *all* solutions and then applying a repairing mechanism to make offspring feasible. There are other ways to deal with constraints.

An alternative method that does not need to use any repairing mechanism is to have a geometric crossover that searches only the space of balanced solutions. This is the approach we take here. This reduces the size of the search space considerably (the set of balanced solutions is a fraction of the whole search space).

Representation: The starting point for restricting the search to balanced-partitions only is to see the object representing the solution not as a vector of integer but as a permutation with repetitions. Every position in the permutation still represents a vertex of the graph and the integer at that position still represents the label of the group the vertex is assigned to.

A solution is balanced when the number of vertices of any two partitions differ almost of one. For example, in the solution `(a b a c b c c)` vertices 1 and 3 are assigned to group `a`, vertices 2 and 5 to group `b`, and vertices 4, 6 and 7 to group `c`. This solution is balanced because the sizes of the groups differ at most of 1 since `a` and `b` have two elements and `c` has three elements. *Notice that two balanced solutions do not necessarily belong to the same repetition class.* For example, another balanced solution is `(a b a a b c c)` in which the group `a` has three elements and the groups `b` and `c` have two elements. So, the permutation with repetitions representing the former solution has sizes of repetition classes $n_a = 2, n_b = 2$ and $n_c = 3$, whereas the permutation with repetitions representing the latter solution has sizes of repetition classes $n_a = 3, n_b = 2$ and $n_c = 2$.

However, we can restrict the search to the space of balanced solutions by using search operators that preserve the repetition classes of the permutations with repetitions as follows.

Equally balanced initial population: we need to seed the initial population with balanced solutions represented by permutation with repetitions belonging to the same repetition class. Seeding the population with balanced solutions is not sufficient.

A balanced crossover (cycle crossover): The generalization of cycle crossover presented in chapter 5 preserves repetition class. So, given two balanced parents belonging to the same repetition class, it returns offspring of the same repetition class, hence balanced. So, there is no need for repairing mechanisms.

A balanced mutation (swap mutation): We need to use a mutation that keeps a permutation with repetitions within the same repetition class. So that, if a solution is balanced, the mutated solution is still balanced. A simple mutation with this characteristic is the simple swap mutation based on the swap move.

In chapter 5, the cycle crossover was shown to be non-geometric under swap distance. However, we have also seen that in practice cycle crossover is a good approximation of a geometric crossover under swap distance.

11.5.2 Crossover landscape of cycle crossover

As seen in chapter 3, one important benefit of knowing that a certain search operator is geometric under a specific distance is that it is possible to tell *a priori* whether the search operator is likely to be a good choice for a specific problem. As a rule-of-thumb, search operators associated with a smooth fitness landscape are likely to perform well. In the following, we will show that the fitness landscapes associated with the swap move is smooth, making the search operators proposed a good choice for the graph partitioning problem.

In the following, we derive the normalized Lipschitz constant for the graph partitioning problem under swap distance. The fitness function (to minimize) is the cut size: the number of edges that connect nodes in different groups. The best possible fitness is zero that corresponds to the case in which all nodes are connected only with nodes of the same group. Now, let us consider the worst possible fitness. If d is the maximum degree of a node in the graph (the maximum number of connections with other nodes) the worst case happens when each node has d connections with nodes not in its group. So, if we have n nodes in the graph, the worst possible fitness is $n \cdot d/2$. The division by two arises from the fact that we count the edges between the nodes, not the pairs of nodes. Each edge correspond to two pairs of nodes (with reversed order of nodes). In the worst case, the *maximum global fitness variation* is, therefore, $\Delta F = n \cdot d/2$, that is the difference between the worst possible fitness and the best possible fitness (zero).

Let us now consider the *maximum possible fitness variation* of two solutions one swap away. The worst case happens when two nodes belonging to different groups that have connections only with nodes of their own groups are swapped. So, in the worst case the total change in fitness due to a single swap is $\Delta f_{\text{swap}} = 2d$ because, in the transition to a different group, each node increments the cut size by its degree.

The normalized Lipschitz constant $\bar{k}$ of the landscape is maximum local variation over maximum global variation. The smaller this measure, the smoother the landscape.

$$\bar{k} = \Delta f_{\text{swap}}/\Delta F = 2d/(n \cdot d/2) = 4/n.$$

This does not depends on the maximum degree d of the node in the graph but only on the number n of its nodes.

For $n = 500$, as typical instance size of the problem, we have a value of landscape smoothness $\bar{k} = 0.008$. *The fitness landscape under swap distance is very smooth* when compared with the two extreme of smoothness 0 (no change between neighbor solutions) and 1 (maximum and minimum are neighbors).

It is also interesting to notice that the smoothness $\bar{k}$ of the landscape is inversely proportional to the problem size n. *So, for increasing size of the problem, the fitness landscape becomes smoother.* This may counteract the increasing difficulty of the search due to a larger space, making the performance of the evolutionary algorithm scale well.

In summary, the cycle crossover has a number of advantages. It searches only feasible solutions without any need of repairing mechanism. Doing so, it also restricts the search to a much smaller set of candidate solutions than the original one. Finally, it is associated with a very smooth landscape that makes it a good choice for the graph partitioning problem.

11.5.3 Combining labeling-independent crossover and cycle crossover

Combination of relabeling and balanced solutions: Cycle crossover (Cycle H-GX) searches the space of balanced partitions. LI-GX (see Section 11.3) searches the space of labeling-independent partitions. We combine these two geometric crossovers obtaining a new geometric crossover with both advantages: it operates fully within the space of labeling-independent balanced partition space, which is a fraction of the original space and could produce highly competitive performance. *The new crossover (Cycle LI-GX) consists of a labeling-normalization phase before applying cycle crossover.* The normalization is done using the Hungarian method as for LI-GX.

Geometricity of compound crossover: Cycle LI-GX is still geometric on the phenotypic space restricted to balanced phenotypes. It is in fact the traditional mask-based crossover restricted to the subspace of vectors that take the form of fixed-size class permutations with repetitions.

Equally balanced solutions: Relabeling a solution does not affect its balancedness but it may change its repetition class. Cycle crossover without normalization is able to deal with different partition sizes. Cycle crossover plus normalization requires all partitions to have exactly the same size. In this special case, the relabeling does not change the repetition class of the solution and the cycle crossover is therefore applied to solutions of the same repetition class.

Inexact balance and cycle crossover: However we would like to apply the cycle crossover with normalization not only to the restricted class of problems with partitions of exactly the same size but to all balanced partitions. We therefore have to further extend cycle crossover to the recombination of solutions belonging to different repetition classes. When attempting to use the cycle crossover on permutations of different repetition classes, we may not obtain a proper decomposition in cycles. In this case we consider some non-cycles (paths) as cycles and proceed with the recombination. (For details, see section 5.8.4). This modification produces offspring with intermediate repetition class with respect to the repetition classes of the parents.

11.6 Experiments

11.6.1 Genetic framework

We used the general structure of hybrid steady-state genetic algorithms. In the following, we describe the framework of a genetic algorithm used in our experiments. In this framework, we will test different crossover operators.

- *Encoding*: We use a K-ary string for each individual to represent a K-way partition. For example, if vertex v_i belongs to partition C_j, the value of the i^{th} gene is j.

- *Initialization*: We randomly create p individuals with balanced partitions. We set the population size p to be 50.

- *Selection*: We use the roulette-wheel-based proportional selection scheme with dynamic scaling. The probability that the best individual in the population is chosen was set to four times higher than the probability that the worst individual in the population is chosen. To do this, the fitness value f_i of the i^{th} chromosome can be represented as

  $$f_i = (c_w - c_i) + (c_w - c_b)/3$$

  where c_b, c_w, and c_i are the cut sizes of the best chromosome, the worst chromosome, and the i^{th} chromosome, respectively.

- *Mutation*: After traditional 5-point crossover with/without normalization (5pt H-GX, GEFM[73], and 5pt LI-GX),[6] individuals are usually not balanced. We run the following

[6]Among n-point crossovers, 5-point crossover has been widely used in the literature [14] [66] [73] [61].

balance adjustment used in [73]. This can be interpreted as mutation, so we do not add any specific mutation after these crossovers.

Algorithm 5 Pseudo-code for balance adjustment

```
Function: offspring balance_adjust(offspring)
for $k = 1$ to $K$ do
  if partition $k$ is deficient (partition size $\leq |V|/K - 1$) then
    Change the gene values belonging to the excessive partitions (partition size $> |V|/K$)
    into the value $k$,
    starting at a random gene position and moving to the right
    (wrapping around if necessary) until the balance is satisfied;
  end if
end for
for $k = 1$ to $K$ do
  if partition $k$ is excessive then
    Change the gene values belonging to partition $k$
    into an arbitrary not-excessive partition number,
    starting at a random gene position and moving to the right
    (wrapping around if necessary) until the balance is satisfied;
  end if
end for
return offspring;
```

After cycle crossover (Cycle H-GX) or normalized cycle crossover (Cycle LI-GX), we run the following swap mutation.

Algorithm 6 Pseudo-code for swap mutation

```
Function: offspring swap_mutate(offspring)
for $i = 1$ to $n$ do
  if p_mut > rnd_real_range(0,1) then
    offspring = do_rnd_swap(offspring, $i$);
  end if
end for
return offspring;
```

where $n = |V|$, `rnd_real_range(0,1)` returns random real numbers in the range [0,1] and `do_rnd_swap(offspring,` i`)` chooses a random point j ($j \neq i$) and swaps positions i and j.

The mutation parameter `p_mut` is set to be 0.005. Then, the expected Hamming distance between individuals before and after mutation is approximately 1% of the problem size $|V|$ (because a single swap affects two positions).

- *Local optimization*: As local optimization engine in our genetic algorithm, we use the algorithm proposed in [73]. (See Section 11.2) Its time complexity is $O(K|E|)$.
- *Replacement*: If it is superior to its closest parent, the offspring replaces it; if not, the other parent is replaced by the offspring if the offspring is better. Otherwise the worst in the population is replaced.
- *Stopping criterion*: For stopping, we use the number of consecutive failures to replace one of the parents. We set the number to 50.

11.6.2 Test environment

Before showing the experimental results, we introduce the benchmarks used in the experiments and test environment. Our test-bed is composed by two groups of graphs, four random graphs (whose names start with 'G') and four random geometric graphs (whose names start with 'U'), all with 500 vertices from [63]. The two classes were used in a number of other graph-partitioning studies [14] [10] [82] [74].

We conducted tests on 32-way and 128-way partitioning. A *C* language program was used on a Pentium III 1GHz computer with Linux operating system. It was compiled using *gcc* compiler.

We know the best-known solutions for the 32-way partitioning from previous literature [66] [73] [72]. However, as far as we know, the 128-way partitioning was never tested on the problem instances in our test-bed.

11.6.3 Results

We compare the geometric crossover based on the Hamming distance (5pt H-GX), the geometric crossover based on the corresponding labeling-independent distance (5pt LI-GX), the geometric crossover based on the Hamming distance restricted to permutation with repetitions (Cycle H-GX), the geometric crossover based on the corresponding labeling-independent distance (Cycle LI-GX), and a crossover which uses an heuristic form of normalization that is known to be very good [73] (GEFM). Table 11.3 presents the correspondence between search spaces and associated geometric crossovers for graph partitioning presented in this chapter.

Table 11.3: Search spaces associated to geometric crossovers for graph partitioning.

crossover	search space
5pt H-GX	the space of all labeled partitions
5pt LI-GX	the space of all unlabeled partitions
Cycle H-GX	the space of all labeled well-balanced partitions
Cycle LI-GX	the space of all unlabeled well-balanced partitions

Table 11.4: The Results of 32-way Partitioning

Graph	5pt H-GX			GEFM[73]			5pt LI-GX		
	Best	Ave†	Gen‡(CPU*)	Best	Ave†	Gen‡(CPU*)	Best	Ave†	Gen‡(CPU*)
G500.2.5	182	185.18	1091(173.33)	178	181.83	1503(159.39)	178	181.77	1529(180.30)
G500.05	626	637.25	1424(330.64)	624	631.27	1974(359.25)	624	630.07	2367(334.80)
G500.10	1576	1587.23	1984(713.07)	1575	1582.42	2250(672.64)	*1573*	1581.40	2422(571.50)
G500.20	4040	4049.44	2247(1502.26)	4038	4045.41	2425(1402.75)	*4034*	4044.89	2522(1245.96)
U500.05	112	120.65	1327(319.04)	113	117.06	1592(314.26)	112	116.75	1599(331.95)
U500.10	534	542.75	1163(449.76)	529	537.81	1468(464.70)	531	537.04	1494(483.50)
U500.20	1837	1846.30	1123(814.94)	1829	1841.68	1327(715.80)	1832	1841.02	1353(747.04)
U500.40	5363	5389.93	1043(1399.31)	5355	5380.19	1353(1410.09)	5353	5380.30	1374(1398.19)

† Average over 100 runs.
‡ Average number of generations for each run.
∗ CPU seconds on Pentium III 1GHz.

Tables 11.4 and 11.5 show the results of 32-way partitioning. For each problem instance, boldface highlights the best average performance among the five crossovers, and italics highlights the best top performance. On random graphs (G500 class), Cycle H-GX does not dominate 5pt H-GX on average, but it performed better in terms of best solution found. On random geometric graphs (U500 class), Cycle H-GX performed better than 5pt H-GX both on average and in terms of best solution found. Cycle LI-GX and 5pt LI-GX always outperformed Cycle H-GX and 5pt

Table 11.5: The Results of 32-way Partitioning

Graph	Best	Cycle H-GX			Cycle LI-GX		
	Known	Best	Ave[†]	Gen[‡](CPU[*])	Best	Ave[†]	Gen[‡](CPU[*])
G500.2.5	178	*177*	185.59	1269(154.60)	*177*	**180.50**	1582(204.50)
G500.05	624	627	637.58	1660(282.69)	*623*	**628.63**	2232(378.46)
G500.10	1574	1575	1587.38	1987(537.10)	*1573*	**1580.53**	2381(641.58)
G500.20	4037	4039	4049.65	2198(1270.58)	4035	**4043.29**	2538(1354.42)
U500.05	113	113	120.41	1245(300.42)	*109*	**112.60**	2056(371.45)
U500.10	529	524	539.67	1263(472.37)	*523*	**528.50**	2086(583.02)
U500.20	1825	1834	1843.80	1170(790.02)	*1825*	**1831.55**	1646(844.36)
U500.40	5328	5372	5391.77	999(1314.44)	5348	**5365.00**	1691(1515.15)

† Average over 100 runs.

‡ Average number of generations for each run.

∗ CPU seconds on Pentium III 1GHz.

Table 11.6: The Results of 128-way Partitioning

Graph	5pt H-GX			GEFM[73]			5pt LI-GX		
	Best	Ave[†]	Gen[‡](CPU[*])	Best	Ave[†]	Gen[‡](CPU[*])	Best	Ave[†]	Gen[‡](CPU[*])
G500.2.5	316	320.14	844(535.13)	318	320.08	853(513.96)	*310*	314.08	950(759.72)
G500.05	850	853.09	869(688.03)	847	852.38	936(684.05)	*839*	843.61	1020(947.68)
G500.10	1904	1907.78	932(1300.94)	1901	1906.71	996(1224.89)	1894	1898.03	1168(1316.21)
G500.20	4568	4571.93	965(2333.56)	4565	4570.48	1028(2187.80)	*4560*	4566.40	1116(2261.33)
U500.05	697	704.06	935(910.83)	698	703.30	971(830.69)	695	702.90	978(1227.62)
U500.10	1679	1684.13	913(1302.62)	1676	1683.92	925(1223.74)	1676	1683.47	921(1618.17)
U500.20	3836	3841.45	890(1437.07)	3836	3841.57	895(1378.42)	3836	3841.44	874(1790.69)
U500.40	8066	8068.54	853(1971.41)	8066	8068.71	850(1847.12)	*8065*	8068.77	831(2250.17)

† Average over 100 runs.

‡ Average number of generations for each run.

∗ CPU seconds on Pentium III 1GHz.

H-GX. Cycle LI-GX showed better performance than 5pt LI-GX. Except on U500.40, Cycle LI-GX found solutions better than or equal to the best known. LI-GX was better than GEFM on average and in terms of best solution found. The algorithms stopped after a similar number of generations.

Tables 11.6 and 11.7 show the results of 128-way partitioning. As for 32-way partitioning, Cycle LI-GX performed best. Except on G500.05 and U500.10, it found the best solutions. The performance of Cycle H-GX was better than in the case of 32-way partitioning. On random geometric graphs, it outperformed 5pt LI-GX. But on random graphs, 5pt LI-GX performed better. 5pt H-GX was always dominated by others. LI-GX was better than 5pt H-GX and

Table 11.7: The Results of 128-way Partitioning

Graph	Cycle H-GX			Cycle LI-GX		
	Best	Ave†	Gen‡(CPU*)	Best	Ave†	Gen‡(CPU*)
G500.2.5	311	314.54	911(319.23)	*310*	**313.05**	964(694.41)
G500.05	*839*	844.13	977(463.58)	840	**843.19**	1058(876.97)
G500.10	1896	1899.32	1033(766.90)	*1893*	**1896.71**	1191(1200.03)
G500.20	4564	4567.94	1004(1774.22)	*4560*	**4565.06**	1164(2149.44)
U500.05	695	701.40	941(684.25)	*692*	**698.96**	1219(1480.44)
U500.10	*1673*	1682.74	923(1095.62)	1675	**1681.55**	988(1656.25)
U500.20	3838	**3840.37**	864(1186.80)	*3835*	3841.15	887(1799.27)
U500.40	*8065*	**8067.74**	845(1493.73)	*8065*	8068.42	832(2115.29)

† Average over 100 runs.

‡ Average number of generations for each run.

∗ CPU seconds on Pentium III 1GHz.

GEFM both on the average and on the best.

Among all new crossovers, we got clear improvements on the best-known solutions for all the tested instances in terms of the quality of solutions achieved. In particular, for 32-way partitioning on random geometric graphs, there was a large improvement.

Let us now turn to the computing time. For small K, normalization by the Hungarian method affects computational time little. In 32-way partitioning, Cycle LI-GX was about 1.2 times slower than Cycle H-GX. But, normalization time increases as K increases. In fact, Cycle LI-GX was about 1.7 times slower than Cycle H-GX in 128-way partitioning. Cycle crossovers were faster than 5-point crossovers. In summary, Cycle H-GX and Cycle LI-GX were faster than 5pt H-GX and 5pt LI-GX, respectively. Consequently, Cycle H-GX was fastest among them.

11.6.4 Geometricity

From uncountable experiments with many neighbourhood search meta-heuristics, it is well-known that, as a rule of thumb, the choice of a good neighbourhood structure for the problem at hand has a major impact on the performance, other algorithmic details being of secondary importance [122]. This principle can be extended to geometric crossover by generalizing the neighbourhood structure to the distance associated with the search space (see chapter 3). In this section, we test this principle on the case of graph partitioning.

Let LI be the labeling-independent metric, let p_1, p_2 be the parents, c be the offspring after crossover, and o be the offspring after mutation (swap mutation or balance adjustment). A measure of *non-geometricity* of a crossover operator is its average degree of distortion from the shortest path, which can be defined as the average value of

$$(LI(p_1, c) + LI(c, p_2) - LI(p_1, p_2))/LI(p_1, p_2)$$

or

$$(LI(p_1, o) + LI(o, p_2) - LI(p_1, p_2))/LI(p_1, p_2).$$

Tables 11.8 and 11.9 report the measure of non-geometricity of the crossovers and problem instances considered in the previous section. From Theorem 11.3.5, the values in the "After xover" columns in 5pt LI-GX are all zero. Note also that the values in the "After mutation" columns in Cycle LI-GX were the smallest.

Table 11.8: Non-geometricity of crossovers on 32-way Partitioning

Graph	5pt H-GX		GEFM		5pt LI-GX		Cycle H-GX		Cycle LI-GX	
	After xover^{1}	After mut^{2,†}	After xover^{1}	After mut^{2,†}	After xover^{1}	After mut^{2,†}	After xover^{2}	After mut^{2,‡}	After xover^{2}	After mut^{2,‡}
G500.2.5	0.56	0.69	0.01	0.14	0.00	0.12	0.51	0.53	0.01	0.03
G500.05	0.42	0.53	0.01	0.14	0.00	0.13	0.40	0.42	0.01	0.04
G500.10	0.42	0.53	0.01	0.13	0.00	0.11	0.41	0.43	0.01	0.04
G500.20	0.39	0.49	0.01	0.12	0.00	0.12	0.40	0.42	0.01	0.03
U500.05	1.07	1.25	0.01	0.15	0.00	0.14	1.02	1.05	0.01	0.06
U500.10	1.49	1.72	0.01	0.15	0.00	0.15	1.82	1.87	0.01	0.12
U500.20	1.90	2.18	0.01	0.15	0.00	0.14	1.53	1.57	0.01	0.09
U500.40	0.96	1.12	0.01	0.15	0.00	0.14	1.03	1.06	0.01	0.06

1 Not balanced (not feasible) solution. 2 Balanced (feasible) solution.
† Balance adjustment. ‡ Swap mutation (1%).
Average over 100 runs.

From a geometrical viewpoint, it is not surprising that 5pt LI-GX and GEFM performed similarly, even if 5pt LI-GX was better, because after all GEFM is almost geometric under the distance 5pt LI-GX is based upon. From the landscape analysis we know that this is a very meaningful distance for the problem at hand. So, our results corroborate the rule-of-thumb that given a good distance, one gets a good geometric crossover, other details (such as the exact

Table 11.9: Non-geometricity of crossovers on 128-way Partitioning

Graph	5pt H-GX		GEFM		5pt LI-GX		Cycle H-GX		Cycle LI-GX	
	After xover[1]	After mut[2,†]	After xover[1]	After mut[2,†]	After xover[1]	After mut[2,†]	After xover[2]	After mut[2,‡]	After xover[2]	After mut[2,‡]
G500.2.5	0.75	0.99	0.03	0.25	0.00	0.21	0.70	0.72	0.02	0.05
G500.05	0.69	0.91	0.03	0.24	0.00	0.22	0.59	0.61	0.02	0.05
G500.10	0.62	0.83	0.03	0.24	0.00	0.21	0.56	0.58	0.02	0.05
G500.20	0.55	0.75	0.03	0.23	0.00	0.21	0.51	0.53	0.02	0.04
U500.05	1.14	1.45	0.03	0.25	0.00	0.22	0.92	0.95	0.02	0.07
U500.10	0.83	1.07	0.03	0.25	0.00	0.22	0.69	0.71	0.02	0.05
U500.20	0.55	0.74	0.03	0.23	0.00	0.21	0.49	0.51	0.02	0.04
U500.40	0.42	0.60	0.03	0.22	0.00	0.19	0.39	0.41	0.01	0.03

1 Not balanced (not feasible) solution. 2 Balanced (feasible) solution.

† Balance adjustment. ‡ Swap mutation (1%).

Average over 100 runs.

probability distribution of the search operator) being of secondary importance. The argument that geometricity is what really counts is reinforced by the fact (see Table 11.9) that GEFM was a worse approximation of geometric crossover for 128-way partitioning (0.03 non-geometric) and this was mirrored in Table 11.6 by a bigger difference between the average results of GEFM and 5pt LI-GX. So, when GEFM was less geometric, this was immediately reflected in a loss of performance.

11.7 Summary

The multiway graph partitioning problem presents two challenges for crossover: feasibility and redundancy. In this chapter we have addressed these issues and made the following contributions:

- We have adapted the generalization of cycle crossover presented in chapter 5 to the graph partitioning problem (Cycle H-GX) and showed that it naturally suits partition problems and addresses the feasibility problem.

- Also, we have shown that the important notion of normalization before recombination, that is very effective on problems with redundant encodings, can be naturally cast in geometric terms using a distance that filters the redundancy of the encoding together with the formal definition of geometric crossover.

- This geometric point of view on normalization has allowed us to study its effect on the fitness landscape and explain, within the geometric framework, why normalization before recombination for redundant encodings is a good idea.
- The landscape analysis also provided evidence for the fact that the labeling-independent distance is more suitable for the solution space of multiway graph partitioning problem than the Hamming distance.
- We have designed another geometric crossover (5pt LI-GX) based on the labeling-independent distance that addresses the redundancy problem.
- We have then combined these two crossovers obtaining a new, much superior geometric crossover (Cycle LI-GX) that suits partition problems with redundant encodings. In experimentation, we demonstrated that this crossover outperforms previously known methods, either providing new solutions or equalling known best solutions in graph partitioning benchmark problems.

This chapter shows that the theory of geometric crossover is not just theory for its own sake. Indeed it can be put at work in practice and produce excellent results for hard combinatorial optimization problems such as the multi-way graph partitioning problem. Redundancy and feasibility are important issues arising in practice in many problems. Geometric crossover can address them in a natural way within its general framework.

Chapter 12

Geometric PSO

Using a geometric framework for the interpretation of crossover of recent introduction, we show an intimate connection between particle swarm optimization (PSO) and evolutionary algorithms. This connection enables us to generalize PSO to virtually any solution representation in a natural and straightforward way. The new PSO (GPSO) applies naturally to both continuous and combinatorial spaces. We demonstrate this for the cases of Euclidean, Manhattan and Hamming spaces and report extensive experimental results. We also demonstrate the applicability of GPSO to non-trivial combinatorial spaces. We apply GPSO to solve the Sudoku puzzle.

12.1 Introduction

Particle swarm optimisation [70] has traditionally been applied to continuous search spaces. Although a version of PSO for binary search spaces has been defined [69], attempts to extend PSO to richer spaces, such as, for example, combinatorial spaces, have had no real success [17].

There are two ways of extending PSO to richer spaces: a) adapting the PSO for each new solution representation, or b) making use of a rigorous mathematical generalisation to a general class of spaces of the notion (and motion) of particles. This second approach has the advantage that a PSO can be derived in a principled way for any search space belonging to the given class. Here we follow this approach. In particular, we show *formally* how a general form of PSO (without the momentum term) can be obtained by using theoretical tools developed for evolutionary algorithms using geometric crossover and geometric mutation.

Firstly, we formally derive Geometric PSOs (GPSOs) for Euclidean, Manhattan and Hamming spaces and discuss how to derive GPSOs for virtually any representation in a similar way. Then, we test the GPSO theory experimentally: we implement the specific GPSO for Euclidean, Manhattan and Hamming spaces and report extensive experimental results showing that GPSOs perform very well.

Finally, we also demonstrate that GPSO can be specialised easily to non-trivial combinatorial spaces. In chapter 9 we have used the geometric framework to design an evolutionary algorithm to solve the Sudoku puzzle and obtained very good experimental results. In this chapter, we apply GPSO to solve the Sudoku puzzle.

In Section 12.2, we introduce the notion of multi-parental geometric crossover. In Section 12.3, we recast PSO in geometric terms and generalize it to generic metric spaces. In Section 12.4, we apply these notions to the Euclidean, Manhattan and Hamming spaces. In Section 12.5, we discuss how to specialise the general PSO automatically to virtually any solution representation using geometric crossover. Then, in section 12.6, we report experimental results with the GPSOs for Euclidean, Manhattan and Hamming spaces and we compare them with a traditional PSO. In section 12.7, we apply GPSO to Sudoku and in section 12.8, we report experimental results.

12.2 Multi-parental geometric crossover

To extend geometric crossover to the case of multiple parents we need the following definitions [154].

Definition 12.2.1. A family $\mathcal{X}$ of subsets of a set X is called *convexity on* X if:

(C1) the empty set $\emptyset$ and the universal set X are in $\mathcal{X}$,

(C2) if $\mathcal{D} \subseteq \mathcal{X}$ is non-empty, then $\bigcap \mathcal{D} \in \mathcal{X}$, and

(C3) if $\mathcal{D} \subseteq \mathcal{X}$ is non-empty and totally ordered by inclusion, then $\bigcup \mathcal{D} \in \mathcal{X}$.

The pair $(X, \mathcal{X})$ is called *convex structure*. The members of $\mathcal{X}$ are called *convex sets*. By the axiom (C1) a subset A of X of the convex structure is included in at least one convex set, namely X.

From axiom (C2), A is included in a smallest convex set, the *convex hull* of A:

$$co(A) = \bigcap\{C|A \subseteq C \in \mathcal{X}\}.$$

The convex hull of a finite set is called a *polytope*.

The axiom (C3) requires *domain finiteness* of the convex hull operator: a set C is convex iff it includes $co(F)$ for each finite subset F of C.

The convex hull operator applied to set of cardinality two is called *segment operator*. Given a metric space $M = (X, d)$ the segment between a and b is the set $[a, b]_d = \{z \in X|d(x, z) + d(z, y) = d(x, y)\}$. The abstract *geodetic convexity* $\mathcal{C}$ on X induced by M is obtained as follow: a subset C of X is geodetically-convex provided $[x, y]_d \subseteq C$ for all x, y in C. If co denotes the convex hull operator of $\mathcal{C}$, then $\forall a, b \in X : [a, b]_d \subseteq co\{a, b\}$. The two operators need not to be equal: there are metric spaces in which metric segments are not all convex.

We can now provide the following extension:

Definition 12.2.2. (Multi-parental geometric crossover) In a multi-parental geometric crossover, given n parents $p_1, p_2, \ldots, p_n$ their offspring are contained in the metric convex hull of the parents $co(\{p_1, p_2, \ldots, p_n\})$ for some metric d.

Theorem 12.2.1. *(Decomposable three-parent recombination) Every multi-parental recombination $RX(p_1, p_2, p_3)$ that can be decomposed as a sequence of 2-parental geometric crossovers under the same metric GX and GX', so that $RX(p_1, p_2, p_3) = GX(GX'(p_1, p_2), p_3)$, is a three-parental geometric crossover.*

Proof. Let P be the set of parents and $co(P)$ their metric convex hull. By definition of metric convex hull, for any two points $a, b \in co(P)$ their offspring are in the convex hull $[a, b] \subseteq co(P)$. Since $P \subseteq co(P)$, any two parents $p_1, p_2 \in P$ have offspring $o_{12} \in co(P)$. Then any other parent $p_3 \in P$, when recombined with o_{12}, produces offspring o_{123} in the convex hull $co(P)$. So the three-parental recombination equivalent to the sequence of geometric crossover $GX'(p_1, p_2) \rightarrow o_{12}$ and $GX(o_{12}, p_3) \rightarrow o_{123}$ is a multi-parental geometric crossover. □

12.3 Geometric PSO

12.3.1 Canonical PSO algorithm and geometric crossover

Consider the canonical PSO in Algorithm 7. It is well known [18] that one can write the equation of motion of the particle without making explicit use of its velocity.

Let x be the position of a particle and v be its velocity. Let $\hat{x}$ be the current best position of the particle and let $\hat{g}$ be the global best. Let v and v' be the velocity of the particle and $x' = x + v$ and $x'' = x' + v'$ its position at the next two time ticks. The equation of velocity update is the linear combination: $v' = \omega v + \phi_1(\hat{x} - x') + \phi_2(\hat{g} - x')$ where ω, ϕ_1 and ϕ_2 are scalar coefficients. To eliminate velocities, we substitute the identities $v = x' - x$ and $v' = x'' - x'$ in the equation of velocity update and rearrange it to obtain an equation that expresses x'' as function of x and x':

$$x'' = (1 + \omega - \phi_1 - \phi_2)x' - \omega x + \phi_1 \hat{x} + \phi_2 \hat{g}$$

If we set $\omega = 0$ (i.e., the particle has no inertia), x'' becomes independent on its position x two time ticks earlier. If we call $w_1 = 1 - \phi_1 - \phi_2$, $w_2 = \phi_1$, and $w_3 = \phi_2$, the equation of motion becomes:

$$x'' = w_1 x' + w_2 \hat{x} + w_3 \hat{g}. \tag{12.3.1}$$

In these conditions, the main feature that allows the motion of particles is the ability to perform linear combinations of points in the search space. As we will see in the next section, we can achieve this same ability by using multiple (geometric) crossover operations. This makes it possible to obtain a generalisation of PSO to generic search spaces.

In the following, we illustrate the parallel between an evolutionary algorithm with geometric crossover and the motion of a particle in PSO (see Figure 12.1). Geometric crossover picks offspring C on a line segment between parents A and B. Geometric crossover can be interpreted as a motion of a particle: consider a particle P that moves in the direction of a point D reaching, in the next time step, position P'. If one equates parent A with the particle P and parent B with the direction point D, the offspring C is, therefore, the particle at the next time step P'. The distance between parent A and offspring C is the magnitude of the velocity of the particle P. Notice that the particle moves from P to P': this means that the particle P is replaced by the particle P' in the next time step. In other words, the new position of the particle replaces the previous position. Coming back to the evolutionary algorithm with geometric crossover, this means that the offspring C replaces its parent A in the new population. The fact that at

Algorithm 7 Standard PSO algorithm.

```
1: for all particle $i$ do
2:     initialise position $x_i$ and velocity $v_i$
3: end for
4: while stop criteria not met do
5:     for all particle $i$ do
6:         set personal best $\hat{x}_i$ as best position found so far by the particle
7:         set global best $\hat{g}$ as best position found so far by the whole swarm
8:     end for
9:     for all particle $i$ do
10:        update velocity using equation
               $$v_i(t+1) = \kappa\left(\omega v_i(t) + \phi_1 U(0,1)(\hat{g}(t) - x_i(t)) + \phi_2 U(0,1)(\hat{x}_i(t) - x_i(t))\right)$$
           where, typically, either ($\kappa = 0.729$, $\omega = 1.0$), or ($\kappa = 1.0$, $\omega < 1$)
11:        update position using equation
               $$x_i(t+1) = x_i(t) + v_i(t+1)$$
12:    end for
13: end while
```

a given time all particles move is equivalent to say that each particle is selected for "mating". Mating is a weighted multi-recombination involving the memory of the particle and the best in the current population.

In the standard PSO, weights represent the propensity of a particle towards memory, sociality and cognition. In the GPSO, they represent the attractions towards the particle's previous position, the particle's best position and the swarm's best position.

Naturally, particle motion based on geometric crossover leads to a form of search that cannot extend beyond the convex hull of the initial population. Mutation can be used to allow non-convex search. We explain these ideas in detail in the following sections.

12.3.2 Geometric interpretation of linear combinations

Definition 12.3.1. If $v_1, ..., v_n$ are vectors and $a_1, ..., a_n$ are scalars, then the *linear combination* of those vectors with those scalars as coefficients is : $a_1v_1 + a_2v_2 + a_3v_3 + \cdots + a_nv_n$.

A linear combination on n linearly independent vectors spans completely an n-dimensional space but not a higher dimensional one (e.g., the linear combination of three linearly independent

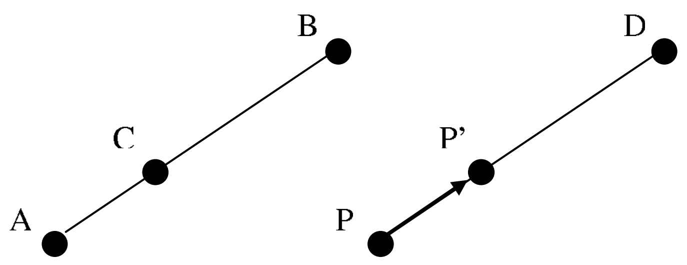


Figure 12.1: Geometric crossover and particle motion.

points spans a 3-dimensional space but not a 4-dimensional one).

Definition 12.3.2. An *affine combination* of vectors $v_1, ..., v_n$ is a linear combination $\sum_i a_i \cdot x_i$ in which $\sum_i a_i = 1$.

When a vector represents a point in space, the affine combination of 2 independent points spans completely the line passing through them; the affine combination of 3 points spans completely the plane (2D line) passing through them (note that some a_i can be negative). An increasing number of linearly independent points spans completely higher dimensional "lines".

Definition 12.3.3. A *convex combination* is an affine combination of vectors where all coefficients are non-negative.

It is called "convex combination" because when vectors represent points in space, the set of all convex combinations constitutes the convex hull.

A special case is $n = 2$, where a point formed by the convex combination will lie on a straight line between two points. For three points, their convex hull is the triangle with the points as vertices.

Theorem 12.3.1. *In a PSO with no inertia ($\omega = 0$) and where acceleration coefficients are such that $\phi_1 + \phi_2 \leq 1$, the next position x' of a particle is within the convex hull formed by its current position x, its local best $\hat{x}$ and the swarm best $\hat{g}$.*

Proof. As we have seen in Section 12.3.1, when $\omega = 0$, a particle's update equation becomes the linear combination in equation (12.3.1). Notice that this is an affine combination since the coefficients of x', $\hat{x}$ and $\hat{g}$ add up to 1. Interestingly, this means that the new position of the particle is coplanar with x', $\hat{x}$ and $\hat{g}$. If we restrict w_2 and w_3 to be positive and their sum to be less than 1, equation (12.3.1) becomes a convex combination. *Geometrically this means that the new position of the particle is in the convex hull formed by (or more informally, is between) its previous position, its local best and the swarm best.* $\square$

In the next section, we generalize this simplified form of PSO[1] from real vectors to generic metric spaces. As mentioned above, mutation will be required to extend the search beyond the convex hull.

12.3.3 Convex combinations in metric spaces

Linear combinations are well-defined for vector spaces: algebraic structures endowed with scalar product and vectorial sum. A metric space is a set endowed with a notion of distance. The set underlying a metric space does not normally come with well-defined notions of scalar product and sum among its elements. Therefore, a linear combination of its elements is not defined. How can we then define a convex combination in a metric space? Vectors in a vector space can easily be understood as points in a metric space. However, the interpretation of scalars is not as straightforward: what do the scalar weights in a convex combination mean in a metric space?

As seen in Section 12.3.2, a convex combination is an algebraic description of a convex hull. However, even if the notion of convex combination is not defined for metric spaces, convexity in metric spaces is still well-defined through the notion of metric convex set that is a straightforward generalization of traditional convex set. Since convexity is well-defined for metric spaces, we still have hope to generalize the scalar weights of a convex combination trying to make sense of them in terms of distance.

The weight of a point in a convex combination can be seen as a measure of relative linear "attraction" toward its corresponding point, versus attractions toward the other points of the combination. The closer a weight to 1, the stronger the attraction to the corresponding point. The point resulting from a convex combination can be seen as the equilibrium point of all the attraction forces. The distance between the equilibrium point and a point of the convex combination is, therefore, a decreasing function of the level of attraction (weight) of the point: the stronger the attraction, the smaller its distance to the equilibrium point. This observation can be used to reinterpret the weights of a convex combination in a metric space as follows:

[1]The simplified PSO presented here relies on a condition which makes it differ from the traditional PSO in which, usually, $\phi_1 = \phi_2 = 2$. In future work, we will present a generalization of the traditional PSO that will not require any restriction on the parameters used.

$y = w_1x_1+w_2x_2+w_3x_3$ with w_1, w_2 and w_3 greater than zero and $w_1+w_2+w_3 = 1$ is generalized to mean that y is a point such that $d(x_1, y) = f(w_1)$, $d(x_2, y) = f(w_2)$ and $d(x_3, y) = f(w_3)$ where f is a decreasing function.

This definition is formal and valid for all metric spaces, but it is non-constructive. In contrast, a convex combination not only defines a convex hull, but it tells also how to reach all its points. So, how can we actually pick a point in the convex hull respecting the above distance requirements? Geometric crossover will help us with this, as we show in the next section.

To summarise, the requirements for a convex combination in a metric space are:

1. *Convex weights*: the weights respect the form of a convex combination: $w_1, w_2, w_3 > 0$ and $w_1 + w_2 + w_3 = 1$.

2. *Convexity*: the convex combination operator combines x_1, x_2 and x_3 and returns a point in their metric convex hull (or simply triangle) under the metric of the space considered.

3. *Coherence between weights and distances*: the distances to the equilibrium point are decreasing functions of their weights.

4. *Symmetry*: the same value assigned to w_1, w_2 or w_3 has the same effect (e.g., in a equilateral triangle, if the coefficients have all the same value, the distances to the equilibrium point are the same).

12.3.4 Geometric PSO algorithm

The generic GPSO algorithm is illustrated in Algorithm 8. This differs from the standard PSO (Algorithm 7) in that:

- there is no velocity,
- the equation of position update is the convex combination,
- there is mutation, and
- the parameters w_1, w_2, and w_3 are non-negative and add up to one.

The specific PSOs for the Euclidean, Manhattan and Hamming spaces use the randomized convex combination operators described in Section 12.4 and space-specific mutations.

Algorithm 8 Geometric PSO algorithm.

```
1: for all particle $i$ do
2:   initialise position $x_i$ at random in the search space
3: end for
4: while stop criteria not met do
5:   for all particle $i$ do
6:     set personal best $\hat{x}_i$ as best position found so far by the particle
7:     set global best $\hat{g}$ as best position found so far by the whole swarm
8:   end for
9:   for all particle $i$ do
10:    update position using a randomized convex combination
           $x_i = CX((x_i, w_1), (\hat{g}, w_2), (\hat{x}_i, w_3))$    (12.3.2)
11:    mutate $x_i$
12:  end for
13: end while
```

12.4 Geometric PSO for specific spaces

12.4.1 Euclidean space

The GPSO for the Euclidean space is *not* an extension of the traditional PSO. We include it to show how the general notions introduced in the previous section materialize in a familiar context. The convex combination operator for the Euclidean space is the traditional convex combination that produces points in the traditional convex hull.

In Section 12.3.3, we have mentioned how to interpret the weights in a convex combination in terms of distances. In the following we show analytically how the weights of a convex combination affect the relative distances to the equilibrium point. In particular, we show that the relative distances are decreasing functions of the corresponding weights.

Theorem 12.4.1. *In a convex combination, the distances to the equilibrium point are decreasing functions of the corresponding weights.*

Proof. Let a, b and c be three points in $\mathbb{R}^n$ and $x = w_a a + w_b b + w_c c$ be a convex combination. Let us now decrease w_a to $w'_a = w_a - \Delta$ such that w'_a, w'_b and w'_c still form a convex combination and that the relative proportions of w_b and w_c remain unchanged: $\frac{w'_b}{w'_c} = \frac{w_b}{w_c}$. This requires w'_b and w'_c to be $w'_b = w_b(1 + \Delta/(w_b + w_c))$ and $w'_c = w_c(1 + \Delta/(w_b + w_c))$. The equilibrium point for the new convex combination is

$$x' = (w_a - \Delta)a + w_b(1 + \Delta/(w_b + w_c))b + w_c(1 + \Delta/(w_b + w_c))c$$

The distance between a and x is

$$|a - x| = |w_b(a - b) + w_c(a - c)|$$

and the distance between a and the new equilibrium point is

$$\begin{aligned} |a - x'| &= |w_b(1 + \Delta/(w_b + w_c))(a - b) + w_c(1 + \Delta/(w_b + w_c))(a - c)| \\ &= (1 + \Delta/(w_b + w_c))|a - x| \end{aligned}$$

So when w_a decreases ($\Delta > 0$) and w_b and w_c maintain the same relative proportions, the distance between the point a and the equilibrium point x increases ($|a - x'| > |a - x|$). Hence the distance between a and the equilibrium point is a decreasing function of w_a. For symmetry this applies to the distances between b and c and the equilibrium point: they are decreasing functions of their corresponding weights w_b and w_c, respectively. □

The traditional convex combination in the Euclidean space respects the four requirements for a convex combination presented in Section 12.3.3.

12.4.2 Manhattan space

In the following we first define a multi-parental recombination for the Manhattan space and then prove that it respects the four requirements for being a convex combination presented in Section 12.3.3.

Definition 12.4.1. (Box recombination family) Given two parents a and b in $\mathbb{R}^n$, a box recombination operator returns offspring o such that $o_i \in [min(a_i, b_i), max(a_i, b_i)]$ for $i = 1 \ldots n$.

Theorem 12.4.2. *(Geometricity of box recombination) Any box recombination is a geometric crossover under Manhattan distance.*

Proof. Theorem 12.4.2 is an immediate consequence of the product geometric crossover theorem (see chapter 9). □

Definition 12.4.2. (Three-parent Box recombination family) Given three parents a, b and c in $\mathbb{R}^n$, a box recombination operator returns offspring o such that $o_i \in [min(a_i, b_i, c_i), max(a_i, b_i, c_i)]$ for $i = 1 \ldots n$.

Theorem 12.4.3. *(Geometricity of three-parent box recombination) Any three-parent box recombination is a geometric crossover under Manhattan distance.*

Proof. We prove it by showing that any multi-parent box recombination $BX(a, b, c)$ can be decomposed as a sequence of two simple box recombinations. Since the simple box recombination is geometric (Theorem 12.4.2), this theorem is a simple corollary of the multi-parental geometric decomposition theorem (Theorem 12.2.1).

We will show that $o' = BX(a, b)$ followed by $BX(o', c)$ can reach any offspring $o = BX(a, b, c)$. For each i we have $o_i \in [\min(a_i, b_i, c_i), \max(a_i, b_i, c_i)]$. Notice that $[\min(a_i, b_i), \max(a_i, b_i)] \cup [\min(a_i, c_i), \max(a_i, c_i)] = [\min(a_i, b_i, c_i), \max(a_i, b_i, c_i)]$. We have two cases: (i) $o_i \in [\min(a_i, b_i), \max(a_i, b_i)]$ in which case o_i is reachable by the sequence $BX(a, b)_i \to o_i, BX(o, c)_i \to o_i$; (ii) $o_i \notin [\min(a_i, b_i), \max(a_i, b_i)]$ then it must be in $[\min(a_i, c_i), \max(a_i, c_i)]$ in which case o_i is reachable by the sequence $BX(a, b)_i \to a_i, BX(a, c)_i \to o_i$. □

Definition 12.4.3. (Weighted multi-parent Box recombination) Given three parents a, b and c in $\mathbb{R}^n$ and weights w_a, w_b and w_c, a weighted box recombination operator returns offspring o such that $o_i = w_{a_i} a_i + w_{b_i} b_i + w_{c_i} c_i$ for $i = 1 \ldots n$, where w_{a_i}, w_{b_i} and w_{c_i} are a convex combination of randomly perturbed weights with expected values w_a, w_b and w_c

The difference between box recombination and linear recombination (Euclidean space) is that in the latter the weights w_a, w_b and w_c are randomly perturbed only once and the same weights are used for all the dimensions, whereas the former one has a different randomly perturbed version of the weights for each dimension.

The weighted multi-parent box recombination belongs to the family of multi-parent box recombination because $o_i = w_{a_i} a_i + w_{b_i} b_i + w_{c_i} c_i \in [\min(a_i, b_i, c_i), \max(a_i, b_i, c_i)]$ for $i = 1 \ldots n$, hence it is geometric.

Theorem 12.4.4. *(Coherence between weights and distances) In weighted multi-parent box recombination, the distances of the parents to the expected offspring are decreasing functions of the corresponding weights.*

Proof. The proof of Theorem 12.4.4 is a simple variation of that of Theorem 12.4.1. □

In summary, in this section we have introduced the weighted multi-parent box recombination and shown that it is a convex combination operator satisfying the four requirements of a metric convex combination for the Manhattan space: convex weights (Definition 12.4.2), convexity (geometricity, Theorem 12.4.3), coherence (Theorem 12.4.4) and symmetry (self-evident).

12.4.3 Hamming space

In this section, we first define a multi-parental recombination for binary strings that is a straightforward generalization of mask-based crossover with two parents and then prove that it respects the four requirements for being a convex combination in the Hamming space presented in Section 12.3.3.

Definition 12.4.4. (Three-parent mask-based crossover family) Given three parents a, b and c in $\{0,1\}^n$, generate randomly a crossover mask of length n with symbols from the alphabet $\{\texttt{a},\texttt{b},\texttt{c}\}$. Build the offspring o filling each position with the bit from the parent appearing in the crossover mask at the corresponding position.

The weights w_a, w_b and w_c of the convex combination indicate, for each position in the crossover mask, the probability of having the symbols `a`, `b` or `c`.

Theorem 12.4.5. *(Geometricity of three-parent mask-based crossover) Any three-parent mask-based crossover is a geometric crossover under Hamming distance.*

Proof. We prove it by showing that any three-parent mask-based crossover can be decomposed as a sequence of two simple mask-based crossovers. Since simple mask-based crossover is geometric, this theorem is a simple corollary of the multi-parental geometric decomposition theorem (Theorem 12.2.1).

Let m_{abc} be the mask to recombine a, b and c, producing the offspring o. Let m_{ab} be the mask obtained by substituting all occurrences of `c` in m_{abc} with `b`, and m_{bc} be the mask obtained by substituting all occurrences of `a` in m_{abc} with `b`. First, recombine a and b using m_{ab} obtaining b'. Then, recombine b' and c using m_{bc} where the `b`'s in the mask stand for alleles in b'. The offspring produced by the second crossover is o, so the sequence of the two simple crossovers is equivalent to the three-parent crossover. This is because the first crossover passes to the offspring all genes it needs to take from a according to m_{abc} and the rest of the genes are all from b; the second crossover corrects those genes that should have been taken from parent c according to m_{abc} but were taken from b instead. □

Theorem 12.4.6. *(Coherence between weights and distances) In weighted three-parent mask-based crossover, the distances of the parents to the expected offspring are decreasing functions of the corresponding weights.*

Proof. We want to know the expected distance from parent p_1, p_2 and p_3 to their expected offspring o as a function of the weights w_1, w_2 and w_3. To do so, we first determine, for each position in the offspring, the probability of it being the same as p_1. Then from that we can easily compute the expected distance between p_1 and o. We have that

$$pr\{o = p_1\} = pr\{p_1 \to o\} + pr\{p_2 \to o\} \cdot pr\{p_1|p_2\} + pr\{p_3 \to o\} \cdot pr\{p_1|p_3\} \qquad (12.4.1)$$

where: $pr\{o = p_1\}$ is the probability of a bit of o at a certain position to be the same as the bit of p_1 at the same position; $pr\{p_1 \to o\}$, $pr\{p_2 \to o\}$ and $pr\{p_3 \to o\}$ are the probabilities that a bit in o is taken from parent p_1, p_2 and p_3, respectively (these coincide with the weights of the convex combination w_1, w_2 and w_3); $pr\{p_1|p_2\}$ and $pr\{p_1|p_3\}$ are the probabilities that a bit taken from p_2 or p_3 coincides with the one in p_1 at the same location. These last two probabilities equal the number of common bits in p_1 and p_2 (and p_1 and p_3) over the length of the strings n. So $pr\{p_1|p_2\} = 1 - H(p_1, p_2)/n$ and $pr\{p_1|p_3\} = 1 - H(p_1, p_3)/n$ where $H(\cdot, \cdot)$ is the Hamming distance. So equation (12.4.1) becomes

$$pr\{o = p_1\} = w_1 + w_2(1 - H(p_1, p_2)/n) + w_3(1 - H(p_1, p_3)/n).$$

Hence, the expected distance between the parent p_1 and the offspring o is: $E(H(p_1, o)) = n \cdot (1 - pr\{o = p_1\}) = w_2 H(p_1, p_2) + w_3 H(p_1, p_3)$.

Notice that this is a decreasing function of w_1 because increasing w_1 forces w_2 or w_3 to decrease since the sum of the weights is constant, hence $E(H(p_1, o))$ decreases. Analogously, $E(H(p_2, o))$ and $E(H(p_3, o))$ are decreasing functions of their weights w_2 and w_3, respectively. □

In summary, in this section we have introduced the weighted multi-parent mask-based crossover and shown that it is a convex combination operator satisfying the four requirements of a metric convex combination for the Hamming space: convex weights (Definition 12.4.4), convexity (geometricity, Theorem 12.4.5), coherence (Theorem 12.4.6) and symmetry (self-evident).

12.5 Geometric PSO for other representations

Before looking into how we can extend GPSO to other solution representations, we will discuss the relation between 3-parental geometric crossover and the symmetry requirement for a convex combination.

For each of the spaces considered in Section 12.4, we have first considered (or defined) a three-parental recombination and then proved that it is a three-parental geometric crossover by showing that it can actually be decomposed into two sequential applications of a geometric crossover for the specific space.

However, we could have skipped the *explicit definition* of a three-parental recombination altogether. In fact, to obtain the three-parental recombination, we could have used two sequential applications of a known two-parental geometric crossover for the specific space. This composition is indeed a three-parental recombination (it combines three parents) and it is decomposable by construction. Hence, it is a three-parental geometric crossover. This, indeed, would have been simpler than the route we took.

The reason we preferred to define explicitly a three-parental recombination is that the requirement of symmetry of the convex combination is true by construction: if the roles of any two parents are swapped by exchanging in the three-parental recombination both positions and the respective recombination weights, the resulting recombination operator is equivalent to the original operator.

The symmetry requirement becomes harder to enforce and prove for a three-parental geometric crossover obtained by two sequential applications of a two-parental geometric crossover. We illustrate this in the following.

Let us consider three parents a, b and c with positive weights w_a, w_b and w_c which add up to one. If we have a symmetric three-parental weighted geometric crossover ΔGX, the symmetry of the recombination is guaranteed by the symmetry of the operator. So, $\Delta GX((a, w_a), (b, w_b), (c, w_c))$ is equivalent to $\Delta GX((b, w_b), (a, w_a), (c, w_c))$. Hence, the requirement of symmetry on the weights of the convex combination holds. If we consider a three-parental recombination defined by using twice a two-parental genetic crossover GX we have:

$$\Delta GX((a, w_a), (b, w_b), (c, w_c)) = GX((GX((a, w'_a), (b, w'_b)), w_{ab}), (c, w'_c))$$

with the constraint that w'_a and w'_b are positive and add up to one and w_{ab} and w'_c are positive and add up to one. Notice the inherent asymmetry in this expression: the weights w'_a and w'_b are not directly comparable with w'_c because they are relative weights between a and b. Moreover, there is the extra weight w_{ab}. This asymmetry makes the requirement of symmetry problematic to meet: given the desired w_a, w_b and w_c, what values of w'_a, w'_b, w_{ab} and w'_c should we choose to obtain an equivalent symmetric 3-parental weighted recombination expressed as a sequence of two two-parental geometric crossovers?

For the Euclidean space, it is easy to answer to this question using simple algebra:

$$\Delta GX = w_a \cdot a + w_b \cdot b + w_c \cdot c = (w_a + w_b)\left(\frac{w_a}{w_a + w_b} \cdot a + \frac{w_b}{w_a + w_b} \cdot b\right) + w_c \cdot c$$

Since the convex combination of two points in the Euclidean space is $GX((x, w_x), (y, w_y)) =$

$w_x \cdot x + w_y \cdot y$ and $w_x, w_y > 0$ and $w_x + w_y = 1$ then

$$\begin{aligned}\Delta GX((a, w_a), (b, w_b), (c, w_c)) \; = \; & GX\,[\\ & \left(GX\left(\left(a, \frac{w_a}{w_a + w_b}\right), \left(b, \frac{w_b}{w_a + w_b}\right)\right), w_a + w_b\right), \\ & (c, w_c)]\end{aligned}$$

However, the question may be less straightforward to answer for other spaces, although we could use the equation above as a rule-of-thumb to map the weights of ΔGX and the weights in the sequential GX decomposition.

Where does this discussion leave us in relation to the extension of GPSO to other representations? We have seen that there are two alternative ways to produce a convex combination for a new representation: (i) explicitly define a symmetric three-parental recombination for the new representation and then prove its geometricity by showing that it is decomposable into a sequence of two two-parental geometric crossovers (*explicit definition*), or (ii) use twice the simple geometric crossover to produce a symmetric or nearly symmetric three-parental recombination (*implicit definition*). The second option is also very interesting because *it allows us to extended automatically GPSO to all representations we have geometric crossovers for* (such as permutations, GP trees, variable-length sequences, to mention a few) *and virtually to any other complex solution representation.*

12.6 Experimental results for Euclidean, Manhattan and Hamming spaces

We have run two groups of experiments: one for the continuous version of the GPSO (`EuclideanPSO`, or EPSO for short, and `ManhattanPSO`, or MPSO), and one for the binary version (`HammingPSO`, or HPSO). In the following, we report only a brief summary of the experimental results. For more detail see [31].

For the Euclidean and Manhattan versions, we have compared the performances with those of a standard continuous PSO (`BasePSO`, or BPSO) with constriction. We have run the experiments

Table 12.1: Parameters for BPSO.

Population Size	20, 50 particles
Stop Condition	200 iterations
$V_{max} = X_{max}$	MAX_VALUE - MIN_VALUE
κ	0.729
ω	1.0
$\phi_1 = \phi_2$	2.05

for dimensions 2, 10 and 30 on the following five benchmark functions: *F1C - Sphere*, *F2C - Rosenbrock*, *F3C - Ackley*, *F4C - Griewangk* and *F5C - Rastrigin*. The Hamming version has been tested on the De Jong's test suite: *F1 - Sphere (30)*, *F2 - Rosenbrock (24)*, *F3 - Step (50)*, *F4 - Quartic (240)* and *F5 - Shekel (34)*, where the numbers in brackets are the dimensions of the problems in bits.

For the standard continuos version (BPSO), we have used the standard PSO parameter set (Table 12.1). Since there is no equivalent GPSO with $\phi_1 = \phi_2 = 2.05$ ($\phi_1 + \phi_2 > 1$, which does not respect the conditions in Theorem 12.3.1), we have decided to set w_1, w_2 and w_3 proportional to ω, ϕ_1 and ϕ_2, respectively, and summing up to one (see Table 12.2).

For the binary version, the parameters of population size, number of iterations and w_1, w_2 and w_3 have been tuned on the sphere function and are as in Table 12.3. From the parameters tuning, it appears that there is a preference for values of w_1 close to zero. This means that there is a bias towards the swarm and particle bests, and less attraction towards the current particle position.

For each set up, we performed 20 independent runs. Table 12.4 shows the best and the mean fitness value (i.e., the fitness value at the position where the population converges) found by the swarm when exploring continuous spaces. This table summaries the results for the three

Table 12.2: Parameters for EPSO and MPSO.

Population Size	20, 50 particles
Stop Condition	200 iterations
$V_{max} = X_{max}$	MAX_VALUE - MIN_VALUE
Mutation	uniform in [-0.5,0.5]
w_1	$\omega/(\omega + \phi_1 + \phi_2) = 1.0/5.10$
w_2	$\phi_1/(\omega + \phi_1 + \phi_2) = 2.05/5.10$
w_3	$\phi_2/(\omega + \phi_1 + \phi_2) = 2.05/5.10$

Table 12.3: Selected parameters for HPSO.

Population size	100 particles
Iterations	400
Bitwise mutation rate	$1/N$
$w_1 = 0$	$w_2 = w_3 = 1/2$
$w_1 = 1/6$	$w_2 = w_3 = 5/12$

Table 12.4: Test results for continuous version: best and mean fitness values found by the swarm over 20 runs at last iteration (iteration 200).

		BPSO			EPSO			MPSO		
Dim.		2	10	30	2	10	30	2	10	30
F1C	Best	-5.35e-14	-1.04	-59.45	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
	Mean	-6.54e-09	-20.75	-168.19	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
F2C	Best	-0.00	-36.18	-1912.05	-0.71	-8.98	-28.97	-0.66	-8.96	-28.97
	Mean	-97.91	-979.56	-8847.44	-1.0	-9.0	-29.0	-1.0	-9.0	-29.0
F3C	Best	-3.06e-05	-8.05	-18.09	0.0	0.0	0.0	0.0	0.0	0.0
	Mean	-0.00	-14.86	-20.49	0.0	0.0	0.0	0.0	0.0	0.0
F4C	Best	-0.31	-1.10	-6.67	-0.29	-1.0	-1.0	-0.29	-1.0	-1.0
	Mean	-1.52	-2.98	-17.04	-0.29	-1.0	-1.0	-0.29	-1.0	-1.0
F5C	Best	-0.33	-58.78	-305.11	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
	Mean	-10.41	-160.98	-504.62	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0

algorithm presented, over the five test functions, for population size 20. Overall the GPSOs (EPSO and MPSO) compare very favourably with BPSO, outperforming it in many cases. This is particularly interesting, since it suggests that the inertia term (not present in GPSO) is not necessary for good performance.

Table 12.5 shows the mean of the best fitness value and the best fitness value over the whole population for the binary version of the algorithm. The algorithm compares well with results reported in the literature, with HPSO obtaining near optimal results on all functions. Interestingly, the algorithm works at its best when w_1, the weight for x_i (the particle position), is zero. This corresponds to a degenerated PSO that makes decisions without considering the current position of the particle.

Table 12.5: Test results for HPSO with selected parameters for the De Jong's test suite.

		F1	**F2**	**F3**	**F4**	**F5**
$\overline{\omega} = 0.0$	Best	-0.00015	-0.00034	-0.0	3.45170	-1.13183
	Mean	-5.51540	-54.14453	-2.594	-5.38233	-142.67853
$\overline{\omega} = \frac{1}{6}$	Best	-0.000125	-0.000297	-0.0	3.273980	-1.111220
	Mean	-5.375902	-85.170099	-2.949	-6.919343	-167.283327

```
mask:  1  2  2  3  1  3  2

  p1:  1  2  3  4  5  6  7

  p2:  3  5  1  4  2  7  6

  p3:  3  2  1  4  5  7  6

    -----------------
   o:  1  5  3  4  2  7  6
```

Figure 12.2: Example of multi-parental sorting crossover.

12.7 Geometric PSO for Sudoku

In this section we will put into practice the ideas discussed in section 12.5 and propose a geometric PSO to solve the Sudoku puzzle. In the following, we present a three parental crossover for Sudoku and show that is a convex combination.

We first define a multi-parental recombination for permutations and then prove that it respects the four requirements for being a convex combination presented in Section 12.3.3.

Let us consider the example in Figure 12.2 to illustrate how the *multi-parental sorting crossover* works.

The mask is generated at random and is a vector of the same length of the parents. The number of 1's, 2's and 3's in the mask is proportional to the recombination weights w_1, w_2 and w_3 of the parents. Every entry of the mask indicates to which parent the other two parents need to be equal to for that specific position. In a parent, the content of a position is changed by swapping it with the content of another position in the parent. The recombination proceeds as follows. The mask is scanned from the left to the right. In position 1 the mask has a 1. This means that, at position 1, parent p_2 and parent p_3 have to become equal to parent p_1. This is done by swapping the element 1 and 3 in parent p_2 and the element 1 and 3 in parent p_3. The recombination now continues on the updated parents: parent p_1 is left unchanged and current parent p_2 and parent p_3 are the original parent p_2 and p_3 after the swap. At position 2 the mask has 2. This means that, at position 2, current parent p_1 and current parent p_3 have to become equal to current parent p_2. So at position 2, parent p_1 and parent p_3 have to get 5. To achieve this, in parent p_1 we need to swap elements 2 and 5 and in parent p_3 we need to swap elements 2 and 5. The recombination continues on the updated parents for position 3 and so on, up to the last position in the mask. At this point, the three parents are now equal because at each position one element of the permutation has been fixed in that position and it is automatically not involved in any further swap. Therefore, after all positions have been considered, all elements are fixed. The permutation to which the three parents converged is the offspring permutation. This recombination sorts by swaps the three parents towards each others according to the contents of the crossover mask and the offspring is the result of this multiple sorting. This recombination can be easily generalized to any number of parents.

Theorem 12.7.1. *(Geometricity of three-parental sorting crossover) Three-parental sorting crossover is a geometric crossover under swap distance.*

Proof. A three-parental sorting crossover with recombination mask m_{123} is equivalent to a sequence of two two-parental sorting crossovers: the first between parent p_1 and p_2 with recombination mask m_{12} obtained by substituting all 3's with 2's in m_{123}. The offspring p_{12} so obtained is recombined with p_3 with recombination mask m_{23} obtained by substituting all 1's with 2's in m_{123}. So, for Theorem 12.2.1, the three-parental sorting crossover is geometric. □

Theorem 12.7.2. *(Coherence between weights and distances) In weighted multi-parent sorting crossover, the swap distances of the parents to the expected offspring are decreasing functions of the corresponding weights.*

Proof. The weights associated to the parents are proportional to their frequencies in the recombination mask. The more occurrences of a parent in the recombination mask, the smaller the swap distance between this parent and the offspring. This is because the mask tells the parent to copy at each position. So, the higher the weight of a parent, the smaller its distance to the offspring. □

The weighted multi-parental sorting crossover is a convex combination operator satisfying the four requirements of a metric convex combination for the swap space: convex weights sum to 1 by definition, convexity (geometricity, Theorem 12.7.1), coherence (Theorem 12.7.2) and symmetry (self-evident).

The solution representation for Sudoku is a vector of permutations. For the product geometric crossover theorem, the compound crossover over the vector of permutations that applies a geometric crossover to each permutation in the vector is a geometric crossover. Theorem 12.7.3 extends to the case of a multi-parent geometric crossover.

Theorem 12.7.3. *(Product geometric crossover for convex combinations) A convex combination operator, applied to each entry of a vector, results in a convex combination operator for the entire vector.*

Proof. The product geometric crossover theorem (chapter 9) is true because the segment of a product space is the Cartesian product of the segments of its projections. A segment is the convex hull of two points (parents). More generally, it holds that the convex hull (of any number of points) of a product space is the Cartesian product of the convex hulls of its projections [154]. The product geometric crossover then naturally generalises to the multi-parent case. □

12.8 Experimental results for Sudoku

In order to test the efficacy of the GPSO algorithm on the Sudoku problem, we ran several experiments in order to thoroughly explore the parameter space and variations of the algorithm. The algorithm in itself is a straightforward implementation of the GPSO algorithm given in Section 12.3.4 with the search operators for Sudoku presented in Section 12.7.

The parameters we varied were swarm sociality (w_2) and memory (w_3), each of which were in turn set to 0, 0.2, 0.4, 0.6, 0.8 and 1.0. Since the attraction to each particle's position is defined as $w_1 = 1 - w_2 - w_3$, the space of this parameter was implicitly explored. Likewise, mutation probability was set to either 0, 0.3, 0.7 or 1.0. The swarm size was set to be either 20, 100 or 500 particles, but the number of updates was set so that each run of the algorithm resulted in exactly 100,000 fitness evaluations (thus performing 5,000, 1,000 or 200 updates). Further, each combination was tried with ring topology, von Neumann topology (or lattice topology) and global topology.

Table 12.6: Average of bests of 50 runs with population size 100, lattice topology and mutation 0.0 varying sociality (vertical) and memory (horizontal).

	Memory					
Sociality	**0.0**	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**
1.0	208	-	-	-	-	-
0.8	227	229	-	-	-	-
0.6	230	233	235	-	-	-
0.4	231	236	237	240	-	-
0.2	232	239	241	**242**	**242**	-
0.0	207	207	207	207	207	207

As explained in Section 12.5, there are two alternative ways of producing a convex combination: either using a convex combination operator or simply apply twice a 2-parental weighted recombination with appropriate weights to obtain the convex combination. Both ways to produce convex combination operators, explicit and implicit, were tried on preliminary runs and turned out to produce indistinguishable results. In the end we used the convex combination operator.

12.8.1 Effects of varying coefficients

The best population size is 100. The other two sizes we studied (20 and 500) were considerably worse. The best topology is the lattice (von Neumann) topology. The other two topologies we studied were worse (see Table 12.9).

Table 12.7: Average of bests of 50 runs with population size 100, lattice topology and mutation 0.3 varying sociality (vertical) and memory (horizontal).

	Memory					
Sociality	**0.0**	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**
1.0	238	-	-	-	-	-
0.8	238	237	-	-	-	-
0.6	239	239	240	-	-	-
0.4	240	240	241	241	-	-
0.2	240	241	**242**	**242**	**242**	-
0.0	213	231	232	233	233	233

Table 12.8: Average of bests of 50 runs with population size 100, lattice topology and mutation 0.7 varying sociality (vertical) and memory (horizontal).

	Memory					
Sociality	**0.0**	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**
1.0	232	-	-	-	-	-
0.8	232	240	-	-	-	-
0.6	228	241	241	-	-	-
0.4	224	**242**	**242**	**242**	-	-
0.2	219	234	**242**	**242**	**242**	-
0.0	215	226	233	233	236	236

From Tables 12.6-12.8, we can see that mutation rates of 0.3 and 0.7 perform better than no mutation at all. We can also see that parameter settings with w_1 (i.e., the attraction ot the particle's previous position) set to more than 0.4 generally perform badly. The best configurations generally have w_2 (i.e., sociality) set to 0.2 or 0.4, w_3 (i.e., memory) set to 0.4 or 0.6, and w_1 0 or to 0.2. This gives us some indication of the importance of the various types of recombinations in GPSO as applied at least to this particular problem. Surprisingly, the algorithm works at its best when the weight of the particle position (w_1) is zero or nearly zero. In the case of w_1 set to 0, GPSO in fact degenerates to a type of evolutionary algorithm with deterministic uniform selection, mating with the population best with local replacement between parents and offspring.

Table 12.9: Success rate of various methods.

Method	**Success**
GA	50/50
Hillclimber	35/50
GPSO-global	7/50
GPSO-ring	20/50
GPSO-von Neumann	36/50

12.8.2 PSO vs EA

Table 12.9 compares the success rate of the best configurations of various methods we have tried. Success is here defined as the number of runs (out of 50) where the global optimum (243) is reached. All the methods were allotted the same number of function evaluations per run.

From the table, we can see that the von Neumann topology clearly outperforms the other topologies we tested, and that a GPSO with this topology can achieve a respectable success rate on this tricky non-continuous problem. However, the best genetic algorithm still significantly outperforms the best GPSO we have found so far. We believe this to be at least partly the effect of the even more extensive tuning of parameters and operators undertaken in our GA experiments.

12.9 Summary

We have extended the geometric framework with the notion of multi-parent geometric crossover, that is a natural generalization of two-parental geometric crossover: offspring are in the convex hull of the parents. Then, using the geometric framework, we have shown an intimate relation between a simplified form of PSO (without the inertia term) and evolutionary algorithms. This has enabled us to generalize in a natural, rigorous and automatic way PSO for any type of search space for which a geometric crossover is known.

We specialized the general PSO to Euclidean, Manhattan and Hamming spaces, obtaining three instances of the general PSO for these spaces: EPSO, MPSO and HPSO, respectively. We have performed extensive experiments with these new GPSOs. In particular, we applied EPSO, MPSO and HPSO to standard sets of benchmark functions and obtained a few surprising results. Firstly, the GPSOs have performed really well, beating the canonical PSO with standard parameters most of the time. Secondly, they have done so right out of the box. That is, unlike the early versions of PSO which required considerable effort before a good general set of parameters could be found, with GPSO we have done very limited preliminary testing and parameter tuning, and yet the new PSOs have worked well. This suggests that they may be quite robust optimisers.

Thirdly, HPSO works at its best with only weak attraction toward the current position of the particle. With this configuration, GPSO almost degenerates to a type of genetic algorithm.

An important feature of the GPSO algorithm is that it allows one to automatically define PSOs for all spaces for which a geometric crossover is known. Since geometric crossovers are defined for all of the most frequently used representations and many variations and combinations of those, our geometric framework makes it possible to derive PSOs for all such representations. GPSO is rigorous generalization of the classical PSO to general metric spaces. In particular, it applies to combinatorial spaces.

We have demonstrated how simple it is to specify the general GPSO algorithm to the space of Sudoku grids (vectors of permutations), using both an explicit and implicit definition of convex combination. We have tested the new GPSO on Sudoku and have found that: (i) the communication topology makes a huge difference and that the lattice topology is by far the best; (ii) as for HPSO, the GPSO on Sudoku works better with weak attraction toward the current position of the particle; (iii) the GSPO on Sudoku finds easily near-optimal solutions but it does not always find the optimum. Admittedly, GPSO is not the best algorithm for the Sudoku puzzle where the aim is to obtain the correct solution all the times, not a nearly correct one. This suggests that GPSO would be much more profitably applied to combinatorial problems for which one would be happy to find near-optimal solutions quickly.

In summary, it is the first time that a PSO algorithm has been quite successfully applied to a non-trivial combinatorial space. This shows that GPSO is indeed a natural and promising generalization of classical PSO.

Chapter 13

One-point Crossover

In chapter 3 we have shown that uniform crossover has a simple and natural geometric interpretation. In this chapter we show that also one-point crossover has a surprisingly simple characterization in geometric terms. We generalize one-point crossover for generic metric spaces and specify it for a number of representations.

13.1 Introduction

Independently from the specific mask used, all offspring produced by a mask-based crossover for binary strings are in the line segment under Hamming distance between the parents strings.

A specific mask-based crossover is fully-specified by the specific joint probability distribution characterizing the co-occurrence of values in the mask at different positions. For uniform crossover, there is equal probability of a one or zero at any position in the mask and this probability is independent from the probabilities at other positions.

Thanks to its simple and symmetric definition, the probability distribution characterizing uniform crossover can be expressed entirely as a function of the Hamming distance, consequently it can be naturally generalized to generic metric spaces by replacing the Hamming distance with a generic distance.

One-point crossover for binary strings selects a common crossover point uniformly at random on the length of the parent strings and produces two offspring by swapping the tails of the parent strings after the crossover point. *Cutting and swapping tails is an operation so much relying on* *the special characteristic of binary strings of being a vector that it would seem there is no hope to cast it in general geometric terms*, hence no hope to find a general formal recipe to find the equivalent of one-point crossover for any solution representation as we did for uniform crossover. In this chapter, we prove this intuition wrong, and, in section 13.2, we propose a generalization of one-point crossover to generic metric spaces. In section 13.3, we specialize one-point geometric crossover for Euclidean and Manhattan spaces, permutations, GP trees, sequences and sets.

13.2 Generalization of one-point crossover

13.2.1 Partially-specified geometric operators

In the following we illustrate a subtle but important difference among search operators defined using the distance associated with the search space. As we will see in the next sections, this difference is important for the generalisation of one-point crossover to general metric spaces.

As we have seen in chapter 3, geometric operators are search operators *completely* defined in two steps:

- the occurrence of the offspring is limited to a certain part of the space defined as some geometric condition between the location of the parents and those of the offspring (image of the operator). This defines a *class* of search operators with a common geometric characteristic.
- a specific *instance* of a geometric operator belonging to a given class is fully-specified when a conditional probability distribution over the image of the operator is given.

The geometric shapes that define the class of geometric operators such as geometric mutation and geometric crossover can be expressed in terms of distances only. However, specific instances of geometric operators belonging to this class do not necessarily need be functions of the distances among parents and offspring. This is because the actual values of probability of picking offspring are not necessarily function of the distances among offspring and parents only, but they also depend on some peculiar characteristics of the underlying solution representation. For example, in variable-length representations, this probability could be also function of the size of the parents. So, we have two types of geometric operators in relation to whether they can be defined using distances only: (i) those for which both image and probability distribution are fully-defined using distances only; (ii) those for which only the image is fully-defined using distances but the probability distribution requires extra-parameters to be fully-specified. This distinction is important because it sets apart uniform crossover and one-point crossover.

Uniform crossover for binary strings is a geometric operator of type (i) because it is *fully-specified* as a function of the Hamming distance only: its conditional probability distributions is only function of the Hamming distance and, in particular, it does not require any reference to the underlying solution representation. Because of this, uniform crossover can be generalized to any metric space by substituting the Hamming distance with a generic metric so giving rise to a *general definition which is fully-specified* without requiring any extra information about the underlying representation.

In the following section, we will show that one-point crossover for binary strings is a geometric operator of type (ii), that is, it is only *partially-specified* as a function of the Hamming distance and it needs some further reference to the specific representation to be fully-specified. This, as we will see, affects its generalization to generic metric spaces.

13.2.2 One-point geometric crossover

In this section we consider crossover operators for binary strings that return one offspring. In the next section we consider crossover operators that return two complementary offspring.

Definition 13.2.1. (Sequence generator) A sequence generator is an operator that given in input two binary strings a and b of length n and a permutation p of n elements returns a sequence $s = \{s_0, s_1, \cdots, s_n\}$ of binary strings that starts from a and ends in b as follows. The permutation p is a priority vector which specifies in which order to consider the positions on the alignment of a and b. The initial element of the sequence s is the string a, so $s_0 = a$. The subsequent element s_1 is obtained by exchanging the bits of the strings a and b at the position i specified by the permutation p for iteration 1 (i where $p_i = 1$). The second new element s_2 is obtained by exchanging the bits of s_1 and b at the position i specified by the permutation p for iteration 2 (i where $p_i = 2$). So on so forth.

In figure 13.1 we give an example of sequence generator. The sequence of binary strings

$s_0, s_1, ..., s_5$ is generated from parent a and b using the permutation p as explained in the definition of sequence generator (definition 13.2.1). Notice that, in the sequence, s_1, s_2 and s_3 correspond to the same binary string. The bottom part of the figure reports the sequence $s_0, s_1, ..., s_5$ after removing those binary strings which are repeated, leaving only one copy of each distinct binary string.

```
 p: 1 3 5 2 4
 a: 1 1 1 1 1
 b: 0 1 0 1 0

s0: 1 1 1 1 1
s1: 0 1 1 1 1
s2: 0 1 1 1 1
s3: 0 1 1 1 1
s4: 0 1 1 1 0
s5: 0 1 0 1 0

sequence without repetitions:

    1 1 1 1 1
    0 1 1 1 1
    0 1 1 1 0
    0 1 0 1 0
```

Figure 13.1: Example of sequence generator.

Theorem 13.2.1. *After removing the repetitions, the sequence s generated by the sequence generator applied to the strings a and b is a shortest path linking a and b in the Hamming space for any choice of the generating permutation p.*

Proof. We have a number of remarks on the sequence s that will lead to prove this theorem. First, the final string of s is the string b because the permutation p exchanged one by one the contents of a at every location into the contents of b. Second, two consecutive strings in the sequence s differ at most by one bit by construction. So they form a connected path in the Hamming space, possibly with some subsequence of repeated states. Let us call s' the path without repetitions. Third, the number of strings in s' is $HD(a, b) + 1$ because a and b differ at exactly $HD(a, b)$ positions by definition of Hamming distance hence by changing a one position at a time one gets exactly $HD(a, b)$ new strings, no more than that because there are not enough differences between a and b, no less than that because all positions have been processed. So including the initial string a the number of strings in the sequence s' is $HD(a, b) + 1$. Finally, the sequence s' is a path of length $HD(a, b)$ (the length of the path is given by the number of arcs connecting the nodes). Hence, s' is a shortest path between a and b because the length of this path equals the Hamming distance between a and b that by definition is the smallest possible distance between a and b. □

Theorem 13.2.2. *All geodesics (shortest paths) between a and b in the Hamming space are generated by using all possible permutations as input of the sequence generator and there are $HD(a, b)!$ distinct shortest paths between a and b.*

Proof. We have a few more remarks on the sequence s that will lead to prove this theorem. First, the fact that in the sequence s there are subsequences of repeated elements has its origin in the fact that at some positions the strings a and b do not differ. When the contents at that positions are exchanged the newly generated string equals the previous string in the sequence. So, if we restrict the permutation that generates the sequence to those positions in which a and b differ we can generate the sequence s' outright without repetitions. By definition, the number of positions in which a and b differ is $HD(a, b)$, so the reduced permutation p' needs to be a permutation of $HD(a, b)$ elements. Second, different reduced permutations produce different sequences. Third, each reduced permutation defines a complete order on how to sequentially apply all the differences between a and b to a to be turned into b without backtracking. Hence, all reduced permutations, that is, all orders of application of these differences, account for all shortest paths between a and b. So we have $HD(a, b)!$ distinct shortest paths between a and b, that is the number of possible orders of the reduced permutation on a and b. □

Theorem 13.2.3. *(One-point crossover) All possible offspring of one-point crossover for binary strings are on a single geodesic in the Hamming space between parents.*

Proof. All the offspring generated by one-point crossover can be generated from the left to the right by exchanging one by one the contents of the two parents at each position. This is the same sequence obtained by applying the sequence generator with generating permutation $(1 \dots n)$ to the two parents where n is the length of the parents. For theorem 13.2.1 this is a shortest path linking the parents. □

Since the notion of geodesic is well-defined in every metric space, the previous theorem allows us to generalize one-point crossover to any metric space, as follows.

Definition 13.2.2. (One-point geometric crossover) In one-point geometric crossover all offspring are on a single geodesic between parents.

Notice that in general there may be more than one geodesic between two points (parents). We leave deliberately the geodesic unspecified. It could be any, as long as the offspring are all on it. *The reason we do not specify a specific geodesic is that since they are all indistinguishable from a distance viewpoint, we cannot specify anyone in particular using only the distance.* Only using extra-information based on the underlying representation, we can refer to one geodesic in particular. So a representation-independent definition of one-point crossover based on distance necessarily cannot refer to any particular geodesic. This necessary indetermination in the definition of one-point geometric crossover makes it an "over-generalization" of the one-point crossover for binary strings, as we show in the following.

To see this, let us first specify the probability distribution in the following definition:

Definition 13.2.3. (Uniform one-point geometric crossover) Any uniform one-point geometric crossover picks offspring uniformly at random on a geodesic between parents.

Unlike the uniform geometric crossover that, when applied to a specific space, returns a unique *fully-specified* search operator, uniform one-point geometric crossover is an *inherently partially-specified* search operator that returns a family of operators. This difference originates from the fact that uniform geometric crossover is based on the segment between two given points (parents) and this is unique in any metric space. Uniform one-point geometric crossover is based on a geodesic between two given points (parents) and in general there could be more than one linking these two points. So, given a metric space there is a unique notion of uniform geometric crossover, in contrast, *the notion of uniform one-point crossover is non unique with respect to its underlying metric space.*

In the specific case of the Hamming space, the definition of uniform one-point geometric crossover does not give rise to a single search operator, namely the traditional one-point crossover for binary strings. It corresponds, instead, to a *family of operators* based on the sequence generator fed with any possible generating permutation; it returns any one of the strings in the output sequence without repetitions uniformly at random.

When a space allows for different uniform one-point geometric crossovers, they are all indistinguishable from a distance viewpoint. So all are the right uniform one-point geometric crossovers for the specific space. In section 13.3 we will see that in some spaces one member of this family may be preferable to others for reasons linked with the specific character of the underlying representation.

13.2.3 N-point geometric crossover

In this section we extend the result of generalization of one-point crossover to the general n-point crossover.

It is tempting to conjecture that since 1-point geometric crossover picks points on a single geodesic, two-point geometric crossover picks points on two distinct geodesics and so on. This can be shown to be wrong by a counting argument: the number of different offspring generated by two-point crossover is bigger than the number of offspring laying on two shortest paths between parents. The way of generalizing n-point crossover is another. To state it, first we need to generalize the notion of complementary offspring to generic metric spaces.

Definition 13.2.4. (Complementary offspring) Let a and b be two parents, and c an offspring under geometric crossover. The complementary offspring c' has the property that $d(a, c') = d(b, c)$ and $d(b, c') = d(a, c)$ with $d(c, c')$ maximal.

It is easy to verify that for binary strings under Hamming distance this corresponds to the traditional notion of complementary offspring (generated using complementary crossover masks). The example in figure 13.2 illustrates the relation among the distances of the parents a and b and the offspring c and c' obtained by one-point crossover.

```
 a: 1 1|1 1 1
 b: 0 1|0 1 0

 c: 1 1 0 1 0
c': 0 1 1 1 1

d(a,c')=d(b,c)=1
d(b,c')=d(a,c)=2

d(c,c')=d(a,b)=3
```

Figure 13.2: Distance relation between complementary offspring.

For the case of binary strings, the condition of $d(c, c')$ being maximal is automatically satisfied because the distance between complementary offspring equals always the distance of their parents. This is the maximal distance between offspring because two points in a (metric) segment cannot be at a greater distance than the length of the segment, which is the distance between its extremes (the parents).

Lemma 13.2.4. *(Uniqueness of the conjugate end-point of a metric segment) Given a metric segment s on any metric d with an end-point a, there is a unique conjugate end-point b such that $s = [a, b]_d$.*

Proof. Let us assume by absurd that there are two distinct conjugate end-points b and c of a of the segment s. Then we have $a, b, c \in s$. The length of the segment s is by definition the distance between its end-points. Then we have that the length $l(s)$ of the segment s equals both the distances $d(a, c)$ and $d(a, b)$. This cannot be because $c \in [a, b]$ implies $d(a, c) + d(c, b) = d(a, b)$, so $l(s) + d(c, b) = l(s)$ and consequently $d(c, b) = 0$, hence $b = c$. □

Lemma 13.2.5. *(End-points exchange of a metric segment) On a metric space d with convex segments, if $b, b' \in [a, a']$, and $d(b, b') = d(a, a')$, then $[a, a'] = [b, b']$.*

Proof. We first prove that $a, a' \in [b, b']$. Since $b, b' \in [a, a']$, we have that $d(a, b) + d(b, a') = d(a, a')$ and $d(a, b') + d(b', a') = d(a, a')$. Summing these equations and using $d(b, b') = d(a, a')$ we get $d(b, a) + d(a, b') + d(b, a') + d(a', b') = 2d(b, b')$. For the triangular inequality between b, a and b' and between b, a' and b', the previous expression is true only when $d(b, a) + d(a, b') = d(b, b')$ and $d(b, a') + d(a', b') = d(b, b')$. This means $a, a' \in [b, b']$.

In a metric space with convex segments, we have that $b, b' \in [a, a']$ implies $[b, b'] \subseteq [a, a']$. We have also that $a, a' \in [b, b']$ implying $[a, a'] \subseteq [b, b']$. So, we have $[a, a'] = [b, b']$. □

Theorem 13.2.6. *For the case of binary strings, the definition of complementary offspring (definition 13.2.4) gives rise to a unique complementary offspring.*

Proof. The Hamming space has convex segments[154]. The Hamming distance $d(c, c')$ between two complementary children c and c' equals the Hamming distance $d(a, b)$ between their parents a and b. Also, since homologous crossover is a geometric crossover under Hamming distance we have that $c, c' \in [a, b]_d$. So, for the end-points exchange lemma (lemma 13.2.5) we have $[c, c'] = [a, b]$. So, c is an end-point of the segment $[a, b]$ with conjugate c'. So, for the lemma on the uniqueness of the conjugate end-point (lemma 13.2.4), c' is unique. □

In section 13.3 we will specify complementary offspring for a number of different spaces. This turns out to be a very instructive exercise. A word of warning: the definition of complementary offspring does not always lead to a *unique* complementary offspring. In fact, it is possible to show that in some metric spaces the complementary offspring of one point is not a unique point and that in some other metric spaces the complementary offspring of one point does not exist. Fortunately, these metric spaces are extreme cases and are not associated with simple and regular solution representations.

Theorem 13.2.7. *The version of one-point crossover for binary strings that returns two complementary offspring differs from the version that returns a single offspring in the fact that complementary offspring, in general, lay on two distinct geodesics and not on the same one.*

Proof. We have to prove that the complementary offspring, in general may lay on more than one geodesic. So, we have to show that there is at least a case for which this is true. Also, we have to show that they do not lay on more than two geodesics.

The first geodesic is generated by inputting the generating permutation $(1 \dots n)$ to the sequence generator applied to the parent strings as in the case of single offspring one-point crossover. The second geodesic, which corresponds to the sequence of complementary offspring in reversed order, is generated by using $(n \dots 1)$ as generating permutation with the sequence generator applied to the parent strings. In the example in figure 13.3, a and b are the parent strings, and p and p' the generating permutations for the offspring and the complementary offspring, respectively. The geodesic of the offspring is the sequence $s_0, s_1, \cdots, s_5$, and the geodesic of the complementary offspring is the sequence $s'_0, s'_1, \cdots, s'_5$.

```
  p: 1 2 3 4 5    p’: 5 4 3 2 1
  a: 1 1 1 1 1     a: 1 1 1 1 1
  b: 0 1 0 1 0     b: 0 1 0 1 0

 s0: 1 1 1 1 1   s0’: 1 1 1 1 1
 s1: 0 1 1 1 1   s1’: 1 1 1 1 0
 s2: 0 1 1 1 1   s2’: 1 1 1 1 0
 s3: 0 1 0 1 1   s3’: 1 1 0 1 0
 s4: 0 1 0 1 1   s4’: 1 1 0 1 0
 s5: 0 1 0 1 0   s5’: 0 1 0 1 0
```

Figure 13.3: Example of complementary offspring on distinct geodesics.

Each sequence contains all points on a geodesic. The two sequences contain different points, so the offspring are on two distinct geodesics. By construction, the second sequence contains in reverse order the complementary offspring of the first sequence. So, for example the complementary offspring of the second element of the first sequence (s_1) is the second element from the bottom of the second sequence (s'_4). □

The way to generalize two-point crossover to generic metric spaces is to notice that two-point crossover for binary strings can be decomposed into two applications of one-point crossover (see figure 13.4): the first application (phase 1) takes in input the two parents a and b and returns the two complementary offspring c_1 and c_2; the second application (phase 2) takes in input c_1 and c_2 and returns the two complementary offspring d_1 and d_2. Analogously, three-point crossover can be decomposed into three applications of one-point crossover and so forth.

```
 a: 1 1|1 1|1
 b: 0 1|0 1|0

o1: 1 1 0 1 1
o2: 0 1 1 1 0

phase 1:

 a: 1 1|1 1 1
 b: 0 1|0 1 0

c1: 1 1 0 1 0
c2: 0 1 1 1 1

phase 2:

c1: 1 1 0 1|0
c2: 0 1 1 1|1

d1: 1 1 0 1 1
d2: 0 1 1 1 0

o1 = d1; o2 = d2
```

Figure 13.4: Example of decomposition of two-point crossover into a sequence of two one-point crossovers.

Definition 13.2.5. (Geometric n-point crossover) A recombination operator is a geometric n-point crossover if its offspring can be obtained by the sequential application of n times the same[1] one-point geometric crossover to the complementary offspring output of the previous application.

Geometric n-point crossover is only function of the distance because it is defined in terms of one-point geometric crossover that is only function of the distance. Hence it can be specified for any representation. When geometric n-point crossover is specified for a new representation, it does not need to be necessarily implemented as multi-stage one-point crossover. For some representations, geometric n-point crossover may be equivalently expressed as a single stage operation, as for example the multi-cut operation in the case of binary strings.

13.3 Specialization of one-point geometric crossover

13.3.1 Euclidean and Manhattan spaces

Since in the Euclidean space a segment comprises a single geodesic, in this specific space there is only one possible specification of one-point geometric crossover. *Both uniform one-point geometric crossover and uniform geometric crossover when specified for the Euclidean space pick points uniformly at random on this only geodesic. So they become equivalent.*

Specifying the definition 13.2.4 of complementary offspring to the case of Euclidean space, complementary offspring of geometric crossover in the Euclidean space are on the same geodesic at the same distances from different parents. Notice also that the midpoint of the segment is the complementary of itself. For the Euclidean space, the images of geometric crossover, one-point crossover and, in general, n-point crossover coincide with the entire segment between two parents. *So, the image sets of 1-point geometric crossover and n-point geometric crossover when specified for the Euclidean space coincide. However, the probability distributions of their offspring differ.*

[1]Notice that a one-point geometric crossover for being fully specified requires that a specific geodesic be specified together with a probability distribution on it. The way to identify a specific geodesic depends on the specific underlying representation. Independently from the specific probability distribution, once we have a one-point geometric crossover with a geodesic specified, a two-point crossover *based on the same geodesic* can be defined. Since there are as many one-point geometric crossovers as the number of geodesics, there will be as many two-point geometric crossover. This extends to n-point crossovers.

In the 2-D Manhattan space, a segment is a rectangle and its endpoints are two diagonally opposite corners. Unlike the Euclidean case, a segment in the 2-D Manhattan space has two pairs of endpoints: the endpoints of the two diagonals connecting opposite corners of the rectangle. In higher dimensions, a segment becomes a hyper-rectangle. For this specific space, uniform geometric crossover picks uniformly offspring in the hyper-rectangle.

In the 2-D Manhattan space, a segment does not coincide with a single geodesic: it comprises infinitely many geodesics linking two points. The geodesics between two points are all monotonic curves joining the two points. The union of all the points of all geodesics corresponds to the segment between the two points. *So in the Manhattan space there are infinitely many one-point crossovers, one for each monotonic curve connecting the two parents.*

Applying the definition 13.2.4 of complementary offspring to the case of Manhattan space, we obtain an unpleasant effect: given the offspring a and its complement b, the complement of b is not always a. This requires some revision of the definition of complementary offspring. We suggest the following. When a segment has more than a pair of endpoints, we further require that the complementary offspring must be at the same distances from any pairs of endpoints of the segment. So, let us consider two parents a and c; for the definition of geometric crossover under Manhattan distance in the 2-D plane, a and c can pick offspring in the rectangle which has corners a, b, c and d, where a and c, and b and d are pairs of opposite corners of the rectangle. Any offspring o of a and c is also offspring of b and d and *vice versa*. The original requirement of complementary offspring o' is that: $d(a, o) = d(c, o')$ and $d(c, o) = d(a, o')$. As extra requirement we also ask: $d(b, o) = d(d, o')$ and $d(d, o) = d(b, o')$. With this extra requirement, *complementary offspring of geometric crossover in the Manhattan plane are situated in symmetric positions with respect to the diagonals of the rectangle.* Notice that the extra requirement is well-defined for generic metric spaces and it is compatible with the traditional notion of complementary offspring for binary strings.

An example of two-point geometric crossover in the Manhattan plane follows. Let us first consider a specific one-point crossover by specifying a geodesic for each pair of possible parents

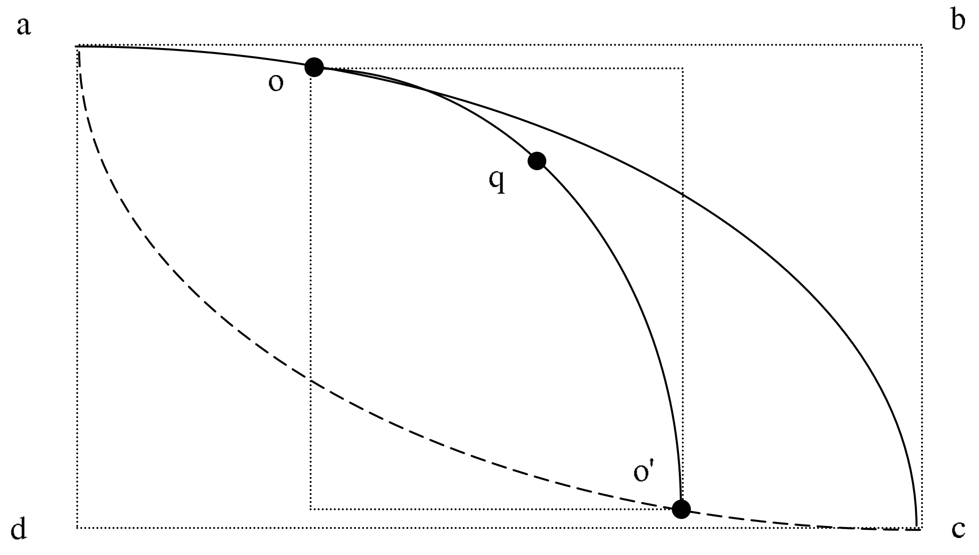


Figure 13.5: 2-point crossover in Manhattan plane.

(points in the plane). For this, we consider a monotone curve joining two points described by an equation e where the two points are parameters in the equation of the curve. In figure 13.5, the parents a and c identify the rectangle *abcd*; the geodesic joining a and c is the solid decreasing curve joining them ($e(a, c)$). The uniform one-point geometric crossover associated with this curve returns offspring uniformly at random on this curve. The point o in the figure is a offspring of a and c. The point o' is the complementary offspring with respect to o: for symmetry, from the figure it is easy to see that the Manhattan distances of o from parent a equals that of o' from parent c, and the same holds exchanging parents. Notice that there are other points in the rectangle *abcd* that fulfill the equations on the distances. The extra condition of complementary offspring introduced above requires that the Manhattan distances of o from b equals that of o' from d, and the same holds exchanging b and d. In the figure, this can be seen to hold for symmetry. Also, this requirement makes o' the only complement of o and *vice versa*. To obtain a two-point geometric crossover, we apply the *same* one-point geometric crossover based on the same (parametric) geodesic to o and o' producing the offspring q on the curve $e(o, o')$. This offspring is also the offspring of two-point geometric crossover of a and c associated with the geodesic equation e.

13.3.2 Permutations

In chapter 5 we have studied various crossovers for permutations, revealing that PMX, a well-known crossover for permutations, is geometric under swap distance. Also, we found that cycle crossover, another traditional crossover for permutations, is geometric under swap distance and under Hamming distance.

We also showed that geometric crossovers for permutations based on edit moves are naturally associated with sorting algorithms: picking offspring on a minimum path between two parents corresponds to picking partially sorted permutations on the minimal sorting trajectory between the parents. We called these crossovers sorting crossovers.

In the following, we consider two different perspectives on one-point geometric crossover for permutations: sorting crossovers and cut-and-fill crossovers. Then, using the latter one, we will start answering the question on whether PMX is a two-point geometric crossover.

Sorting crossovers and one-point crossovers

Given two permutations, deterministic sorting algorithms sort the elements of one permutation into the order of the elements of the other permutation always on the same sorting trajectory, that is to say, given two parents they pick offspring always on the same geodesic. So, *deterministic sorting crossovers are one-point geometric crossovers.* In non-deterministic sorting algorithms, the sorting trajectory is still minimal but not deterministic, hence *sorting crossovers based on non-deterministic sorting algorithms are not one-point geometric* because for two given permutations (parents), different applications of the sorting crossover may return partially ordered permutations (offspring) belonging to different sorting trajectories (geodesics).

Cut-and-fill crossovers and one-point crossovers

We now introduce cut-and-fill crossovers that are intuitive extensions of one-point crossover for the permutation representation and then we show that they indeed are one-point geometric crossovers.

If the one-point crossover for binary strings is applied directly on permutations, the offspring so obtained are not permutation. So, this operator cannot be applied. However it can be easily adapted: the first parent is cut at a crossover point and the part before the cutting point is passed to the offspring as in the traditional one-point crossover; the second part is then filled in using the order in the second parent avoiding elements already present in the offspring before the crossover point. We call this crossover insertion cut-and-fill crossover. Figure 13.6 shows an example of this crossover. $P1$ and $P2$ are the parent permutations and O is the offspring permutation. The vertical bar in $P1$ indicates the crossover point, and the dashes indicate elements of parent $P2$ whose relative order is preserved in the offspring O.

```
P1: a b c|d e f
      _   _ _
P2: c d b f e a

O:  a b c d f e
```

Figure 13.6: Example of insertion cut-and-fill crossover.

The insertion cut-and-fill crossover is one-point geometric because it is equivalent to a sorting crossover based on the insertion move: it is like sorting parent $P2$ into parent $P1$ using the insertion sort algorithm and stopping it when all the elements before the crossover point are sorted.

The connection between cut-and-fill crossover and sorting crossover suggests that more types of cut-and-fill crossovers can be defined depending on the type of move used as base of the sorting. So we can define a swap cut-and-fill crossover that sorts parent $P2$ into parent $P1$ using selection sort (swap-based minimal sort algorithm) and stopping it when the elements before the crossover point are sorted. *Since swap cut-and-fill crossover is based on a deterministic sorting algorithm, it is a one-point geometric crossover.*

Figure 13.7 gives an example of swap cut-and-fill crossover. Using the same parents and the same crossover point as for the insertion cut-and-fill crossover we obtain a different offspring (the dashes indicate the elements of parent $P2$ that have been swapped to match the order of parent $P1$):

```
P1: a b c|d e f

P2: c d b f e a
    -         -
    a d b f e c
      - -
    a b d f e c
        -     -
O:  a b c f e d
```

Figure 13.7: Example of swap cut-and-fill crossover.

If, as a base of the cut-and-fill crossover, we use the adjacent swap move that is associated with the bubble sort algorithm, we obtain again the insertion cut-and-fill crossover. This happens because, apart from the number of moves required (an insertion is equivalent to a sequence of adjacent swaps), after ordering the elements of parent $P2$ into the order of parent $P1$ up to the crossover point the order of the remaining elements is the same as when using insertions or adjacent swaps. Notice, however, that in general edit distances based on adjacent swaps and insertions give rise to two different geometric crossovers. Imagine two parents one insertion away: under insertion distance the segment between them comprises only two points (possible offspring) the two parents themselves; under adjacent swap the possible offspring of the same two parents are all the intermediate permutations one adjacent swap away from each other that link the two parents. So, *the insertion cut-and-fill crossover is bi-geometric, under insertion distance and under adjacent-swap distance.* In chapter 5 we showed that cycle-crossover is bi-geometric as well.

PMX and two-point crossovers

As we have seen in chapter 5, PMX crossover is geometric crossover under swap distance. This crossover can be seen as a sorting crossover based on the swap move: two crossover points are selected at random; then the second parent is sorted using the minimal amount of swaps so that the elements between the two crossover points in the two parents coincide. Since PMX is a geometric crossover and, arguably, it is the closest adaptation to permutations of the traditional two-point crossover for binary strings, it is natural to wonder whether PMX is a two-point geometric crossover.

Figure 13.8 shows an example of PMX as sorting crossover. $P1$ and $P2$ are the parent permutations, and $O1$ and $O2$ the offspring permutations. The vertical bars in $P1$ and $P2$ at the top of the figure indicate the crossover points, that are at the same positions in $P1$ and $P2$. On the left, $P1$ and $P2$ are recombined to obtain $O1$. On the right, $P2$ and $P1$ are recombined to obtain $O2$. On the left (right), the dashes indicate the elements of parent $P2$ ($P1$)that have been swapped to match the order of parent $P1$ ($P2$).

```
P1: a b|c d e|f g   P2: c f|e b a|d g

P2: c f e b a d g   P1: a b c d e f g
    _   _               _   _
    e f c b a d g       a b e d c f g
          _   _           _   _
    e f c d a b g       a d e b c f g
    _       _           _       _
O1: a f c d e b g   O2: c d e b a f g
```

Figure 13.8: Example of example of PMX as sorting crossover.

In section 13.2.3, we have shown that for binary strings two-point crossover can be decomposed into a sequence of two one-point crossovers. Since PMX is geometric under swap distance, it is reasonable to conjecture that PMX is a two-point geometric crossover under swap distance. As a starting point to corroborate or disprove this conjecture, we use the swap cut-and-fill crossover, which is one-point geometric under swap distance, as a base for a two-point geometric crossover, which we name two-point swap cut-and-fill crossover, using the same crossover points, xp_1 on the left and xp_2 on the right, and the same parents, $P1$ and $P2$, as the example in figure 13.8. Figures 13.9 and 13.10 show two sequential applications of the cut-and-fill crossover equivalent to the two-point swap cut-and-fill crossover. In figure 13.9, the crossover point xp_2 is used in the first phase of the application of the swap cut-and-fill crossover, and the crossover point xp_1 in the second phase of the application of the swap cut-and-fill crossover. In the first phase, the crossover is applied twice to parents $P1$ and $P2$ with their roles exchanged, to produce offspring $O1$ and $O2$. In the second phase, the crossover is applied twice to $O1$ and $O2$ whit their roles exchanged, to produce the offspring $O3$ and $O4$ of the two-point swap cut-and-fill crossover.

```
P1: a b c d e|f g    P2: c f e b a|d g

P2: c f e b a d g    P1: a b c d e f g

O1: a b c d e f g    O2: c f e b a d g


O2: c f|e b a d g    O1: a b|c d e f g

O1: a b c d e f g    O2: c f e b a d g

O3: c f a d e b g    O4: a b e f c d g
```

Figure 13.9: Example two-point geometric crossover based on swap-based cut-and-fill crossover (second crossover-point first).

Figure 13.10 shows the same procedure as in figure 13.9 with the difference that the crossover point xp_1 is used in the first phase, and the crossover point xp_2 in the second phase. The offspring $O3$ and $O4$ coincide with the offspring $O3$ and $O4$ of figure 13.9.

Offspring $O3$ and $O4$ in the two examples above are different from offspring $O1$ and $O2$ of the example of the PMX. So the two-point geometric crossover proposed is not equivalent to PMX. Why is it so? The reason seems to lay on the fact that PMX requires the region between the two crossover points of the first parent to be passed untouched to the offspring. However, a sequence of two cut-and-fill crossovers does not fulfill this requirement.

```
P1: a b|c d e f g    P2: c f|e b a d g

P2: c f e b a d g    P1: a b c d e f g

O1: a b e f c d g    O2: c f a d e b g


O2: c f a d e|b g    O1: a b e f c|d g

O1: a b e f c d g    O2: c f a d e b g

O3: c f a d e b g    O4: a b e f c d g
```

Figure 13.10: Example of two-point geometric crossover based on swap-based cut-and-fill crossover (first crossover-point first).

13.3.3 GP trees

For GP trees, there are two recombination operators that can be thought as extensions of one-point crossover for binary strings to GP trees: Koza's subtree swap crossover and one-point homologous crossover. In chapter 14, we will show that Koza's subtree swap crossover is not a geometric crossover. So it is not a one-point geometric crossover either because this is a subclass of geometric crossover. Homologous one-point crossover aligns parent trees at the root and then cut-and-swap subtrees at the same position in the two parents. In chapter 7, we have seen that the whole family of homologous crossovers for GP trees are geometric crossovers under Structural Hamming distance, so also one-point homologous crossover is. However, in chapter 14, we will show that one-point homologous crossover is not a one-point geometric crossover. Non-geometricity results are all treated together in chapter 14.

After these two negative results, one may wonder how one-point geometric crossovers for GP trees look like. In the following we first present a theorem that helps detecting whether a crossover operator is a one-point geometric crossover. Then we consider a family of crossovers for GP trees that are a subclass of mask-based homologous crossover. Since all mask-based homologous crossovers for GP trees are geometric also this family of crossovers is geometric. Then we show that all these crossovers are one-point geometric.

Theorem 13.3.1. *The points $o_1, o_2, \ldots, o_n \in [a,b]_d$ belong to a single geodesic g linking a and b in the metric space d iff they can be ordered as $\bar{o}_1, \bar{o}_2, \ldots, \bar{o}_n$ such that $d(a,\bar{o}_1) + d(\bar{o}_1,\bar{o}_2) + \ldots + d(\bar{o}_{n-1},\bar{o}_n) + d(\bar{o}_n,b) = d(a,b)$.*

Proof. If such an order exists the length of the path g connecting a and b passing thought $o_1, o_2, \ldots, o_n$ is $d(a,b)$. So g is a shortest path linking a and b. If such an order does not exist, for each order of $\bar{o}_1, \bar{o}_2, \ldots, \bar{o}_n$ we have $d(a,\bar{o}_1) + d(\bar{o}_1,\bar{o}_2) + \ldots + d(\bar{o}_{n-1},\bar{o}_n) + d(\bar{o}_n,b) > d(a,b)$ for the triangular inequality. Then the path g connecting a and b passing thought $o_1, o_2, \ldots, o_n$ is larger than $d(a,b)$. So g is not a shortest path linking a and b. Since $o_1, o_2, \ldots, o_n \in [a,b]_d$, for each point there exists a geodesic linking a and b passing for that point. Therefore since there is no geodesic between a and b passing through all points $o_1, o_2, \ldots, o_n$ they must belong to distinct geodesics. □

Definition 13.3.1. (Ordered homologous crossover family) Let us define a total order on the nodes of the common region of two parent trees. The order is a (deterministic) function of the two parents. The offspring that the two parents can produce are those obtained, exchanging in the parents the node at position 1 in the order, plus those obtained by exchanging *simultaneously* the nodes at positions 1 and 2, plus those obtained by exchanging *simultaneously* nodes at positions 1, 2 and 3 and so on.

For example, we can define a total order on the common region by numbering its nodes starting from the root and then visiting and numbering successive nodes of the common region using a breath-first strategy. Then generating uniformly a random number k between 1 and the number of nodes in the common region and exchanging in the two parents all nodes up to k. So we could call this crossover breath-first homologous geometric crossover. One could change the numbering strategy with a depth-first or bottom-up or any other strategy that visits all nodes of a tree and obtain new geometric crossover belonging to the ordered homologous crossover family.

Figure 13.11 illustrates the breath-first ordered homologous crossover. The crossover mask tells for each position of the common region from which of the parents to take the node or subtree to pass to the offspring. The crossover mask on the bottom left is valid because the nodes from 1 to 5 marked with 'X' will be passed to the offspring from parent 1; the node from 6 to 10 marked with 'Y' will be passed to the offspring from parent 2. The crossover mask on the right is invalid for the breath-first ordered homologous crossover because the numbering of the

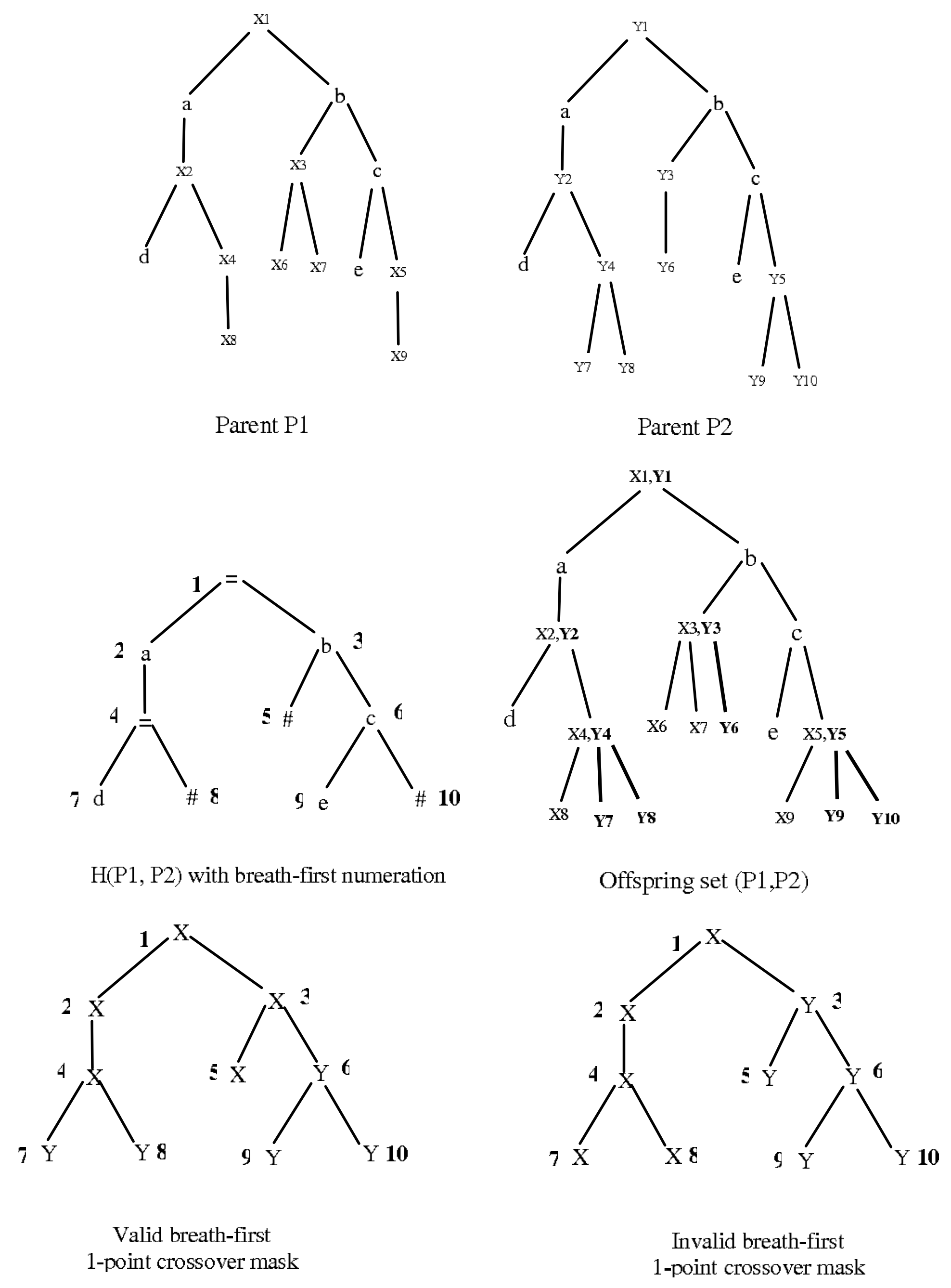


Figure 13.11: Breath-first ordered homologous crossover for GP trees: (top) two parent trees P1 and P2; (center left) their associated hyperschema H(P1,P2) with nodes numbered in breath-first order; (center right) all the potential offspring applying homologous crossover to parents P1 and P2 (the part in bold means alternative content of the tree; in this case there are 5 independent binary alternatives, resulting in 32 possible offspring); (bottom left) a valid crossover mask for the breath-first ordered homologous crossover, and (bottom right) an invalid one.

nodes passed to the offspring from parent 1 (marked with 'X') is not an uninterrupted sequence (since X-marked nodes are: 1, 2, 4, 7 and 8).

Theorem 13.3.2. *Ordered homologous crossovers are one-point geometric crossovers.*

Proof. We prove it by showing that the offspring $o_1, o_2, \ldots, o_n$ generated respectively by exchanging nodes at position 1, and at positions 1 and 2, and at positions 1, 2 and 3 and so on respect the condition of theorem 13.3.1 to be on a geodesic. It is easy to see that the sequence of offspring is a cumulative sequence of independent syntactic differences. Since the metric SHD is a weighted Hamming distance it is an additive distance on the contribution of independent syntactic differences. So we have $d(a, b) = d(a, o_1) + d(o_1, b)$ and $d(a, b) = d(a, o_1) + d(o_1, o_2) + d(o_2, b)$ and $d(a, b) = d(a, o_1) + d(o_1, o_2) + d(o_2, o_3) + d(o_3, b)$ and so on. So we have $d(a, o_1) + d(o_1, o_2) + \ldots + d(o_{n-1}, o_n) + d(o_n, b) = d(a, b)$. □

The application of one-point geometric crossover to GP trees is instructive because it shows that operators that would have been intuitively understood as reasonable extensions of one-point geometric crossover for binary strings to GP trees, in fact, are not one-point geometric crossovers. This point deserves some attention.

As a general rule, a geometric crossover based on (additive) edit distances to be one-point geometric must have all offspring totally orderable so as to have a sequence of cumulative independent syntactic differences that leads incrementally form one parent to the other parent. Let us refer to this property as "cumulativeness" of one-point geometric crossover.

In syntactic terms, it is, therefore, the cumulativeness characteristic of one-point crossover for binary strings that has been generalized by one-point geometric crossover rather than the fact that adjacent syntactic elements (nodes in the common region in the case of GP trees) are more likely to be passed together than other syntactic elements. This second property depends entirely on which one-point geometric crossover out of the many possible one considers. So for GP trees, if the total order on the common region is chosen in such a way that adjacent nodes in the common region have consecutive numbering, the specific one-point geometric crossover associated to that total order will pass with higher probability adjacent nodes together (nodes with consecutive order) than nodes apart (nodes with non-consecutive order). For example, the breath-first order crossover introduced above has this property of syntactic adjacency and it can be understood as slicing the tree incrementally starting from the root (see also figure 13.11).

Although this property makes this operator more in the spirit of one-point for binary strings, it is not more one-point geometric than any other operator belonging to the ordered homologous crossover family that does not have such a syntactic adjacency property. In fact, from the point of view of the distance they are all equivalent operators, hence geometrically indistinguishable. What makes an operator one-point geometric is therefore its syntactic cumulativeness property (for additive edit distances with independent edit moves), not the adjacency property that happens to be coincidental.

13.3.4 Sequences

In chapter 8 we have introduced the class of alignment-based homologous operators for sequences of variable-length in which parent sequences are aligned optimally before exchanging genetic material using a crossover mask on the alignment. Then, we proved that this class of operators are geometric crossover under edit distance for sequences. One-point and two-point homologous crossovers for sequences are operators belonging to the class of alignment-based homologous operator in which the crossover masks on the alignment are the traditional one-point and two-point masks of the correspondent crossover operators for binary strings. In the following we show that one-point and two-point homologous crossovers for sequences are, respectively, one-point geometric and two-point geometric crossovers.

One-point homologous crossover

Theorem 13.3.3. *One-point alignment-based homologous crossover is one-point geometric crossover.*

Proof. An optimal edit transcript T contains a smallest set E of edit moves to transform parent u in parent v. A mask m selects a subset of edit moves $E_m \subseteq E$ from the transcript T to apply to u and produces the offspring z. z is on a shortest path for the geometricity of homologous crossover. Any homologous crossover operator for which all offspring are generated employing a set of masks m_i that forms a total order (when understood as vectors) generates offspring on a single shortest path between parents. This is because: (i) the sets of edit moves E_{m_i} corresponding to the masks m_i can be totally ordered under inclusion; (ii) the contribution of each edit move to the distance between parents is independent and additive; (iii) hence, when considered in this order, E_{m_i} generate a sequence of offspring z_i on a single shortest path between parents that incrementally leads from u to v. One-point alignment-based homologous crossover uses crossover masks that form a total order: (00...0) < (10...0) < (11...0) < ... < (11...1). So all its offspring are on a single shortest path between parents. Hence it is one-point geometric. □

Two-point homologous crossover

In the following we give an example in which the two-point homologous crossover can be decomposed into a sequence of two one-point homologous crossover. We *conjecture* that this is true in general.

Figure 13.12 shows an example of two-point homologous crossover. $P1$ and $P2$ are the parent sequences, $\bar{P}1$, and $\bar{P}2$ are the aligned parent sequences. The verticals bars in $\bar{P}1$ and $\bar{P}2$ indicate the crossover points. $\bar{O}1$ and $\bar{O}2$ are the offspring sequences obtained by recombining the aligned parents, and $O1$ and $O2$ are the offspring sequences after removing the dashes.

```
P1: a g c a c a c a g

P2: a c a c a c t a t


--
P1: a g c|a c a c -|a g


--
P2: a - c|a c a c t|a t
--
O1: a g c a c a c t a g
--
O2: a - c a c a c - a t


O1: a g c a c a c t a g

O2: a c a c a c a t
```

Figure 13.12: Example of two-point two-point homologous crossover (for sequences).

Figure 13.13 shows an example of two-point homologous crossover as a sequence of two one-point homologous crossovers. We have used the same parents $P1$ and $P2$ and the same crossover points in both examples in figures 13.12 and 13.13. Since the offspring sequences $O1$

and $O2$ in the first example equal the offspring sequences $O5$ and $O6$ in the second example, in this specific case the two-point homologous crossover can be decomposed in a sequence of two one-point homologous crossovers. This happens because the offspring interleaved with dashes $\bar{O}3$ and $\bar{O}4$ of optimally aligned parents $\bar{P}1$ and $\bar{P}2$ are optimally aligned. So $\bar{O}3$ and $\bar{O}4$ equal the optimally aligned parents $\bar{O}3'$ and $\bar{O}4'$ of the second one-point homologous crossover. We conjecture that in general for one-point homologous crossover optimal aligned parents produce optimally aligned offspring. If this conjecture is correct, two-point homologous crossover is a two-point geometric crossover.

```
--
P1: a g c|a c a c - a g
--
P2: a - c|a c a c t a t
--
O3: a g c a c a c t a t
--
O4: a - c a c a c - a g


O3: a g c a c a c t a t

O4: a c a c a c a g

--
O3': a g c a c a c t|a t
--
O4': a - c a c a c -|a g
--
O5: a g c a c a c t a g
--
O6: a - c a c a c - a t


O5: a g c a c a c t a g

O6: a c a c a c a t
```

Figure 13.13: Example of two-point homologous crossover as a sequence of two one-point homologous crossovers.

Table 13.1: One-point geometric crossover for sets.

a b c d e	1 2 3 4 5
{a,b}	1 1 0 0 0
{b,c,d}	0 1 1 1 0
{a,b}	1 1 0 0 0
{b}	0 1 0 0 0
{b}	0 1 0 0 0
{b,c}	0 1 1 0 0
{b,c,d}	0 1 1 1 0
{b,c,d}	0 1 1 1 0

13.3.5 Sets

In chapter 6, we have seen that there is a duality between the geometric crossover under Hamming distance for binary strings and the geometric crossover under ins/del edit distance for sets. In fact, these two crossovers, although being based on two different solution representations, are associated to isomorphic metric spaces, hence they are completely equivalent. Using the duality, in the following we will show the equivalent for sets of the one-point crossover for binary strings.

Let us consider two binary strings of length 5, $p_1 = 11000$ and $p_2 = 01110$. Let $U = \{a, b, c, d, e\}$ be the universal set. The corresponding sets of p_1 and p_2 are $s_1 = \{a, b\}$ and $s_2 = \{b, c, d\}$. In table 13.1, in the column on the right the generating sequence (12345) (first row) is applied on the parent strings p_1 and p_2 (second and third rows) producing the sequence of offspring in the remaining rows. The column on the left reports for each row the set corresponding to the binary string on the right, except for the first row that reports the elements of the universal set ordered according to the corresponding order of the elements in the generating permutation. So the sets in the table are the offspring sets of one-point crossover for sets applied to parents s_1 and s_2.

Removing the repetitions the sequence of offspring sets is: $\{a, b\}, \{b\}, \{b, c\}, \{b, c, d\}$. Notice that this sequence transforms s_1 into s_2 a move at a time using the ins/del edit move. This is the interpretation of one-point crossover for sets.

Let us contrast one-point crossover with uniform crossover for sets. As we have seen in chapter 6, uniform crossover for sets has offspring o satisfying the relation $s_1 \cap s_2 = \{b\} \subseteq o \subseteq$

$s_1 \cup s_2 = \{a, b, c, d\}$. Dually, in terms of binary strings this is equivalent to saying that uniform crossover on $p_1 = 11000$ and $p_2 = 01110$ returns offspring in the schema $*1**0$, in which $*$ appears at the positions in which p_1 and p_2 differ. So, one-point crossover generates only a subsets of the offspring of uniform crossover and in our specific example one-point crossover cannot generate: $\{b, d\}, \{a, b, c\}, \{a, b, d\}, \{a, b, c, d\}$.

13.4 Summary

In this chapter we have extended the geometric framework with the generalization of one-point crossover to any metric space. This extension is interesting for a number of reasons. Firstly, the definition of one-point crossover seems so tightly dependent on the fact that the underlying representation is a vector, that one may believe that its generalization to generic metric spaces, hence to any representation, is simply not possible. This would constitute a natural limit to the geometric interpretation of crossover. The fact that one-point crossover indeed generalizes and has a simple geometric characterization gives further depth to the geometric interpretation of crossover. Secondly, there are a number of recombination operators that extend the notion of one-point crossover to different solution representations. These extensions are based on analogy and intuition and are adaptations of the original one-point crossover to the special characteristics of the new representation. One-point geometric crossover makes rigorous the notion of one-point crossover across representations and formalizes the intuition behind it. This leads us to a third point. Since one-point geometric crossover is well-defined for any representation, it can be used to generate new one-point crossovers for new representations without involving any element of arbitrariness. Fourthly, since we now have a formal definition of one-point crossover that holds for all representations, we can use it to derive properties common to all one-point crossovers. We will see this in chapter 14.

Chapter 14

Non-geometric Crossover

Showing that a given recombination operator is a geometric crossover requires finding a distance for which offspring are in the metric segment between parents. However, proving that a recombination operator is not a geometric crossover is much more difficult because it requires excluding that one such distance exists. It is, therefore, very difficult to draw a clear-cut line between geometric crossovers and non-geometric crossovers. In this chapter we develop some theoretical tools to solve this problem and we prove that some well-known operators are not geometric.

14.1 Introduction

The remainder of this chapter is organized as follows. In section 14.2, we show that the definition of geometric crossover can be cast in three equivalent, but conceptually very different, forms: functional, abstract and existential. When proving geometricity the existential form is the relevant one. We use this form also to show why proving non-geometricity of an operator looks impossible. In section 14.3, we develop some general tools to prove non-geometricity of recombination operators. In section 14.4, we prove that three recombination operators for vectors of reals, permutations and syntactic trees are not geometric. Importantly, this implies that there are two *non-empty* classes of representation-independent recombination operators: geometric crossovers and non-geometric crossovers. Finally, in section 14.5 we prove that the class of one-point geometric crossover is a proper subclass of geometric crossover. That is, there exist geometric crossovers which are not one-point crossovers with respect to any choice of metric.

14.2 Interpretations of the definition of geometric crossover

In chapter 3, we have defined geometric crossover as a function of the distance d of the search space. In this section we take a closer look at the meaning of this definition *when the distance d is not known*. We identify three fundamentally different interpretations of the definition of geometric crossover. We show that proving that a recombination operator is non-geometric may be impossible.

14.2.1 Functional interpretation

Geometric crossover is function of a *generic distance*. If one considers a specific distance one can obtain a specific geometric crossover for that distance by functional application of the definition of geometric crossover to this distance. This approach is particularly useful when the specific distance is firmly rooted in a solution representation (e.g., edit distances). In this case, in fact, the specification of the definition of geometric crossover to the distance acts as a formal recipe that indicates how to manipulate the syntax of the representation to produce offspring from parents. This is a general and powerful way to get new geometric crossovers for any type of solution representation. For example, given the Hamming distance on binary strings, by functional application of the definition of geometric crossover we obtain the family of mask-based crossover for binary strings. In particular, by functional application of the definition of uniform geometric crossover one obtains the traditional uniform crossover for binary strings.

14.2.2 Abstract interpretation

The second use of the definition of geometric crossover does not require to specify any distance. In fact, we do apply the definition of geometric crossover to a generic distance. Since the distance is a metric that is a mathematical object defined axiomatically, the definition of geometric crossover becomes an axiomatic object as well. This way of looking at the definition of geometric crossover is particularly useful when one is interested in deriving general theoretical results that hold for geometric crossover under any specific metric. We will use this abstract interpretation in section 14.3 to prove the inbreeding properties that are common to all geometric crossovers.

14.2.3 Existential interpretation

The third way of looking at the definition of geometric crossover becomes apparent when the distance d is not known and we want to find it. This happens when we want to know whether a recombination operator RX, defined operationally as some syntactic manipulation on a specific representation, is a geometric crossover and for what distance.

Notice that the definition of geometric crossover applies at two distinct levels at the same time: (a) at a representation level, as a manipulation of candidate solutions, and (b) at a geometric level, on the underlying metric space based on a geometric relation between points. This highlights the inherent *duality* between these two worlds: they are based on the *same* search space seen from different viewpoints, from the representation side and from the metric side.

A definition of geometric crossover in which the duality is made explicit is the following: a recombination operator RX defined operationally on the syntax it manipulates to generate the offspring from their parents is a geometric crossover if there exists a metric space on which RX can be also defined as geometric crossover using the functional definition of geometric crossover. If a recombination operator, defined operationally, does not induce any metric space on which it can be defined as geometric crossover, then it is a non-geometric crossover.

We can cast the previous definition in an existential form: a recombination RX is a geometric crossover if for any choice of the parents all the offspring are in the metric segment between them for some metric.

The definitions above are equivalent because if such a metric exists the operator RX can be defined as geometric crossover on such a space. On the other hand, if an operator is defined on a metric space as geometric crossover in a functional form, such a space exists by hypothesis and offspring are in the segment between parents under this metric by definition.

14.2.4 Geometric crossover classes

The functional definition of geometric crossover induces a natural existential classification of all recombination operators into two classes of operators:

- *geometric crossover class* $\mathcal{G}$: a recombination OP belongs to this class if there exists at least a distance d under which such a recombination is geometric: $OP \in \mathcal{G} \iff \exists d : \forall p_1, p_2 \in S : Im[OP(p_1, p_2)] \subseteq [p_1, p_2]_d$.

- *non-geometric crossover class* $\bar{\mathcal{G}}$: a recombination OP belongs to $\bar{\mathcal{G}}$ if there is no distance d under which such a recombination is geometric: $OP \in \bar{\mathcal{G}} \iff \forall d : \exists p_1, p_2 \in S : Im[OP(p_1, p_2)] \setminus [p_1, p_2]_d \neq \emptyset$.

For this classification to be meaningful we need the two classes to be non-empty. In previous chapters we proved that a number of recombination operators are geometric crossovers so $\mathcal{G}$ is not empty. What about $\bar{\mathcal{G}}$? To prove that this class is not empty we have to prove that at least one recombination operator is non-geometric. However, as we illustrate below this is not easy to do.

Let us first illustrate how one can prove that a recombination operator RX is in $\mathcal{G}$. The procedure is the following: guess a candidate distance d, then prove that all offspring of all possible pairs of parents are in the metric segment associated with d. If this is true then the recombination RX is geometric crossover under the distance d *because the operator RX can be defined as a geometric crossover on this space.* If the distribution of the offspring in the metric segments under d is uniform, RX is the uniform geometric crossover for the metric d *because the operator RX can be defined as the (unique) geometric uniform crossover on this space.* If one finds that some offspring are not in the metric segment between parents under the initially guessed distance d then the operator RX cannot be defined as geometric crossover over this space. However, this does not imply $RX \in \bar{\mathcal{G}}$ because there may exist another metric d' that fits

RX and *makes it definable* as a geometric crossover on d'. So, one has to guess a new candidate distance for RX and start all over again until a suitable distance is found.

Although we developed some heuristics for the selection of a candidate distance, in general proving that a recombination operator is geometric may be quite hard (see for example chapter 7 where we considered homologous crossover for GP trees). Nonetheless, the approach works and, in previous chapters, we proved that a number of recombination operators for the most frequently used representations are geometric crossover under suitable distances.

It is evident, however, that the procedure just described cannot be used to prove that a given recombination operator RX is non-geometric. This is because we would need to test and exclude all possible distances, which are infinitely many, before being certain that RX is not geometric. Clearly, this is not possible.

In the next section we build some theoretical tools based on the abstract interpretation of the definition of geometric crossover to prove non-geometricity in a more straightforward way.

14.3 Inbreeding properties of geometric crossover

How could we actually prove non-geometricity? From the definition of geometric crossover based on a generic notion of distance (abstract interpretation, see section 14.2.2), we could derive metric properties that are common to the class of all geometric crossovers and that could be tested without making explicit use of the distance.

Any reference to the distance needs necessarily to be excluded from these properties because what in fact we need to test is the existence of an underlying distance behind a given recombination operator hence we cannot assume the existence of one *a priori*. So the first requirement is that these properties derive from the metric axioms but cannot be about distance. A second requirement is generality: these properties need to be representation-independent so that recombination for any solution representation can be tested. A third and last requirement is that these properties need to be independent from the specific probability distribution with which offspring are drawn from the segment between the parents. In particular, they must encompass also geometric crossovers where offspring are drawn from only part of the segment.

If necessary properties satisfying these requirements existed, testing a recombination operator for non-geometricity would become straightforward: if such operator does not have a property common to all geometric crossovers it is automatically non-geometric. Fortunately, properties of this type do exists. They are the *inbreeding properties of geometric crossover.*

In the following, we introduce three fundamental properties of geometric crossover arising only from its axiomatic definition (metric axioms), hence valid for any distance, any probability distribution and any underlying solution representation. These properties of geometric crossover are simple properties of geometric interval spaces [154] adapted to the geometric crossover. The properties proposed are based on inbreeding (breeding between close relatives) using geometric crossover and avoid explicit reference to the solution representation. In section 14.4, we will make use of these properties to prove some non-geometricity results.

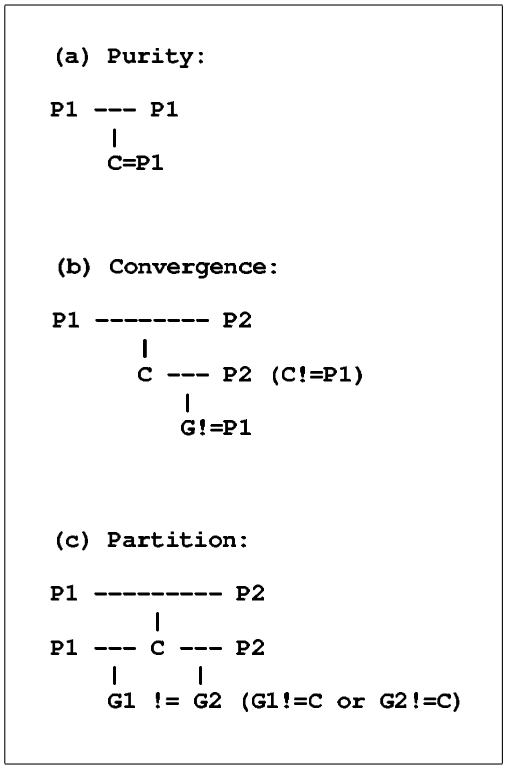
Theorem 14.3.1. *(Property of Purity) If the operator RX is geometric then the recombination of one parent with itself can only produce the parent itself.*

Proof: If RX is geometric there exists a metric d such that any offspring o belongs to the segment between parents s_1, s_2 under metric d: $d(s_1, o) + d(o, s_2) = d(s_1, s_2)$. When the parents coincide, $s = s_1 = s_2$, we have: $d(s, o) + d(o, s) = d(s, s)$ hence, for symmetry and identity axioms of metrics, $d(s, o) = 0$ for any metric. For the identity axiom this implies $o = s$. □

Figure 14.1(a) shows the inbreeding diagram of the property of purity: when the two parents are the same $P1$, their child C must be $P1$.

Theorem 14.3.2. *(Property of Convergence) If the operator RX is geometric then the recombination of one parent with one offspring cannot produce the other parent of that offspring unless the offspring and the second parent coincide.*

Proof: If RX is geometric there exists a metric d such that for any offspring o of parents s_1 and s_2 we have $d(s_1, o) + d(o, s_2) = d(s_1, s_2)$. If one can produce parent s_2 by recombining s_1 and o, it must be also true that $d(s_1, o) = d(s_1, s_2) + d(s_2, o)$. By substituting this last expression in the former one we have: $d(s_1, s_2) + d(s_2, o) + d(o, s_2) = d(s_1, s_2)$, which implies $d(o, s_2) = 0$ and $s_2 = o$ for any metric. □



```
(a) Purity:

P1 --- P1
    |
    C=P1


(b) Convergence:

P1 -------- P2
      |
      C --- P2 (C!=P1)
         |
         G!=P1


(c) Partition:

P1 --------- P2
       |
P1 --- C --- P2
    |     |
    G1 != G2 (G1!=C or G2!=C)
```

Figure 14.1: Inbreeding diagrams.

Figure 14.1(b) shows the inbreeding diagram of the property of convergence: two parents $P1$ and $P2$ produce the child C. We consider a C that does not coincide with $P1$. The child C and its parent $P2$ mate and produce a grandchild G. The property of convergence states that G can never coincide with $P1$.

Theorem 14.3.3. *(Property of Partition) If the operator RX is geometric and* $\mathbf{c}$ *is a child of* $\mathbf{a}$ *and* $\mathbf{b}$*, then the recombination of* $\mathbf{a}$ *with* $\mathbf{c}$ *and the recombination of* $\mathbf{b}$ *with* $\mathbf{c}$ *cannot produce a common grandchild* $\mathbf{e}$ *other than* $\mathbf{c}$*.*

Proof: We have that $\mathbf{c} \in [\mathbf{a}, \mathbf{b}]$, $\mathbf{e} \in [\mathbf{a}, \mathbf{c}]$ and $\mathbf{e} \in [\mathbf{b}, \mathbf{c}]$, from which it follows that $d(\mathbf{a}, \mathbf{c}) + d(\mathbf{c}, \mathbf{b}) = d(\mathbf{a}, \mathbf{b})$, $d(\mathbf{a}, \mathbf{e}) + d(\mathbf{e}, \mathbf{c}) = d(\mathbf{a}, \mathbf{c})$ and $d(\mathbf{b}, \mathbf{e}) + d(\mathbf{e}, \mathbf{c}) = d(\mathbf{b}, \mathbf{c})$. Substituting the last two expressions in the first one we obtain:

$$d(\mathbf{a}, \mathbf{e}) + d(\mathbf{e}, \mathbf{c}) + d(\mathbf{b}, \mathbf{e}) + d(\mathbf{e}, \mathbf{c}) = d(\mathbf{a}, \mathbf{b})$$

Notice that $d(\mathbf{a}, \mathbf{e}) + d(\mathbf{b}, \mathbf{e}) \geq d(\mathbf{a}, \mathbf{b})$ and, so, the previous equation implies $d(\mathbf{e}, \mathbf{c}) = 0$ and $\mathbf{e} = \mathbf{c}$. $\square$

Figure 14.1(c) shows the inbreeding diagram of the property of partition: two parents $P1$ and $P2$ produce the child C. The child C mates with both its parents, $P1$ and $P2$, producing grandchildren $G1$ and $G2$, respectively. We consider the case in which at least one grandchildren is different from C. The property of partition states that $G1$ and $G2$ can never coincide.

Geometric crossovers whose offspring cover completely the segments between their parents, which we will term complete geometric crossovers, have a larger set of properties including extensiveness $(a, b \in Im(UX(a,b)))$ and symmetry $(Im(UX(a,b)) = Im(UX(b,a)))$, which however, are not common to all geometric crossovers.

14.3.1 Relation with forma analysis

Since the inbreeding properties of geometric crossover are related to forma analysis [125] we briefly explain this relation.

Radcliffe developed a theory [126] of recombination operators starting from the notion of *forma* that is a representation-independent generalization of schema. A forma is an equivalence class on the space of chromosomes induced by a certain equivalence relation. Radcliffe describes a number of important formal *desirable properties* that a recombination operator should respect to be a good recombination operator. These properties are representation-independent and are stated as requirements on how formae should be manipulated by recombination operators.

Geometric crossover, on the other hand, is formally defined geometrically using the distance associated with the search space. Unlike Radcliffe's properties, the inbreeding properties of geometric crossover are not desired properties but are properties that are *common* to all geometric crossovers and derive logically from its formal definition only.

It is important to highlight that geometric crossover theory and forma analysis overlap but they are not isomorphic. This becomes clear when we consider what schemata for geometric crossover are. In forma theory, the recombination operators introduced by Radcliffe "respect" formae: offspring must belong to the same formae both parents belong to. A natural generalization of schemata for geometric crossover in this sense are (metric) convex sets: offspring in the line segment between parents belong to all convex sets common to their parents. So *geometric crossover induces a convexity structure over the search space.* A convexity structure is not the same thing as an equivalence relation: convex sets, like equivalence classes, cover the entire space but unlike them convex sets do not partition the search space because they overlap. Interestingly, convex sets seen as schemata naturally unify the notions of inheritance and fitness landscape.

A further advantage of geometric crossover over forma theory is that whereas it is rather easy to define and deal with distances for complex representations such as trees and graphs (using edit distances) it is much harder to use equivalence classes.

14.4 Non-geometric crossovers

In the following we use the properties of purity, convergence and partition to prove the non-geometricity of three important recombination operators: extended line recombination, Koza's subtree swap crossover and Davis's Order Crossover (see, for example, [8] for a description of these operators).

Theorem 14.4.1. *Extended line recombination is not a geometric crossover.*

Proof. The convergence property fails to hold. Let p_1 and p_2 be two parents, and o the offspring lying in the extension line beyond p_1. It is easy to see that using the extension line recombination on o and p_2, one can obtain p_1 as offspring. □

Theorem 14.4.2. *Koza's subtree swap crossover is not a geometric crossover.*

Proof. The property of purity fails to hold. Subtree swap crossover applied to two copies of the same parent may produce offspring trees different from it. □

Theorem 14.4.3. *Davis's Order Crossover is non-geometric.*

Proof. The *convergence property does not hold* in the counterexample in Figure 14.2 where the last offspring coincides with parent 2. □

What are the implications of knowing that these operators are *not* geometric? The first one is that one is not tempted to try to prove its geometricity with yet another distance.

A second immediate and fundamental consequence of knowing that an operator is non-geometric is that, since it is not associable with any metric, it is not associable with any simple fitness landscape defined as a height function on a metric space in a simple way. This is bad news

```
Parent 1 : 12.34.567
Parent 2 : 34.56.127
Section  : --.34.---
Available elements in order: 12756

Offspring: 65.34.127
Parent 3 := Offspring

Parent 3 : 6534.12.7
Parent 1 : 1234.56.7
Section  : ----.12.-
Available elements in order: 73456

Offspring: 3456.12.7
Offspring = Parent 2
```

Figure 14.2: Counterexample to the geometricity of order crossover.

for non-geometric crossovers because the alternative to a simple fitness landscape with a simple geometric interpretation is a complex topological landscape with hardly any interpretation for what is really going on.

This leads us to a third important practical consequence. Just knowing that a recombination operator is geometric or non-geometric cannot tell us anything about its performance. The no free lunch theorem [165] rules. However, as we have seen from some preliminary theoretical analysis presented in chapter 3, when the fitness landscape associated with a geometric crossover is smooth, the geometric crossover associated with it may perform better than random search. If this holds true in the general case, this is fundamental for crossover design because the designer studying the objective function can identify a metric for the problem at hand that gives rise to a smooth fitness landscape and then he/she can pick the geometric crossover associated with this metric. This is a good way to embed problem knowledge in the search. However, since this strategy is inherently linked to the existence of a distance function associated with a recombination operator, non-geometric crossovers cannot make use of it.

14.4.1 Possibility of a general theory of evolutionary algorithms

The forth and last consequence of the mere existence of some non-geometric operators is that this implies the existence of two separate classes of operators. We state this in the following as a theorem.

Theorem 14.4.4. *(Existence of non-geometric crossover) The class of non-geometric crossover is not empty. Hence the space of recombination operator is split into two proper classes: geometric and non-geometric crossover.*

This is an important step when developing a theory of geometric crossover because it allows to meaningfully talk about geometric crossover in general without the need to specify the distance associated with it. This in turn has a critical impact on the possibility of a general theory of geometric crossover and of a programme of unification of evolutionary algorithms.

The main danger of a general theory is being too shallow: would such a general theory be able to tell us anything meaningful or only trivialities encompassing all operators could be derived? A theory of all operators is an empty theory because the performance of an EA derives from how its way of searching the search space is matched with some properties of the fitness landscape. Without restricting the class of operators to a proper subset of all possible operators, there is no common behavior, hence there is no common condition on the fitness landscape to be found to guarantee better than random search performance. This is just another way of stating the NFL theorem. So, a theory of all operators is necessarily a theory of random search in disguise. However, since the definition of geometric crossover does not encompass all operators, it is not futile to pursue a general theory of geometric crossover.

In previous chapters, we have found that many recombinations used in everyday practice are geometric. Without being able to prove the existence of some non-geometric crossovers there are two alternative explanations for this happening: (a) the geometric crossover definition is a tautology and the theory built on it a theory of everything hence an empty theory or (b) if there are non-geometric crossovers, this is hardly a coincidence and the class of geometric crossover indeed captured a deep aspect of the class of "real-word" recombinations.

Theorem 14.4.4 is therefore foundational because it implies that the true explanation is (b).

Therefore, a general theory of geometric crossover makes sense because it is not a theory of random search in disguise and the program of geometric unification of evolutionary algorithms makes sense because it is not a mere tautology.

14.5 Non-one-point geometricity

In this section, we will show how to prove that a geometric crossover is not a one-point geometric crossover. We will use a similar technique as the one to prove non-geometricity results. We will then prove that one-point homologous crossover for GP trees is a geometric crossover but not a one-point geometric crossover as anticipated in chapter 13. The mere existence of a single geometric crossover that is not one-point geometric puts on a firm ground the definition of one-point geometric crossover because it shows that the class of one-point geometric crossover is a proper sub-class of geometric crossover. We will see that this sub-class of geometric crossovers is characterized by a specific inbreeding property.

To prove that a recombination operator OP is one-point geometric one has to show that: (i) OP is geometric under a distance d and (ii) that all offspring lay on a single geodesic between each (ordered) pair of parents under distance d. So if one already knows a distance d such as the operator OP is geometric the theorem 13.3.1 can be used to show that the operator OP is one-point geometric.

Proving that a certain operator is not a one-point geometric crossover, however, is more subtle. We have two cases: (i) if a recombination operator OP is known to be non-geometric crossover, than OP is not one-point geometric because one-point geometric crossovers are a subclass of geometric crossovers; (ii) if a recombination operator OP is geometric under a distance d, to prove that OP is not one-point geometric it is not sufficient to show that for some pair of parents its offspring are not all on a single geodesic between them under the distance d. This is because the recombination operator OP could be geometric under another distance d' different from d for which the conditions of theorem 5 apply, hence making it one-point geometric. Notice that the fact that a geometric crossover is geometric under more than a distance is not simply hypothetical: we have seen that, for example, cycle crossover is geometric under both swap distance and Hamming distance restricted to permutations[1].

So to prove that a certain geometric crossover is not one-point geometric crossover we should prove it for all possible distances. However, we may not know all distances associated with a specific geometric crossover, or we may know them all but we may not be able to prove that we know them all since there are infinitely many possible distances. Then, proving non-one-point geometricity of a geometric operator by exhaustively disproving one-point geometricity for each distance associated to it, is not a feasible way.

The way to proceed is to derive a property P that is *common to all one-point geometric crossovers* under any distance and that *can be checked without explicitly using the distance* associated with the geometric crossover. Then, showing that a geometric crossover OP does not have the property P implies that OP is not one-point geometric.

Definition 14.5.1. (Strict inclusion property) Let $OP : S \times S \to S$ be a recombination operator. Let $Im(OP(a, b))$ be the set of all offspring of a and b. Let $D(a, b)$ be the set of all descendants of a and b obtained by the transitive closure of $Im(OP(a, b))$ under OP. We say that OP has the *strict inclusion property* if $\forall a, b : o_1, o_2 \in Im(OP(a, b))$ and $\{a, b\} \neq \{o_1, o_2\}$ imply $D(a, b) \supset D(o_1, o_2)$.

So, if one operator has the strict inclusion property, there is no way to apply the operator OP to any descendants of a and b different from a and b and obtain a or b as offspring.

Theorem 14.5.1. *All one-point geometric crossovers have the strict inclusion property*

Proof. By absurdum, let us suppose that a one-point geometric crossover GX recombines two parents a and b and returns two offspring o_1 and o_2 such as $o_1, o_2 \in [a, b]$, $\{a, b\} \neq \{o_1, o_2\}$ and $a \in D(o_1, o_2)$. Since GX is geometric crossover the closure D is the convex hull co. So $D(a, b) = co(a, b)$ and $D(o_1, o_2) = co(o_1, o_2)$. Hence, $D(a, b) \supseteq D(o_1, o_2)$. We have to prove that $a \in co(o_1, o_2)$ creates a contradiction. a, o_1 and o_2 are on the same geodesic because of the one-point crossover condition. Without loss of generality, let us suppose they appear on the geodesic in that order. So $o_1 \in co(a, o_2)$ because $o_1 \in [a, o_2]$. $a \in co(o_1, o_2)$ means that GX can produce a as descendent of o_1 and o_2; $o_1 \in co(a, o_2)$ means that GX can produce o_1 as descendent of a and o_2. This contradicts the convergence property of geometric crossover that states that a descendant combined with one of its parent cannot produce the other parent unless the descendant coincides with it. So a not in $D(o_1, o_2)$ and $a \in D(a, b)$ imply $D(a, b) \supset D(o_1, o_2)$ $\square$

[1]The segments under swap distance and under Hamming distance do not coincide.

Corollary 14.5.2. *(Non-one-point geometricity) If a geometric crossover GX does not have the inclusion property, it is not one-point geometric crossover.*

So, the strict inclusion property is necessary condition for one-point geometric crossover. It remains as an open question whether the subclass of geometric crossovers with the inclusion property is a strict superclass of one-point geometric crossover class or these two classes coincide.

Corollary 14.5.3. *One-point homologous crossover for GP trees is not one-point geometric crossover.*

Proof. Let us consider two offspring generated by two crossover points at positions x_1 and x_2 such that the subtrees below them do not intersect. For example, two nodes on the boundary of the common region. So we have two offspring o_1 and o_2 that are equal to one parent p except in a single point or single subtree. Now if we do one-point crossover between o_1 and o_2 at position x_1 we obtain p restoring o_1 using the material in o_2. This contradicts the strict inclusion property of one-point geometric crossover. So, one-point homologous crossover is not one-point geometric. □

The mere existence of a single geometric crossover that is not a one-point crossover is important because it proves that *the class of one-point geometric crossover is a proper sub-class of geometric crossover.*

A word of warning: *the strict-inclusion property is characteristic of all one-point geometric crossovers producing only one offspring.* This property does not hold for one-point geometric crossovers that return two complementary offspring. It is easy to see this in the case of traditional one-point crossover for binary strings: by doing one-point crossover of the two complementary offspring one can recreate the two parents.

14.6 Summary

In this chapter, we have shown that the abstract definition of geometric crossover induces two *non-empty* representation-independent classes of recombination operators: geometric crossovers and non-geometric crossovers. This is a fundamental result that puts a programme of unification of evolutionary algorithms and a general representation-independent theory of recombination operators on a firm ground.

Because of the peculiarity of the definition of geometric crossover, proving non-geometricity of a recombination operator, hence the existence of the non-geometric crossover class, is a task that at first looks impossible. This is because one needs to show that the recombination considered is not geometric under *any distance*. However, taking advantage of the different possible ways of looking at the definition of geometric crossover, we have been able to develop some theoretical tools to prove non-geometricity in a straightforward way. We have then used these tools to prove the non-geometricity of three well-known operators for real vectors, permutations, and syntactic trees representations. Also, we have used a similar technique to prove that the class of one-point geometric crossover is a proper subclass of geometric crossover.

In chapter 15, we will start constructing a general theory of evolutionary algorithms based on the abstract interpretation of the definition of geometric crossover. So, this theory will be able to describe the generic behavior of all evolutionary algorithms equipped with a generic geometric crossover. We anticipate that this is a form of convex search.

Chapter 15

Convex Evolutionary Search

In this chapter, we show that all evolutionary algorithms using geometric crossovers perform the same search, convex search, and discuss the significance of this result.

15.1 Introduction

The origin of the differences of the various flavours of evolutionary algorithms is rooted in the solution representation and relative genetic operators.

Evolutionary computation theory is fragmented. One of the main reasons is that there is not a unified way to deal with different solution representations which has led to the development of significantly different theories for different flavors of evolutionary algorithms.

In the previous chapters we have shown that a common mathematical framework is possible. We have proposed genetic operators that are fully defined without any reference to the solution representation and that instead admit a surprisingly simple abstract geometric definition over the search space.

In this chapter we start building a general theory of evolutionary algorithms starting from the definition of geometric crossover. Abstraction is the key to showing that all evolutionary algorithms present a common behavioral core (universality), to showing how to classify them according to their behavioral nature in a hierarchy of abstractions (taxonomy) and finally to showing how to compare evolutionary algorithms based on different representations rigorously (cross-representation comparison).

In section 15.2, we explain the philosophy and motivate our approach to the description of evolutionary algorithms behavior. As necessary preliminary notion, in section 15.3 we briefly describe abstract convexity. In section 15.4, we formally describe the common behavior of all evolutionary algorithms with geometric crossover. In section 15.5, we investigate the origin of the differences of evolutionary algorithms with geometric crossover. This naturally leads to a behavioral taxonomy of evolutionary algorithms based on a taxonomy of their underlying search spaces. We present this in section 15.6. In section 15.7, we discuss the issue of how fitness landscape and formal algorithm relate and claim that the ultimate reason evolutionary algorithms work well is to be found at an abstract level. In section 15.8, we suggest a general condition for good performance for evolutionary algorithms with geometric crossover.

15.2 Unification by abstraction, universality and taxonomy

In this section we explain the philosophy and motivate our approach to the description of evolutionary algorithms behavior.

15.2.1 Unification by abstraction - following the example of modern mathematics

Although the history of mathematics suggests that the great mathematicians of the past achieved their classic results by obeying the classical injunction *divide and conquer*, modern investigators have made most of their notable contributions by adopting a very different principle of *unify and conquer*. The process of unification by abstraction had its initial success in the study of abstract algebra and then in the development of abstract spaces. From these initial ideas, abstraction in mathematics has developed enormously giving rise to category theory, a powerful foundational framework dealing with structures and systems of structures, able to unify in a single abstract theory all the other fields of mathematics.

Quoting E. H. Moore: "The existence of analogies between central features of various theories implies the existence of a general theory which underlies the particular theories and unifies them with respect to those central features."(In [85], page 628.) Those "central features" appear as abstract, primitive notions whose fundamental properties are given in a set of postulates. The framework proposed in this thesis follows this exact idea building up over an axiomatic notion of distance. Unification by abstraction is the leitmotiv of our framework.

The source of the power and fecundity of modern mathematics lies in its formulation of abstract notions and in its employment of the postulational procedure (axiomatic system) that the introduction of such notions naturally suggests. Whereas the talent of mathematicians of the past manifested itself in part by their ability to define fruitful concepts, the outstanding mathematicians of today are distinguished rather by knowing what not to define; that is, their talent is for detecting the basic features that are common to several mathematical disciplines and developing a more general theory in which those features appear as undefined notions.

15.2.2 Abstraction results in unification

By not specifying, or by voluntarily ignoring, the nature of the elements of a set, a theory is obtained that applies to sets of numbers as well as to sets of functions or sets of chairs. Of course, those properties that a set has by virtue of the nature of its elements are not within the scope of a theory of abstract sets. As properties of specific sets, they are lost by abstraction (generalized out of existence), but it has proved of great value to have at hand a theory that yields important information about every one of a numerous class of things, even though the theory by its very nature is inadequate to investigate exhaustively any one member of the class.

In a similar way, geometries and theories of spaces are unified by the process of abstraction applied to the nature of their elements and to certain fundamental relations (for example, "part of") which are postulated as basic in a unified theory. In this way, various classes of abstract spaces are obtained with geometries depending upon those properties (for example, metric spaces) and are selected for investigation.

15.2.3 Utility of an abstract theory

The similarities of various flavours of evolutionary algorithms indicate that an abstract formulation of properties forms the core or skeleton of many diverse genetic operators across representations, and they suggest that the study of those genetic operators might be unified by studying the abstract system obtained by that abstraction. In this manner, all properties that are common to all of those genetic operators can be derived without any reference to the particular nature of any of them, and the applications of such properties to a special representation is obtained merely by reversing the process of abstraction. Of course, the development of the abstract system (the common core), by its very nature, will not yield properties that are peculiar to genetic operators for specific representations. Those properties are reserved for special investigation. The utility of the unification by abstraction procedure must be determined in each instance by a careful assessment of the results that follow. If only trivialities are obtainable, too much terrain has been covered by the abstraction. The extent to which abstract definitions give a fruitful generalization depends largely on how many interesting examples there are of real genetic operator.

15.2.4 Universality and taxonomy

Taxonomy and universality are the words that point the way forward to establish order in the evolutionary computation field according to Chris Stephens [149]. Taxonomy (classification) of evolutionary algorithms should be based on theoretical understanding of the behavior of algorithms rather than based on historical/political issues or on superficial similarities among algorithms. Universality is about seeing different evolutionary algorithms and noting that, although superficially different, they display from certain points of view the same or similar behavior (hence they admit a theory encompassing them all). The periodic table is a great example: each halogen element is different one from the other, but for a large number of properties they exhibit similar, universal behavior (due to the fact that they all have 7 electrons in their outer shell). A "universality class" of systems all share similar properties for a set of variables that

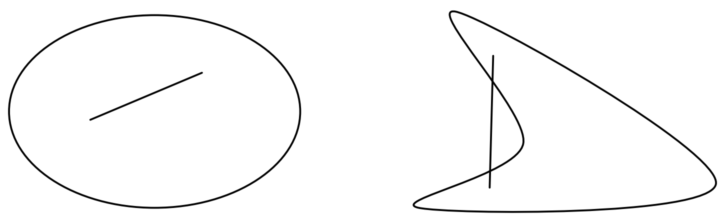


Figure 15.1: Convex set (left) and non-convex set (right).

are universal.

15.3 Abstract convexity

Figure 15.1 shows the classical notion of convex set. To the left a convex set is shown, that is a set in which all the points on the straight line connecting any two points within the set belong to the set. To the right, an example of a set that is not convex is shown.

A natural way to generalize the idea of convex set is by defining a convex set in a metric space requiring that for each pair of points a and b that belong to the set, the segment connecting those points (defined using the distance coming with the metric space) lays all in the set. Given a specific distance, one can, therefore, tell whether a set is convex according to that distance. However, one can define an abstract notion of convex set by requiring that the set is geodesically convex according to any distance that meets the axioms for a metric. An axiomatic approach to geodesically convex sets, therefore, focuses on those properties of convex sets that are independent on the specific distance considered (hence are valid for all distances) and depend only on the convexity *per se*.

The notion of abstract geodetic convexity can be further generalized to the notion of abstract convexity. This does not need a metric to define convex set. Instead, it uses a more elegant definition using sets. In the following, we start from the general definition of abstract convexity, we connect it with abstract geodetic convexity and then we give specific examples of convexities for Euclidean space and Hamming space.

15.3.1 Interval spaces and abstract geodetic convexity

In chapter 12, we gave axiomatic definitions of convex structure, convex sets and convex hull. Interval operators provide a natural method of constructing convex structures [154]. In the following we give an axiomatic definition of interval operator and then show how it connects with convexity and metric spaces.

Let X be a set and let $I : X \times X \longrightarrow 2^X$ be a function with the following properties:

(I1) Extensive law: $a, b \in I(a, b)$

(I2) Symmetry law: $I(a, b) = I(b, a)$

I is called interval operator on X, and $I(a, b)$ is the interval between a and b. The pair (X, I) is called interval space.

The segment operator of a convex structure $(u, v) \mapsto co\{u, v\}$ is an interval operator. Conversely, if (X, I) is an interval space, then the interval convexity $\mathcal{C}$, induced by I on X, is obtained as follows. A subset C of X is interval-convex provided $I(x, y) \subseteq C$ for all x, y in C. If co denotes the convex hull operator of $\mathcal{C}$, then $\forall a, b \in X : I(a, b) \subseteq co\{a, b\}$. The two operators need not to be equal. This is to say an interval I may be non-convex.

Theorem 15.3.1. *The hull of a set A in an interval space is given by $co(A) = \bigcup_{k=1}^{\infty} A_k$ where $A_0 = A$ and recursively $A_{k+1} = \bigcup\{I(a, a')|a, a' \in A_k\}$.*

15.3.2 Abstract convexity and geodetic convexity

Given a metric space $M = (X, d)$ the segment between a and b is the set $[a, b]_d = \{z \in X|d(x, z) + d(z, y) = d(x, y)\}$. The abstract *geodetic convexity* [154] $\mathcal{C}$ on X induced by M is obtained as follow: a subset C of X is geodetically-convex provided $[x, y]_d \subseteq C$ for all x, y in C. If co denotes the convex hull operator of $\mathcal{C}$, then $\forall a, b \in X : [a, b]_d \subseteq co\{a, b\}$. The two operators need not to be equal: there are metric spaces in which metric segments are not all convex.

Theorem 15.3.2. *The (geodetically) convex hull of a set A in a metric space endowed with distance d is given by $co(A) = \bigcup_{k=1}^{\infty} A_k$ where $A_0 = A$ and recursively $A_{k+1} = \bigcup\{[a, a']_d|a, a' \in A_k\}$.*

15.4 Unity of evolutionary search

15.4.1 Formal evolutionary algorithm and abstract evolutionary search

Evolutionary algorithms have a number of parameters, such as the mutation rate just to mention one. Another important parameter is the fitness function of the specific instance of the problem at hand. So, evolutionary algorithms are functions of the mutation rate, of the fitness function and of other parameters. In the same way, we can see an evolutionary algorithm using geometric operators as a function of the metric d. That is, d is a parameter of the algorithm. However, notice the difference in the complexity of the objects passed as parameter: the mutation rate parameter takes values in the interval $[0, 1]$, that is, it is a simple real number, and the metric parameter takes values in the set of metrics, that is, it is a whole space. This interpretation of evolutionary algorithm follows from the functional interpretation of the definition of crossover introduced in chapter 14.

When we do not consider any metric in particular, we can treat an evolutionary algorithm using geometric operators as a formal object with specific formal properties arising from the metric axioms only. This interpretation of evolutionary algorithm follows from the abstract interpretation of the definition of crossover introduced in chapter 14. We refer to an evolutionary algorithm seen according to the latter interpretation as a *formal metric evolutionary algorithm.*

An algorithm can actually be run only when all its parameters have been assigned a value. We call an algorithm with all its parameters specified, a fully-specified algorithm. However, when one has a formal model describing the algorithm, one can use this to describe the behavior of a partially specified algorithm in which same parameter is left unspecified. In other words, *using a formal model one can "run" a partially specified algorithm and describe its abstract behavior, i.e., that part of behavior common to all specific behaviors obtained by assigning all possible specific values to the parameter left unspecified.* In the specific case of evolutionary algorithms, this leads us to the following definition:

Definition 15.4.1. (Abstract evolutionary search): abstract evolutionary search is the behavior of a formal metric evolutionary algorithm.

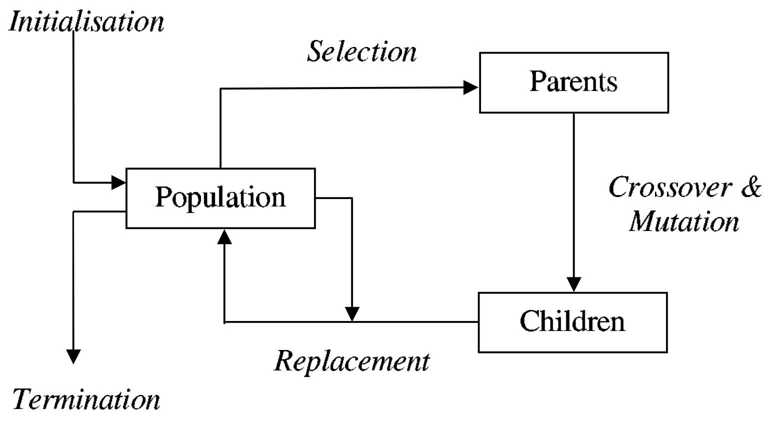


Figure 15.2: Evolutionary algorithm at a population level.

The (abstract) behavior of a formal metric evolutionary algorithm is an axiomatic object itself based on the metric axioms. In the following sections, we will show that the behavior of a formal metric evolutionary algorithm can be profitably described axiomatically.

15.4.2 Genetic operators at a population level

An evolutionary algorithm can be seen as repeating a loop of operations at the population level (see Figure 15.2). The cycle selection-crossover-mutation-replacement can be seen as the sequential functional application of these operators to a population returning another population.

A population is a multi-set whose underlying simple set, obtained by removing the copies of multiple elements, is a subset of the solution set.

In the following, we present a number of population operators that we will use in the next section to prove our main result:

- *identity*: the identity operation is the identity function that takes in input a population and returns the same population.

- *cloning*: the cloning operator is a partially-specified operator that takes in input a population and returns a population in which some of members have been cloned increasing the population size and the frequency of the cloned members. This operator is intentionally left partially-specified without considering what elements are cloned and how many times because this is the level of abstraction suitable to prove our main result later. A partially-specified operator can be seen as a family of completely-specified operators which all match the partial specification. A composition of cloning operations is still a cloning operation. In particular, a composition of cloning operations involving different sub-sets of the population is equivalent to a single cloning operation.

- *selection*: the selection operator is a partially-specified operator that takes in input a population and returns a second population in which some of the elements have been eliminated or reduced in frequency so that the new population has smaller size. Any selection scheme for evolutionary algorithms can be seen as a sequential application of selection and cloning.

- *crossover*: the crossover operator for the population is a partially-specified operator that takes in input a population and returns a second population of offspring obtained by applying any geometric crossover operator (with any probability distribution but respecting the geometric condition of crossover) to pairs of elements in the input population any number of times.

- *mutation*: the mutation operator for the population is a partially-specified operator that takes in input a population and returns a second population of offspring obtained by applying any geometric mutation operator to any element in the input population any number of times.

- *merge*: merge operators is the union operator for multi-set. In particular, the occurrence of a given element in the output multi-set is the sum of the occurrences of the same element in the two multi-sets in input. Any replacement scheme for evolutionary algorithms can be seen as a sequential application of merge parent population and offspring population followed by selection.

The inner loop of any variant of evolutionary algorithm can be seen as a given functional composition of the population operator given above.

15.4.3 Abstract convex search theorem

Definition 15.4.2. (Convex operator): let S be the solution set, an operator $OP : 2^S \rightarrow 2^S$ that takes a subset $P \subseteq S$ as input and returns a subset $OP(P) \subseteq S$ is a convex operator iff $\forall P \subseteq S : OP(P) \subseteq co(P)$

The definition of convex operator extends to multisets substituting multisets with their underlying sets. The definition of convex operator extends to stochastic operators by substituting $OP(P)$, that is a random variable, with its support set $Im(OP(P))$ that is the set of elements that have probability non-zero to be returned by $OP(P)$.

Theorem 15.4.1. *The composition of convex operators is a convex operator.*

Proof. Let OP and OP' be two convex population operators and $OP'' = OP' \circ OP$. To prove that the composition of two convex operators is a convex operator we need to prove that $\forall P : OP(P) \subseteq co(P) \wedge OP'(P) \subseteq co(P) \longrightarrow OP''(P) = OP'(OP(P)) \subseteq co(P)$. By definition of convex population operator, it follows $OP(P) \subseteq co(P)$ and $OP'(OP(P)) \subseteq co(OP(P))$. From the property of convex hull $OP(P) \subseteq co(P)$ implies $co(OP(P)) \subseteq co(P)$. Hence, $OP'(OP(P)) \subseteq co(P)$. □

Theorem 15.4.2. *(Convexity of genetic operators at a population level) Selection, Crossover and Replacement are convex population operators.*

Proof. Selection: the simple set P' of the population obtained by the selection operator $OP_{SEL} : P \longrightarrow P'$ is a subset of the simple set P of the input population to the selection, that is $OP_{SEL}(P) = P' \subseteq P$. Since $P \subseteq co(P)$ then $OP_{SEL}(P) \subseteq co(P)$. Selection is a convex operator.
Crossover: in the crossover operator $OP_{XO} : P \longrightarrow C$ every offspring in C is in the segment between two parents in P. For geodesic convexity, every x, y in P we have $[x, y] \subseteq co(P)$, hence $OP_{XO}(P) \subseteq co(P)$. The crossover operator is a convex operator.
Replacement: the replacement operator $OP_{REP} : P \times C \longrightarrow P$ merges the population P and the children C to produce the new population P'. So $P' \subseteq P \cup C$. We want to prove that when $C \subseteq co(P)$ then $OP_{REP}(P, C) \subseteq co(P)$. Since $C \subseteq co(P)$ then $co(P \cup C) = co(P)$ and $P' = OP_{REP}(P, C) \subseteq co(P)$. The replacement operator is a convex operator. □

Theorem 15.4.3. *(Abstract convex evolutionary search) Let P_n be the population at time n. For any evolutionary algorithm repeating the cycle selection, crossover, replacement we have $co(P_{n+1}) \subseteq co(P_n) \subseteq \cdots \subseteq co(P_1) \subseteq co(P_0)$*

Proof. The compound operator $OP = OP_{REP} \circ (OP_{ID}, OP_{XO} \circ OP_{SEL})$ that is equivalent to the sequential application of selection, crossover and replacement is a convex operator (OP_{ID} is the identity operator that outputs its own input). This is because OP_{SEL} and OP_{XO} are convex operators hence $OP_{XO} \circ OP_{SEL}$ is a convex operator for the composition of convex operators theorem. Hence OP_{REP} is also a convex operator because its second argument $OP_{XO} \circ OP_{SEL}$ is in the convex hull of its first argument OP_{ID} . Hence, for the composition of convex operators theorem, OP is a convex operator. Since $P_{n+1} = OP(P_n)$ then $P_{n+1} \subseteq co(P_n)$ and consequently $co(P_{n+1}) \subseteq co(P_n)$. Then the chain of nested inclusions is true by induction.

□

In section 15.4.5 we will discuss the cases for which the strict inclusion holds.

Theorem 15.4.3 is very general. *An evolutionary algorithm using geometric crossover with any probability distribution, any kind of representation, any problem, any selection and replacement mechanism, does the same search: convex search.* Population size can vary over time and evolutionary search is still convex.

Theorem 15.4.4. *Population mutation operator is not a convex operator.*

Proof. Every convex operator returns points within the convex hull of the input set. The convex hull of a single point is the single point itself. So, when the input set includes a single point, the output set of any convex operator is the point itself. Mutation applied to a single point p produces points different from p, hence it is not a convex operator. □

When applying mutation after crossover the resulting offspring are either (i) within the convex hull of the selected parents like in the case of crossover or (ii) within the convex hull of the previous population but outside the convex hull of the selected parents or (iii) outside the boundaries of the convex hull of the previous population. In case (iii) the evolutionary algorithm does not perform a convex search. However, applying only a micro-mutation or a macro-mutation with little probability the resulting algorithm performs quasi-convex search. The effect of mutation has a stronger impact when the population has (almost) converged to a fix-point and even a small mutation generates offspring outside the convex hull of the population.

15.4.4 Visualization of the abstract convex search theorem

Theorem 15.4.3 applies to all metric spaces. It gives an abstract geometric description of the search that does not depend on any specific distance. In the following, we visualize the abstract convex search for the specific case of the 2-dimensional Euclidean space. This leads to a very simple geometric and useful description of it which illustrates geometrically why the theorem 15.4.3 holds. Indeed, the geometric reason this theorem holds for the 2-dimensional Euclidean case is the same as the reason it holds for generic metric spaces.

In Figure 15.3, we show an example of crossover search. The hollow circles represent individuals in the population at time n, solid circles represent selected individuals (parents), while crosses represent individuals in the populations at time $n+1$. The thin solid lines connecting hollow circles represent the boundaries of the convex hull formed by individuals in the population at time n. The thick solid lines connecting solid circles represent the boundaries of the convex hull formed by selected individuals from the population at time n. The solid lines connecting crosses represent the boundaries of the convex hull formed by individuals in the population at time $n+1$. The broken lines connecting solid circles include all possible offspring of the selected individuals from the population at time n by applying geometric crossover (offspring are in the segment).

Notice that in the absence of mutation the convex hull of the population at time $n+1$ is completely included in the convex hull of the population at time n. So, the convex search is restricted to the convex hull of the initial population. It is now evident how all evolutionary algorithms using geometric crossover search. Notice that in the picture we show the search in a Euclidean space. However, this way of searching extends to generic metric spaces, for which the theory of convex polygons in the Euclidean space generalizes naturally giving rise to more abstract notion of convexity (geodetic convexity). Also, notice that the convex hull formed by binary strings under Hamming distance consists of all the offspring generated by doing gene-pool recombination [113].

15.4.5 Weak convergence

The following corollary is an immediate consequence of the convex evolutionary search theorem (theorem 15.4.3).

Corollary 15.4.5. *(Fixed-point convergence) The formal metric evolutionary algorithm with geometric crossover converges to a fixed-point or stops converging (weak convergence).*

Proof. Let us consider an evolutionary algorithm with crossover. For the abstract convex evolutionary search theorem, the convex hulls of the populations at the successive iterations form a nested chain of inclusions. So, the search never diverges. Since the inclusions in the nested chain are not strict, the search either (i) converges to a fixed-point, or (ii) stops its convergence

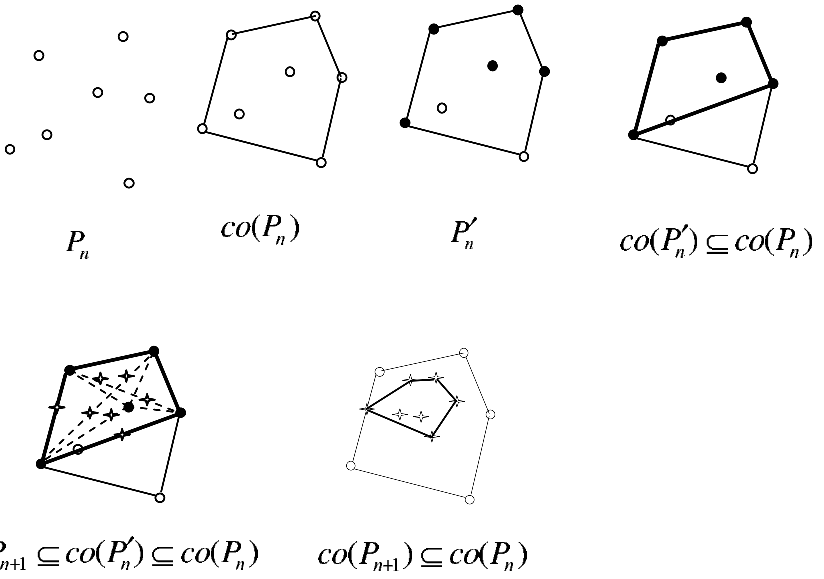

P_n
$co(P_n)$
P'_n
$co(P'_n) \subseteq co(P_n)$
$P_{n+1} \subseteq co(P'_n) \subseteq co(P_n)$
$co(P_{n+1}) \subseteq co(P_n)$

Figure 15.3: Evolutionary algorithm at a population level.

to a single set of points or (iii) stops its convergence to an orbit of sets of points that have the same convex hull. □

In the following, we comments on the three possible types of convergence.

The search stops its convergence not to a fixed-point at time n if $\forall i > n : co(P_{i+1}) = co(P_i)$ and the convex hull does contain more than one point. The condition $co(P_{i+1}) = co(P_i)$ is true when $P_{i+1} = P_i$, which happens when the composition of selection, crossover and replacement is the identity operator. For example, this happens when the selection operator is deterministic and selects uniformly all individuals of the current population, the crossover operator returns as offspring its two parents, and the offspring replace their own parents. Notice that $co(P_{i+1}) = co(P_i)$ does not necessarily implies $P_{i+1} = P_i$. There are two cases in which $co(P_{i+1}) = co(P_i)$ and $P_{i+1} \neq P_i$. In the first case $P_{i+1} \cap P_i$ is not empty and in the second case this intersection may be empty:

- First case: let us consider a convex hull in the 2-dimensional Euclidean plane of a set of points S. By changing points in S not in contact with the border of the convex hull, the convex hull does not change. So, different sets of points give rise to the same convex hull. However, changing points of S in contact with the border of the convex hull changes the convex hull. So, any composition of selection, crossover and replacement that does not affect the individuals of the current population on the border of the convex hull of the current population and only affect those inside still lead to $co(P_{i+1}) = co(P_i)$.

- Second case: this case cannot happen in any Euclidean space, so we will consider the 2-dimensional Manhattan plane. In this space, the line segment between two points a and b is the rectangle that has a and b as diagonally opposite corners. However, unlike the Euclidean case, in the Manhattan plane each segment has two pairs of end-points because each rectangle has two pairs of diagonally opposite corners. Let us call the other two corners c and d, then we have that $[a, b] = [c, d]$ and $\{a, b\} \neq \{c, d\}$. Let us call $\{c, d\}$ the complementary endpoints of $\{a, b\}$. As in the Euclidean case, the convex hull of two points coincides with the segment connecting the two points. Let us consider a population of two points a and b and apply deterministic uniform selection and a geometric crossover that given two parents returns their complementary endpoints that are then used to replace the previous population. In this case we obtain two non-overlapping oscillating populations that are not a single-fixed point: $co(P_{i+1}) = co(P_i)$, $P_{i+1} \cap P_i = \phi$ and $co(P_{i+2}) = co(P_i)$. This situation is also possible in the Hamming space.

We have seen that in certain circumstances the convex search does not lead to a fixed-point due to the not strict inclusion in the nested chain of convex hulls of subsequent populations. However, when the selection operator is probabilistic and the population is finite the search converges to a fixed-point because sampling errors (genetic drift) do not allow to maintain indefinitely the conditions to ensure that $co(P_{i+1}) = co(P_i)$ without errors. When these conditions are not met, the only other option to the search is convergence.

15.5 Origin of EA differences

In section 15.4, we have shown that, at an abstract level, all evolutionary algorithms do the same search. In this section, we focus on the origins of the differences in their behavior.

15.5.1 Duality of the search process: geometry vs. representation

A point in the Euclidean space can be represented as a vector of reals using Cartesian coordinates. Geometric operations in the Euclidean space have their dual as algebraic operations on vectors (for example, translation of a point corresponds to adding an offset vector to the vector representing the point). The convex evolutionary search depicted in Figure 15.3 can be described algebraically using the notion of convex combination of vectors: every point in the convex hull of a set of points S, and only points of the convex hull, can be represented algebraically by a linear combination of the coordinates of the points in S where the weights in the linear combination are positive and add up to one.

Analogously, when using a metric firmly rooted on a representation, such as, for example, an edit distance, geometric operations have their dual as syntactic operations on the representation. The search process, then, can be equivalently described in geometric terms or in terms of syntactic operations on the representation. Notice, however, that there is an important asymmetry between the two ways of describing the evolutionary search process: whereas the geometric description is the same for any choice of distance (for the abstract convex search theorem), the description at the representation level takes a different form for each representation and each distance for that representation. In chapter 3, we have already discussed this duality in connection with the definitions of geometric crossover and geometric mutation. However, here we stress that *this duality extends to the whole search process as a consequence of the abstract convex search theorem*.

We illustrate the asymmetry in the duality of the search process in the case of the Hamming space. The syntax of binary strings gives naturally rise to the Hamming space: the Hamming distance is simply the number of syntactic differences between two strings. When we specialize the convex evolutionary search to the Hamming space, its geometric description remains almost identical to the general case: instead of considering the convex hull of a generic metric d, we consider the specific case of the convex hull of the Hamming distance. This has no effect on the logic of the abstract convex search theorem: it holds in the general case, hence it holds in the specific case of Hamming distance. However, the same convex evolutionary search described in syntactic terms, in the case of the Hamming space takes the specific form of syntactic operations on binary strings (as for the Euclidean space it took the specific form of convex combination of real vectors). Interestingly, it turns out that convex sets and convex hulls become schemata when specified for binary strings under Hamming distance and the convex search theorem is intimately related with the schema theorem[1] [59].

In the following, we present the representation counterpart of the geometric description of the evolutionary search in figure 15.3 for the Hamming and Euclidean spaces.

Hamming space

The population at time n is $P_n = \{00000, 01010, 11000, 00010, 01110\}$. The convex hull of the population at time n is $co(P_n) = * * * * 0$ where $* * * * 0$ is a schema by which we mean a shorthand notation for the set comprising all binary strings matching it. For the Hamming space, the convex hull of a set of binary strings is the schema that presents 0 or 1 at a position if all binary strings present, respectively, 0 or 1 at that position. Otherwise that position in the schema is marked with $*$. This can be verified by noticing that (i) the Hamming space gives rise to a product convexity for which it holds that the convex hull of the product space is the product of the convex hull of its factor spaces; (ii) the convex hull of strings a single-bit long (factors) is either $\{0\}$, $\{1\}$ or $\{0, 1\} = *$ for, respectively, sets of all 0s, sets of all 1s and sets including both 0 and 1. The set P'_n of selected parents is a subset of P_n, say, $P'_n = \{00000, 01010, 00010\}$. The convex hull of the selected parents is $co(P'_n) = 0 * 0 * 0$. As for its geometric dual in figure 15.3, the convex hull of the population at time n includes the convex hull of the selected parents: $co(P'_n) \subseteq co(P_n) \Leftrightarrow 0{*}0{*}0 \subseteq {*}{*}{*}{*}0$. In chapter 3, we have seen that any mask-based crossover for binary strings is geometric crossover under Hamming distance. Let us recombine the selected parents P'_n using any mask-based crossover producing the offspring population $P_{n+1} = \{00000, 01010, 00010, 01000, 00000\}$. The offspring population

[1]We do not discuss the relation among the evolutionary convex search theorem and the schema theorem in this thesis.

is in the convex hull of the selected parents: $P_{n+1} \subseteq co(P'_n)$. In representation terms, this means that all offspring match the schema $0*0*0$. From the geometric perspective, this holds because all points of the line segment joining two points in a convex set are in the convex set. So, the parents are in their own convex hull, that is a convex set, and the offspring are in this convex hull because they are taken from line segments joining parents. The reason, from the representation viewpoint, is that a schema that matches all parents necessarily matches all offspring obtained with any mask-based crossover applied to any pair of parents. The convex hull of the offspring population is $co(P_{n+1}) = 0*0*0$. So, we have that the convex hull of the offspring population, the new population, is in the convex hull of the previous population as required by the abstract convex evolutionary search theorem: $co(P_{n+1}) \subseteq co(P_n) \Leftrightarrow 0*0*0 \subseteq ****0$.

Euclidean space

The population at time n is $P_n = \{(0,0),(1,1),(1,5),(5,-2),(6,7)\}$. The convex hull of the population at time n is the set of the vectors that can be generated by a convex combination of the vectors in P_n: $co(P_n) = \{\sum_{i=1,5} v_i \cdot a_i | v_i \in P_n; \sum_{i=1,5} a_i = 1; a_1 \ldots a_5 \geq 0\}$. The convex hull so defined for the Euclidean plane corresponds to the set of points within the smallest polygon comprising the points in P_n. The set P'_n of selected parents is a subset of P_n, say, $P'_n = \{(0,0),(1,1),(1,5)\}$. The convex hull of the selected parents is $co(P'_n) = \{(0,0)\cdot a_1 + (1,1)\cdot a_2 + (1,5)\cdot a_3 | a_1 + a_2 + a_3 = 1; a_1, a_2, a_3 \geq 0\}$. The convex hull of the population at time n includes the convex hull of the selected parents: $co(P'_n) \subseteq co(P_n)$. This can be seen from their convex combinations: the convex combination of $co(P'_n)$ can be seen as the convex combination of $co(P_n)$ with the additional constraint that a_4 and a_5 are zero. In chapter 4, we have seen that the geometric crossover for the Euclidean plane corresponds to picking offspring in the traditional line segment between the parents. In terms of vectors, the line segment is the set of vectors that can be generated by a convex combination of the parents: $[p_1, p_2] = co(\{p_1, p_2\}) = \{p_1 \cdot a + p_2 \cdot (1-a) | 0 \leq a \leq 1\}$. Let us recombine the selected parents P'_n using line crossover producing the offspring population $P_{n+1} = \{(0.5,0.5),(0.5,2.5),(1,4),(0.1,0.1),(0.1,0.5)\}$. The offspring population is in the convex hull of the selected parents: $P_{n+1} \subseteq co(P'_n)$. In terms of vectors this can be seen by noticing that the convex combination of two parents that returns the offspring can be seen as the convex combination of $co(P'_n)$ with the additional constraint that all coefficients, except those of the two parents are zero. The convex hull of the offspring population is $co(P_{n+1}) = \{\sum_{i=1,5} v_i \cdot a_i | v_i \in P_{n+1}; \sum_{i=1,5} a_i = 1; a_1 \ldots a_5 \geq 0\}$. We have $co(P_{n+1}) \subseteq co(P_n)$, as required by the abstract convex evolutionary search theorem, because we have seen already that $co(P'_n) \subseteq co(P_n)$, and the smallest convex polygon comprising the points P_{n+1} is completely included in any other convex polygon comprising the same points: so, $P_{n+1} \subseteq co(P_{n+1}) \Rightarrow co(P_{n+1}) \subseteq co(P'_n)$. For the duality between convex combinations and convex polygons, the last relation can be proven using convex combinations instead of polygons, but it is more complicated.

15.5.2 Space properties determine search properties

We have seen in section 15.4.1 that since the geometric operators are functions of the distance of the search space, the metric formal evolutionary algorithm is a function of the distance of the search space, too. So, the distance is a parameter of the algorithm. When comparing the same algorithm for two different values of a parameter, for example the mutation rate, while keeping all the other parameters unchanged, it is legitimate to conclude that any change in the search behavior is a direct consequence of the change in the parameter because we treat the search algorithm as a function of it. In the same way, the algorithm being a function of the distance (parameter) allows us *to compare the same algorithm on different search spaces* (different values of the parameter) so admitting *rigorous comparisons* among different representations that seemed before impossible or at best possible only in a vague analogical/metaphorical form. Unlike, the mutation rate parameter, which takes values in $[0, 1]$, the distance parameter takes metric spaces as values that are much more complex objects and, in particular, present a number of geometric properties. An important consequence of this is that the specific search properties of geometric operators and the *specific properties of the behavior* of the metric formal evolutionary algorithm for specific (metric) spaces mirror, and have their ultimate causes in, the *peculiar geometric properties of the underlying spaces.*

In the following we illustrate this concept by an example. To do this, first we need to introduce the notion of *bias* of an operator. Let OP be a search operator, and let $z = OP(p_1, p_2, \ldots, p_n) \in S$ be the offspring where the p_i's are the parents in S.

Definition 15.5.1. We say that a search operator is *unbiased* if when we choose parents independently and uniformly in the solution set S we obtain offspring uniformly at random in the solution set S. In formulas: OP is unbiased if $p_i \sim U(S)$ and independent implies $z = OP(p_1, p_2, \ldots, p_n) \sim U(S)$.

Biasedness is closely related with the preference of a search operator for specific areas of the search space. This is an important search property of a search operator: an evolutionary algorithm using that operator, without selection, is attracted to the areas the search operator prefers. Arguably, also when selection is present, the operator bias acts as a background force that makes the search more keen to go towards the areas preferred by the search operator. This is not necessarily bad if the bias is toward the optimum or an area with high-quality solutions. However, it is bad if the bias is toward an area of poor-quality solutions. If we do not know the nature of the bias, we may prefer not to have it, and instead use the bias of selection, which is better understood.

Interestingly, uniform geometric crossover is unbiased on the Hamming space but the very same operator is biased towards the center on the Euclidean space. This is easy to verify by picking random parents in the two spaces and generating offspring in the respective segments. One way to compensate, but not eliminate, for such a bias in Euclidean and Manhattan spaces is using extended-line and extended-box recombinations. Another way to compensate for the bias relies on understanding the origin of this bias, which leads to a justification of the method based on glued spaces presented in chapter 4.

So, what is the origin of the different bias of geometric crossover for the Hamming space and for the Euclidean space? The answer relies on the notion of *isotropy* of the underlaying metric space. Informally, a metric space is *isotropic* if all its points are equivalent: every point has the same properties in terms of distance.

To define formally the notion of isotropic space, we first need to introduce the notion of isometric spaces.

Definition 15.5.2. Two metric spaces $M = (S, d)$ and $M' = (S', d')$ are isometric if there exists a bijective function $g : S \to S'$ which is distance-preserving: so $\forall x, y \in S : d(x, y) = d'(g(x), g(y))$. The mapping g is called isometry.

Definition 15.5.3. A metric space $M = (S, d)$ is isotropic if for any two points $x, y \in S$, it exists a isometry g from M to itself such that $g(x) = y$.

The Euclidean space, or, more precisely, a bounded hyper-rectangular subspace of the Euclidean space, is not isotropic because the boundaries introduce an asymmetry on the types of points: the space has boundaries and has a center, hence each point is different from each other depending on how far from the boundaries and the center it is. Each point has a special status, hence the space is not isotropic. Conversely, in the Hamming space, each point is the center and at the same time is a boundary point, so each point has the same status and the Hamming space is therefore isotropic.

Let us consider now a new metric space obtained by gluing the opposite extremities of the Euclidean hyper-rectangle; in the case of a simple two-dimensional rectangle the space we obtain is the surface of a torus. Notice that the metric associated with this space is not the Euclidean metric anymore, because points that were at opposite sides of the rectangle are close to each other in the new space. However, the metric of the glued space derives from the Euclidean metric. We presented its formal derivation in Section 4.4. The new space is isotropic because the boundaries, that are the origin of the inhomogeneity of types of points in the Euclidean space, are not there anymore. The uniform geometric crossover based on this new space is unbiased because there cannot be bias towards the center given that every point is the center of the new space as in the case of the Hamming space. *It is therefore the isotropy of the space that causes the bias of the search operator to disappear.* We present this idea formally in the following.

Definition 15.5.4. A *distance-based operator* is any search operator whose (conditional) probability distribution depends only on the distances among parents and offspring.

For example, a search operator OP of arity two is distance-based if there exists a function f such that the (conditional) distribution probability of OP, $Pr\{Z = z | P_1 = p_1, P_2 = p_2\}$, equals $f(d(p_1, p_2), d(p_1, z), d(p_2, z))$.

Geometric operators are not necessarily distance-based operators, because although their image set is only function of the distance, their probability distribution does not need to be.

Uniform geometric crossover is a distance-based operator because its conditional probability distribution $Pr\{Z = z|P_1 = p_1, P_2 = p_2\} = \delta(z \in [p_1, p_2]_d)/|[p_1, p_2]_d|$ is only function of the distances $d(p_1, p_2), d(p_1, z), d(p_2, z)$ since both numerator and denominator are.

Theorem 15.5.1. *Any distance-based operator OP of arity two based on an isotropic metric space M is unbiased.*

Proof. Let OP have conditional probability distribution $Pr\{Z = z|P_1 = p_1, P_2 = p_2\}$. The probability of obtaining $z \in S$ is:

$$Pr\{z\} = \sum_{p_1,p_2 \in S} Pr\{p_1, p_2\} \cdot Pr\{Z = z|P_1 = p_1, P_2 = p_2\}$$

OP is unbiased if: $Pr\{p_1, p_2\} = \frac{1}{|S|^2}$ implies $\forall z \in S : Pr\{z\} = \frac{1}{|S|}$
So,

$$\forall z_1, z_2 \in S : \sum_{p_1,p_2 \in S} \frac{1}{|S|^2} \cdot Pr\{Z_1 = z_1|P_1 = p_1, P_2 = p_2\} = \sum_{p_1,p_2 \in S} \frac{1}{|S|^2} \cdot Pr\{Z_2 = z_2|P_1 = p_1, P_2 = p_2\}$$

Simplifying:

$$\forall z_1, z_2 \in S : \sum_{p_1,p_2 \in S} Pr\{Z_1 = z_1|P_1 = p_1, P_2 = p_2\} = \sum_{p_1,p_2 \in S} Pr\{Z_2 = z_2|P_1 = p_1, P_2 = p_2\} \tag{15.5.1}$$

So, to prove that OP is unbiased we have to prove that the summations in equation 15.5.1 are equal given the hypothesis that M is an isotropic space and OP a distance-based operator.
Since M is isotropic: there exists an isometry g such that $g(z_1) = z_2$.
Since g is bijective and the summation is on all pairs of $p_1, p_2 \in S$ we have:

$$\sum_{p_1,p_2 \in S} Pr\{Z_2 = z_2|P_1 = p_1, P_2 = p_2\} = \sum_{p_1,p_2 \in S} Pr\{Z_2 = g(z_1)|P_1 = g(p_1), P_2 = g(p_2)\}$$

Since OP is distance-based $\exists f$ such that

$$\forall p_1, p_2 : Pr\{Z_2 = g(z_1)|P_1 = g(p_1), P_2 = g(p_2)\} = f(d(g(p_1), g(p_2)), d(g(p_1), g(z_1)), d(g(p_2), g(z_1)))$$

Since g is an isometry the previous expression equals: $f(d(p_1, p_2), d(p_1, z_1), d(p_2, z_1)) = Pr\{Z_1 = z_1|P_1 = p_1, P_2 = p_2\}$
This implies that in equation 15.5.1 each summand in the left-hand summation equals a summand in the right-hand summation and vice versa. So the two summations in equation 15.5.1 are equal.

□

Theorem 15.5.1 can be generalised to all distance-based operators of any arity.

It is therefore the isotropy of the space that causes the bias of the search operator to disappear.

The case of the bias of search operators having its cause in a property of the underlying search space rises to an important question: what other properties of the search space are reflected into properties of the search operators? This leads to the investigation on the taxonomy of evolutionary algorithms based on a taxonomy of the underlying spaces.

15.6 Towards a behavioral taxonomy of evolutionary algorithms

In this section, we first define the notion of behavioral taxonomy of evolutionary algorithms and show that it is possible. Then we look at the most important flavors of evolutionary algorithms and identify a few important properties that form the basis of a taxonomy.

15.6.1 Taxonomy based on search space properties

In order to make a taxonomy of evolutionary algorithms that is based on relevant characteristics, ultimately such characteristics must have an impact on the behavior of the evolutionary algorithm. So, ideally we want a behavioral taxonomy of evolutionary algorithms. Since the characterizing parts (the defining properties) of an evolutionary algorithm are ultimately the solution representation plus the genetic operators for that representation, ideally we would like to obtain a rigorous behavioral taxonomy based on these characteristics only. Other elements, such as selection scheme, details of the operators like specific probability distribution, and the specific problem being solved have to be left out in a taxonomy of evolutionary algorithms.

A taxonomy of evolutionary algorithms based solely on the search space has nothing to say about the performance of the algorithm, that are ultimately a function of how the search space connects with the fitness function (fitness landscape), but it is a way to classify the evolutionary algorithms according to the behavioral part that depends on the search space (for example, the crossover bias). Notice that for the NFL, the way an algorithm performs the search (behavior) and its performance are two completely independent “variables” when no problem knowledge is considered in the design of the search algorithm. Therefore, taxonomy of evolutionary algorithms based on their behaviors says nothing about performance in absolute terms.

The structure of the search space affects the actual definitions of the geometric search operators. Indeed, their definitions depend only on the structure of the search space and not on the connection of the space structure with the fitness or the specific fitness distribution. They are responsible for the specific way the evolutionary algorithm searches the space, but are completely disconnected from the fitness. The structure of the search space is the natural choice for taxonomy of evolutionary algorithms because it captures the main behavioral differences between algorithms independently from the problem to which they are applied to.

A taxonomy is a hierarchy of types, the root is the most general type and leaves the most specific types. Intermediate nodes are more specific than their parents nodes and more general than their offspring nodes. Each node can have more than one parent node. In order to define a taxonomy of evolutionary algorithms we need to define what a evolutionary algorithm type is, and what it means for a type to be more specific or more general of another.

15.6.2 Possibility of a behavioral taxonomy of EAs

In section 15.4.1 we have defined the notion of partially-specified algorithm, as an algorithm for which the values of some parameters are left unspecified, and the notion of abstract behavior, as the behavior of a partially-specified algorithm. As examples of these notions, the metric formal evolutionary algorithm is an algorithm in which the metric parameter is left unspecified and the abstract convex evolutionary search is its abstract behavior.

A partially-specified algorithm arises also in the case the values of some parameters are only *partially* specified. This gives naturally rise to an *hierarchy of abstract behaviors*. We illustrate this concept with an example in the following.

The evolutionary algorithm is function of the mutation rate that takes values in $[0, 1]$. If we do not specify the mutation rate we have a partially-specified algorithm pa_1. However, if we specify that the mutation rate is in the range $[0.5, 1]$ we obtain another partially-specified algorithm pa_2, but more specified than the previous one. If we assign to the mutation rate the value 0.8 we obtain a fully-specified algorithm fa. The algorithms pa_1, pa_2 and fa are related through the range of values their mutation rate is left undefined: their ranges form a chain under inclusion, namely, $[0, 1] \supseteq [0.5, 1] \supseteq 0.8$. Let us now consider all possible partially-specified algorithms in which the mutation rate can vary in all possible ranges within $[0, 1]$. This set of partially-specified algorithms carries a natural partial order under inclusion based on the range of unspecified values of the mutation rate. The order is partial because for any two ranges it does not always hold that one is a subrange of the other or *vice versa*. At one end of this partial order there is a single element, the range $[0, 1]$ which includes every other range considered and which corresponds to the most unspecified partially-specified algorithm (among those considered). At the other end of the partial order there are all ranges including a single value for the mutation rate which cannot include any other range than itself, and which correspond to the set of all fully-specified algorithms with respect with the mutation rate. This partial order on partially-specified algorithms can be depicted as a lattice or direct acyclic graph or a hierarchy with root node the partially-specified algorithm associated with the range $[0, 1]$, leaves nodes the fully-specified algorithms, and in between them all the other ranges. The arcs connecting nodes represent the inclusion relationship between two ranges. This is therefore a *hierarchy of partially-specified algorithms based on their level of indetermination* (of the mutation rate parameter).

Let us now consider abstract behaviors. The abstract behavior of a partially-specified algorithm is the set of all specific behaviors obtained by assigning all possible specific values to the parameter left unspecified. This set of behaviors can be equivalently specified implicitly by a characteristic property common to, and exclusive of, all behaviors it comprises.

Form this definition, the abstract behavior of the partially specified algorithm pa_1 is more abstract than, and includes, the abstract behavior of pa_2 in the sense that the set of all specific behaviors of pa_1 is a superset of the set of all specific behaviors of pa_2 since $[0, 1] \subseteq [0.5, 1]$. In turn, the abstract behavior of pa_2 includes the behavior of the fully-specified algorithm fa, that is a concrete behavior. It is, therefore, evident that the set of abstract behaviors considered inherits a isomorphic partial order from their associated partially-specified algorithms. This is therefore a *hierarchy of abstract behaviors that mirrors the hierarchy of partially-specified algorithms based on their level of indetermination.*

The same line of reasoning applies when we substitute the mutation parameter with the metric parameter of the formal metric evolutionary algorithm. In the case of mutation, the partial order arises from the notion of subrange. In the case of metric spaces, the partial order arises from their relative levels of abstraction. So we have that a *hierarchy of abstract metric spaces induces a hierarchy of abstract processes.* In the following we make this idea more precise.

We draw a distinction among search spaces, seen as metric spaces, on the basis of their abstraction level. A search space is a pair $M = (S, d)$ where S is the solution set and d is a metric defined on S that respects the axioms for a metric. When the metric d is left (partially or totally) unspecified we call M *abstract search space* to emphasize that we are dealing with an object that is defined axiomatically but that is only partially specified. When the metric d is completely specified we call M *concrete search space.* A metric space is completely specified when there exists only one metric space logically consistent with its axiomatic definition (up to isomorphism). An abstract search space can be seen as the set of all concrete spaces that are logically consistent with it. We say that an abstract space M is more abstract than an abstract space M' when the set of the concrete spaces logically consistent with M is a superset of the set of concrete spaces logically consistent with M'. We indicate this by $M \supseteq M'$. This relation of inclusion defines a partial order on the set of all possible abstract metric spaces on S. The order relation is partial because, given two abstract metric spaces, the two sets of concrete metric spaces logically consistent with them may not be one subset of the other. So, the partial order relation on the set of all possible abstract metric spaces defines a hierarchy on the set of abstract metric spaces. At the root of the hierarchy lays the generic metric space defined only by the metric axioms; the leaves of the hierarchy are the concrete metric spaces. Analogously to the case of the mutation rate, this hierarchy induces a hierarchy of abstract process through the application of the formal metric evolutionary algorithm.

In the following we relate space properties with the hierarchy of abstract spaces. There are two different ways to look at an abstract metric space. The first is to look at it as the set of all concrete metric spaces that are logically consistent with it (*extensive interpretation*). The second way is to look at it as a set of properties, or requirements, or axioms, defining a class of concrete metric spaces (*intensive interpretation*).

We have seen that the extensive interpretation of abstract metric space leads naturally to a hierarchy of abstract metric spaces based on their levels of abstraction. The same hierarchy can be seen from the point of view of the intensive interpretation of abstract metric spaces. At the top of the hierarchy there is the formal metric space defined only in terms the metric axioms. At intermediate levels there are classes of metric spaces that rely on more axioms (properties) than the fully general metric space at top level. However, such further properties do not fully identify a unique metric space. At the bottom of the hierarchy, we have metric spaces fully defined up to isometry (isomorphism) by the properties they respect. They cannot be specified further more, at least, as long as the notion of distance is concerned. Pairs of classes of metric spaces are connected by arcs to mean that one class includes another. The including class is referred as parent class and the included class is referred as offspring class. The relation of inclusion is the same as for the extensive interpretation, but recast in terms of *defining properties* of metric space classes. So, properties of the parent classes are also properties of the offspring classes. Offspring classes inherit all properties of their parent classes and may have more properties added. More properties in the offspring class implies that the sets of concrete metric spaces logically consistent with the offspring is a subset of the set of concrete metric spaces logically consistent with its parent class.

In section 15.5.2 we have seen that since the behavior of an evolutionary algorithm endowed with geometric operators is a function of the metric associated with the search space, the properties of the search space are reflected into properties of the behavior of the evolutionary algorithm. As an example, we showed that the isotropy of the search space is mirrored in unbiased search operators.

So, *applying the formal evolutionary algorithm on a hierarchy of abstract metric spaces based on their defining properties, we obtain a correspondent hierarchy of abstract search processes based on the behavioral properties associated with their corresponding metric properties.* Also for the hierarchy of abstract processes, the inheritance of properties of parent abstract processes holds because mirrors the inheritance in the hierarchy of abstract spaces. This actually opens up the way to a taxonomy of abstract processes based on their very defining characteristics.

15.6.3 Important properties of spaces that affect the search behavior

In the previous section, we have seen that a behavioral taxonomy of EAs based on the properties of the search space is possible in principle. In the following we present a very preliminary and uncomplete taxonomy.

In practice, as a preliminary step before starting building the taxonomy we need to identify those properties of the search space that have an impact on the behavior of the search algorithm out the infinitely many possible. The properties of the search space that are not reflected into interesting properties on the behavior of the evolutionary algorithm have to be left out of the taxonomy because, in fact, they are irrelevant. So, the question is: what are then relevant properties of the search space?

In the following we present a list of search space properties and explain why they may be reflected into interesting behavioral properties.

Isotropy: we have seen in section 15.5.2 that isotropy of the search space implies unbiased geometric operators based on it. Since the bias is an important behavioral characteristic of an operator the property of isotropy of a search space is certainly relevant for the taxonomy. Combinatorial spaces such as Hamming space and permutation spaces are typically isotropic because the way they are built: the neighborhood move they are based upon is generally highly symmetric and does not depend on the specific characteristics solution is applied to. We have already seen that the Euclidean space is not isometric, but it can be made so by a topological transformation (gluing). The space of genetic programs is not isotropic because the size of the tree affects the number of neighborhood moves.

Cardinality: another very important characteristic is the cardinality of a search space. We can distinguish it in three major categories: finite, countably infinite and uncountably infinite[2]. Combinatorial problems are typically associated with finite spaces, since for each instance of the problem they have a finite number of solutions. Genetic programming is typically associated with countably infinite spaces, since for each instance of the problem there is an infinite number of solutions (if the solution size, the size of the tree, is let free to grow indefinitely). Continuous optimization problems are typically associated with uncountably infinite spaces, since also they have an infinitely many solutions for each instance of the problem. The cardinality of the search space has an effect of the convergence of an evolutionary algorithm, hence on its behavior. Evolutionary algorithms with geometric crossover based on finite spaces, under mild additional precautions always met in practice, converge to a solution in a finite number of iterations. Evolutionary algorithms with geometric crossover based on countably infinite spaces may not converge to a solution in a finite number of iteration. The observed phenomena of indefinite programs growth and bloat are possible because the underlying space is countably infinite. Evolutionary algorithms with geometric crossover based on uncountably infinite spaces, under condition normally met in practice, never converge to a solution in a finite number of iterations. This is because for any two numbers arbitrarily close, it is always possible to pick a number different from them in between them. In practice of continuous optimization, this is normally dealt with considering a population converged if their members are all within a given small distance from each others.

Dimensionality: in those spaces for which a notion of dimension is well-defined, their dimension is very important because it affects the number of iterations to convergence.

[2]The difference between countable infinite and uncountable infinite is that the former can be put into 1:1 relation with the set of natural number, whereas the latter cannot.

Table 15.1: Classification of most common search spaces.

distance	representation	orthogonal	cardinality	isotropic	dimensions
Hamming	Binary strings	yes	finite	yes	finite
Euclidean	Real vectors	yes	uncountably infinite	no	finite
Cayley	Permutations	no	finite	yes	finite
Chomsky	Syntactic trees	no	countably infinite	no	infinite

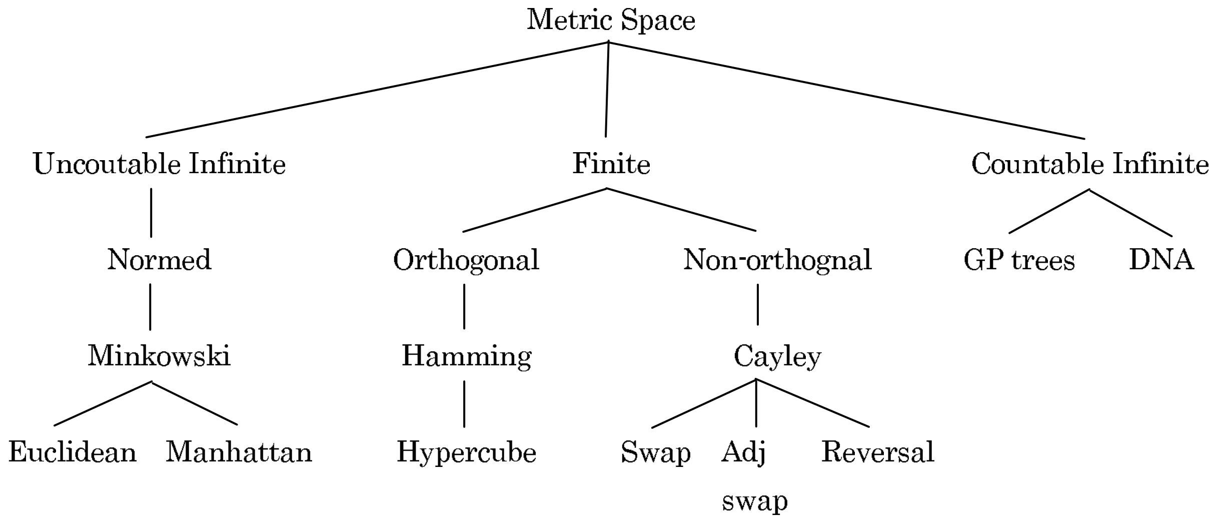


Figure 15.4: Initial taxonomy of search spaces.

Variable-length representation without limit on the solution size are associated with infinite-dimensional search spaces.

Orthogonality[3]**:** by orthogonality we mean that a space is a product of simpler spaces. Hamming space and n-dimensional Euclidean space are orthogonal spaces because they have independent dimensions: one can always change a value at one dimension (an allele) without affecting values in other dimensions. Permutation spaces and GP trees spaces do not have this properties: changing values at one dimension affect values in other dimensions. For example, for permutations, one cannot change a single element and obtain a valid permutation (without repetitions).

In the table 15.1 we summarize how the metric spaces associated with the most common evolutionary algorithm representations relate with the properties listed above. Based on the properties introduced above, we give an initial taxonomy of search spaces in figure 15.4.

The taxonomy presented is very preliminary. To improve it and extend it, it is necessary to find those properties of the search space that have an impact on the behavior of the evolutionary algorithm based on it. There are (at least) three promising ways to find more relevant properties:

1. Looking at important properties of metric spaces, such as completeness, for example, and verify if they correspond to interesting behavioral properties.

2. Since we have shown that evolutionary search is convex search, it makes sense to classify evolutionary algorithms according to characteristics of the convexity of the underlying spaces, such as convexity numbers, for example.

3. Looking at interesting behavioral properties that seem peculiar of a particular representation/operator pair. Then, finding out the distance associated with that operator, if any, and track back what peculiarity of its distance (search space) causes the peculiarity in the behavior.

15.7 Do all EAs work well for the same reason?

The abstract convex evolutionary search theorem (theorem 15.4.3) is important because it is very general, it applies to most of the evolutionary algorithms used in practice and it describes how they all search. However, there is a more fundamental aspect of it that reveals itself only when the abstract convex search is understood together with a fitness landscape: we put forward the hypothesis that all evolutionary algorithms that do convex search work well essentially for the same reason. This reason has only to do with the fact that these algorithms do convex search. In the following, we investigate the relation between convex search and fitness landscape and build an argument for our hypothesis.

The NFL theorem implies that a black-box algorithm must be matched with a problem to perform on average better than random search. When the black-box algorithm is defined on the search space (for example, as function of the distance associated with it like the formal metric evolutionary algorithm), the NFL theorem requires that the black-box algorithm must be matched with the fitness landscape to perform better than random search. In other words, the black-box algorithm works well on fitness landscapes that respect some conditions. If the search algorithm considered is defined using the distance associated with the search space, conditions on the fitness landscape relevant to it are necessarily those that couple distance and fitness in some way.

Since, at an abstract level, all evolutionary algorithms encompassed by the abstract evolutionary search theorem search in the same way, there must be a general condition on the fitness landscape that defines a class of fitness landscapes that matches the evolutionary search at this level of abstraction and guarantees expected performance better than random search. To make sense, the level of abstraction of the condition on the fitness landscape must match the level of abstraction of the evolutionary search: the definition of the condition must be based on the distance of the search space, but it must be meaningful *per se* without referring to any specific distance.

So, whatever could be this condition, it must be the same condition for all evolutionary algorithms because of the unity of their search from the point of view of the distance of the fitness landscape. They all do convex search, so there must be a general condition on the fitness landscape that makes convex search *per se* profitable in terms of performance. This general condition would then apply to all specific convex searches for specific distances.

Although the required abstract level on the fitness landscape condition may be leading to think that it must be a theoretical and somehow exotic condition, the most common measures of smoothness of a fitness landscape, such as the correlation length and fitness-distance correlation, meet successfully the required level of abstraction: (i) they are based on the distance of the fitness landscape and (ii) their measure makes sense without referring to the specific distance used.

However, all measures of smoothness around are known to have counter-examples. They are, therefore, rules of thumb to predict the performance of evolutionary algorithms and cannot be used to define conditions on the fitness landscape that guarantee good performance. The abstract convex evolutionary search theorem opens up a new way of seeking a general condition on the fitness landscape that guarantees performance of evolutionary search: since we know that evolutionary algorithms do convex search we could *infer theoretically* the most general condition on the fitness landscape that makes convex search profitable. This would result in producing a condition without counter-examples, because derived theoretically with this aim. More importantly, it would identify the essential elements of the fitness landscape that affect the performance of the evolutionary search and how these features interact with convex search to produce performance.

What *exactly* this condition is and *how much better than random search* it would guarantee the average performance of the abstract evolutionary search to be are open challenges.

15.8 Convex search and fitness landscape

In the previous section we have argued that, since all evolutionary algorithms are doing the same search, they all must be working well under the same condition on the fitness landscape. In chapter 3 we have given an initial explanation on the reason why the condition on the fitness landscape to consider is "smoothness". Throughout the thesis, we have presented crossover operators designed for specific problems using "smoothness" of the associated fitness landscape as an indicator of goodness of crossover, and corroborated this rule-of-thumb experimentally. In the following, we will present two consequences of jointly considering convex evolutionary search and conditions on the fitness landscape.

Adaptive heritability of crossover

For the weak convergence theorem, we know that geometric crossover pushes individuals in successive populations to get closer and closer. This distinguishes mutation and crossover: whereas for the former the distance between parent and offspring does not change across generations, for the latter the (average) distance between parents and offspring gets smaller and smaller (since the parents are closer, the offspring in the segment between closer parents are also closer to their parents). As seen in chapter 3, on a smooth landscape closer distance between parents and offspring implies stronger heritability of the search operator. So, this means that mutation maintains the same heritability across generations (the same trade-off between heritability and variability), whereas crossover has weak heritability and strong variability in the beginning of a run, and variability is increasingly traded for heritability as the run unfolds.

This characteristic of crossover is useful in terms of performance: in the beginning of a run, strong heritability would be deleterious because the fitness of the offspring would be tied up to the fitness of the parents that are bad (average fitness of all solutions), whereas strong variability (in the extreme, random sampling of the search space) allows to find quickly better than average solutions (the probability of improvement by random search in the beginning is high because it corresponds to the area of the tail of the fitness distribution of the problem after the current parent fitness, which in the beginning is the mean fitness of the problem so making this probability $\frac{1}{2}$). As the search progresses, the increased heritability of crossover becomes beneficial because it counteracts the increased difficulty of producing better offspring than parents, which becomes harder and harder as the fitness of the parents becomes larger and larger (because the area of the tail of a bell-shaped fitness distribution after an above-average fitness value reduces quickly as this value increases).

Concave fitness landscape

We have seen in chapter 3 that, on a smooth landscape, the parent-offspring fitness correlation (the heritability of the search operator) ties together the expected fitness of the offspring to the fitness of their parents. For above average parents, because of the effect of regression toward the mean, the expected fitness of the offspring is *lower* than the average fitness of the parents.

The convex search of an evolutionary algorithm with geometric crossover suggests a class of functions (landscapes) for which we can obtain better performance than for smooth landscapes: the class of concave functions (see Figure 15.5). For the case of n-dimensional Euclidean space, we have the following definition and result.

Definition 15.8.1. A function $f : R^n \rightarrow R$ is strictly concave iff the Jensen's inequality holds for any subset of vectors in its domain. In formulas, let $x_1, x_2, \cdots x_m$ be a set of vectors in R^n and $y = a_1x_1 + a_2x_2 + \cdots + a_mx_m$ a convex combination of $x_1, x_2, \cdots x_m$. Then f is concave iff $f(y) > a_1f(x_1) + a_2f(x_2) + \cdots + a_mf(x_m)$ for any $\{x_i\}$ and any convex combination of them.

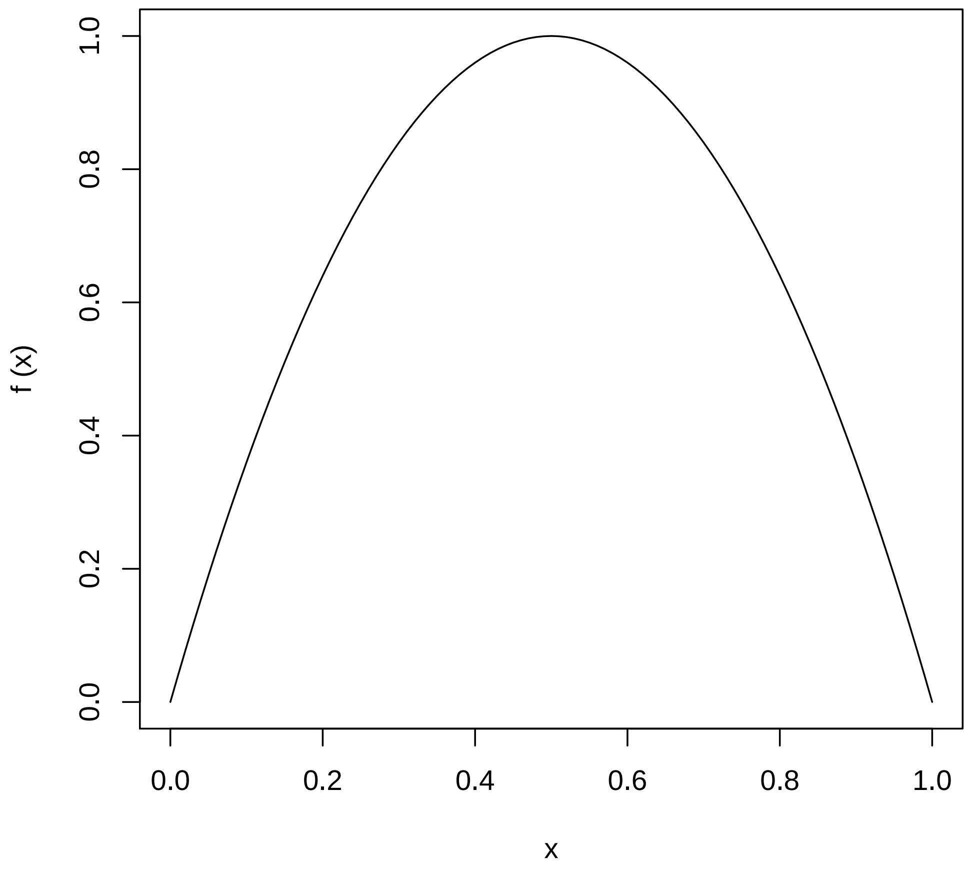


Figure 15.5: Example of concave function.

Theorem 15.8.1. *On a strictly concave fitness function (landscape), the expected fitness of the offspring obtained by picking uniformly at random a point in the convex hull of convexly independent parents is larger than the average fitness of its parents.*

Proof. A set of real vectors is convexly independent when none of the vectors in the set can be expressed as a convex combination of the other vectors in the set. Geometrically, this means that all vectors in the set are corners of the convex hull.

Every distinct choice of weights in a convex combination of convexly independent vectors produces a distinct vector. So, each point in the convex hull can be expressed by only one weighted combinations of its corners. Moreover, each convex combination corresponds to a point in the convex hull. Therefore, there is a one-to-one mapping between the space of weights in the convex combination and the point in the convex hull.

Let us define a partitioning of the convex hull H between convexly independent vectors $x_1, \cdots, x_n$ as follows. Let $W = \{a_1, \cdots, a_n\}$ be a set of convex weights ($\sum a_i = 1, a_i \geq 0$) and let us define a class of vectors C_W in H as all vectors obtained by all possible convex combinations of the vectors $\{x_i\}$ with the weights $W = \{a_i\}$. Over all possible choices of the set of convex weights we obtain a partition of the convex hull: (i) the classes so defined do not overlap because each point in the convex hull is identified by a unique convex combination of convexly independent vectors, so each point belongs to a single class; (ii) these classes completely cover the convex hull because each point in the convex can be expressed as a convex combination, hence it belongs to a class.

For each class it holds that the average fitness of the member of the class is larger or equal to the average fitness of $\{x_i\}$. This can be shown as follows.

In case the set of weights W of a class C_W are all distinct, the number of vectors in the class is $n!$ because this is the number of all possible permutations of the weights W, and each permutation of weights corresponds to a single vector. In case some of the weights in W are identical, the size of the class is smaller than $n!$ because the number of vectors in the class equals the number of all possible distinct permutations with repetitions.

Since f is a concave function we have that: $\forall y_i \in C_W : f(y_i) \geq a_1 \cdot f(x_1) + \cdots + a_n \cdot f(x_n)$. Then summing all over the members of the class C_W, in the case of all distinct weights, we have that $\sum_{y_i \in C_W} f(y_i) \geq A_1 \cdot f(x_1) + \cdots + A_n \cdot f(x_n)$ where $A_1 = A_2 = \cdots = A_n = (n-1)! \cdot (a_1 + \cdots + a_n) = (n-1)!$. By dividing both sides by the number of members of W_C, which is $|C_W| = n!$, we obtain that $\frac{\sum_{y_i \in C_W} f(y_i)}{|C_W|} \geq \frac{\sum f(x_i)}{n}$, which means that the average fitness of the member of the class is larger than the average of the fitness of the parents.

Ultimately, this result holds because the sets of the weights associated with any two x_i in the convex combination across all convex combinations of a class are the same. Then, this results also holds in the case some of the weights are identical.

Since for all classes partitioning the convex hull is true that their average fitness is larger or equal to the average fitness of $\{x_i\}$, then any weighted average (convex combination) of the average fitness of the classes partitioning the convex hull is larger or equal to the average fitness of $\{x_i\}$.

The expected fitness of an offspring picked uniformly at random in the convex hull is by definition the sum of all the fitness of the points in the convex hull divided by the number of this points. This expression can be rewritten as a weighted average of the average fitness of the classes where their weights are proportional to the cardinality of the classes and sum to 1. So, we have that the expected fitness of the offspring is larger or equal to the average fitness of its parents. □

The above result holds more in general: it holds for every probability distribution of the offspring on the convex hull which does not depend on the order of the convex weights, but it depends only on their values, or in other words, for any distribution symmetric with respect to the convex weights.

Theorem 15.8.1 is important because it shows that, unlike the case of smooth landscapes for which the expected fitness of the offspring is always smaller than the average fitness of the parents, on concave functions the expected fitness of the offspring is larger than the average fitness of the parents. So, for smooth functions offspring are better than the parents only occasionally, whereas for concave functions offspring are better than the average fitness of the parents.

To be of general use for the geometric framework, we could think of generalizing the notion of concave function in two directions: (i) from strict concave functions, to function with a concave average trend; (ii) from concave functions on vector spaces, to concave functions on general metric spaces. We discuss both generalizations in the following.

Because of the stochastic nature of evolutionary algorithms, their performance is inherently affected by the average trend of the landscape rather than by specific values at specific points. Therefore, it seems natural to blend the notions of concave function and smooth landscape, and expect that the functions belonging to the resulting class of functions with a "concave trend" (see Figure 15.6) have properties intermediate between concave functions and smooth landscapes. In particular, *we may speculate that the effect of the concavity of the landscape that keeps the fitness of the expected offspring above the fitness of its parents may counteract the effect of the regression, inherent to smooth landscapes, that keeps the fitness of the expected offspring below the fitness of its parents.*

So far we have considered concave functions for vector spaces, which are continuous spaces. Is it possible to generalise the notion of concave function to combinatorial spaces, and, more

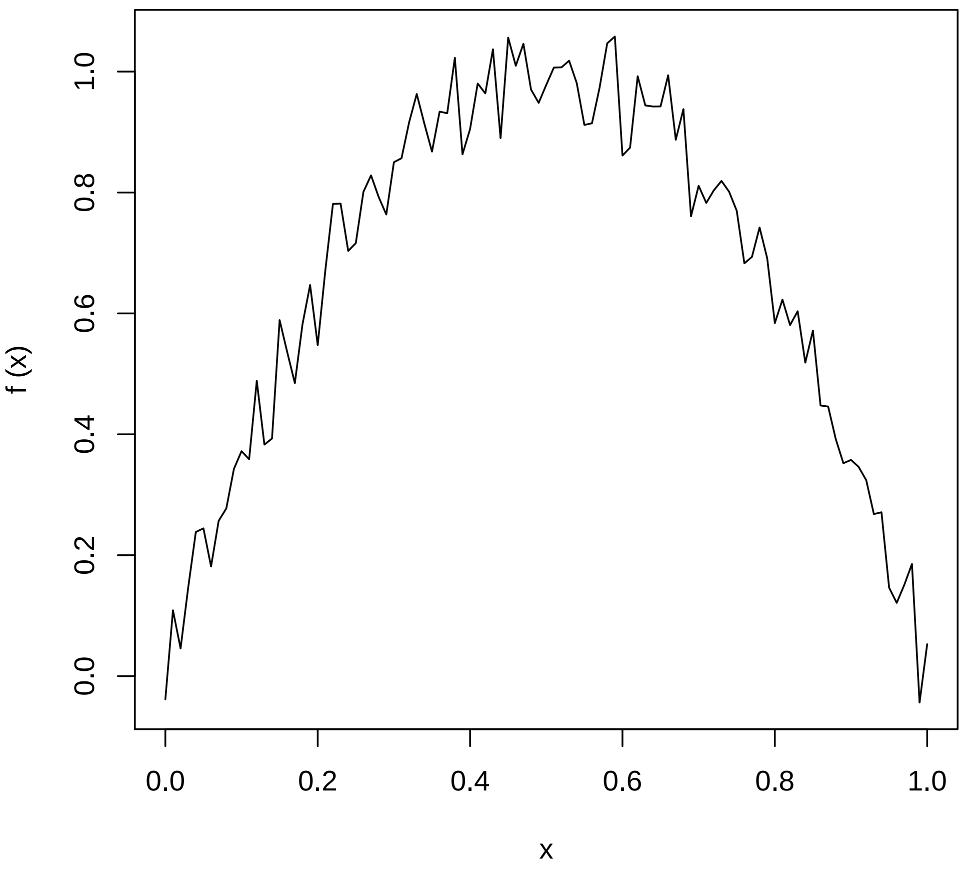


Figure 15.6: Function with a concave trend.

generally, to generic metric spaces retaining the interesting properties we found to hold for vector spaces? There are various ways to generalise the notion of concave function to generic metric spaces, see for example [154]. So, certainly the generalization is possible. However, different generalizations emphasize different aspects of the original notion, and finding a general notion of concave function for which some generalization of theorem 15.8.1 holds might be difficult.

Even granting the existence of a general notion of landscape with a concave trend for which we can prove theoretically that the general class of evolutionary algorithms with geometric crossover perform well, would this notion of concavity of the landscape usable in practice? Can it be actually found and exploited in the practice of evolutionary computation? The answer to this question may be affirmative. In the literature of memetic algorithms (or genetic local search algorithms), there is a well-known notion of global convexity of the fitness landscape[12] that is considered to be beneficial for the performance of memetic algorithms. This notion does not have a completely formal definition. It is stated as follows: when solutions are getting better they are also getting closer to each other; global optima are situated somewhere "in the middle" of this tendency. Global convexity can be detected by a statistical analysis of the landscape[12]. For example, in chapter 11, we studied the global convexity of fitness landscapes of graph partitioning problem. This notion of global convexity is very similar in spirit with the notion of concave landscape discussed above. Interestingly, fitness landscapes naturally associated with many important combinatorial problems have been shown to be globally convex[81].

15.9 Summary

In this chapter we have started developing a general theory of evolutionary algorithms endowed with geometric operators.

We have defined the important notion of abstract behavior and used it to prove our main result on convex evolutionary search. This is a very general result that applies to virtually all types of evolutionary algorithms.

We have then investigated the origin of the differences of evolutionary algorithms. A first source of differences is the representation-geometry duality that tells us that the evolutionary search can be equivalently described in geometric terms and representation terms. However, whereas the geometric description is the same for all spaces, the description based on representation is necessarily representation-specific. We have shown this for the cases of Hamming space and Euclidean space. Different evolutionary algorithms not only look different but, up to a certain extent, behave differently. What is the origin of their behavioral differences? Thank to the formal definition of formal metric evolutionary algorithm, we can now compare rigorously the same algorithm on different representations. We have shown that peculiar properties of the underlying search space are reflected into peculiar properties in the behavior of the search algorithm. As an example, we have shown that the property of isotropy of the search space is reflected into unbiased search by the search algorithm.

This naturally has led us to lay the foundations for a behavioral taxonomy of evolutionary algorithms based on a taxonomy of their underlying search spaces based on their properties.

We then have discussed the issue of how fitness landscape and the search done by the formal metric evolutionary algorithm relate in the light of the NFL theorem. We claimed that the ultimate reason evolutionary algorithms work well is fundamentally the same and it is to be found at an abstract level. Finally, we have suggested that a promising general condition on the fitness landscape that makes it well-matched with convex search is some form of generalized notion of concavity of the fitness landscape.

Chapter 16

Conclusions

In this chapter, we first summarize the achievements of the thesis, then we suggest future work.

16.1 Achievements

In chapter 1, we have motivated the development of a mathematical framework encompassing all evolutionary algorithms. We have claimed that the origin of the differences between evolutionary algorithms is to be found ultimately in the difference of their solution representations, and we have put forward the hypothesis that these differences are only superficial and that there is a deep unity encompassing all evolutionary algorithms. We have also suggested that a unified mathematical framework would be important for both theory and practice: it would be both a starting point for a very general representation-independent theory of evolutionary algorithms, and a formal recipe for the design of search operators for new representations and problems. The stated aim of this thesis was to show that the unification of evolutionary algorithms in a general geometric formal framework is *possible* and *useful*. We believe this goal has been fully met.

In chapter 2, we have presented a broad literature survey on evolutionary computation theory to serve as a general background for the issues dealt with in the thesis.

In chapter 3, we have introduced a geometric framework for evolutionary algorithms that clarifies the connections between representation, genetic operators, neighbourhood structure and distance in the landscape. Thanks to this framework, a novel and general way of looking at crossover (and mutation) that is based on landscape topology and geometry has been put forward. Traditional crossover and mutation for binary strings have been shown to fit our geometric framework. This framework presents a number of additional advantages. The theory is representation independent, and, therefore, it offers a unique opportunity for generality and unification. The theory provides a natural, direct and automatic way of deriving (designing) both mutations *and* crossovers from the neighbourhood structure of landscapes. Conversely, if one adopts our geometric operators, one and only one fitness landscape is induced: that is, we *do not* have a different landscape for each operator, but a common one for both. We have given an initial explanation of why geometric crossover may work well on smooth landscapes by showing that evolvability and heritability of geometric crossover are linked with landscape smoothness modeled within the framework of isotropic random fields. Also, we have shown how problem knowledge and fitness landscape are related and how to embed problem knowledge in the search done by geometric operators.

In chapter 4, we have applied the geometric framework to the real-vector representation. Although the definition of geometric crossover is fairly intuitive for the case of the Euclidean metric, there are other distances suiting real vectors. We have studied formally and in a very general setting geometric crossovers for the family of Minkowski metrics and given efficient algorithms to implement them. We have seen that geometric crossover specified for Minkowski metrics is a biased operator: it produces offspring toward the center of the space with higher probability from uniformly distributed parents. We have explained that the origin of this bias has to be found in the fact that these spaces are non-isotropic: all points are not fully-symmetric in relation to distances. Minkowski spaces can be made isotropic by gluing their opposite sides together and considering the distances associated with these glued spaces. We have then studied formally and in full generality the unbiased geometric crossovers associated with these new spaces. We have also shown that a number of pre-existing recombination operators for real vectors are geometric crossovers under some Minkowski distances. In addition, we have suggested that, building geometric operators associated with non-standard metrics for the real-code representation well-suited to specific problems is a potentially powerful and straightforward way of embedding problem knowledge in the search. This previously has been completely neglected in the specific case of continuous optimization for which the Euclidean distance has been almost always implicitly chosen.

In chapter 5, we have applied the geometric framework to permutations. Permutations differ strongly from binary strings and real vectors, from which our geometric framework has originally arisen. For example, the geometric notion of orthogonality, common to binary strings and real vectors, is not present in permutations. One might wonder, then, whether there would be any hope of applying our geometric framework to permutations. We have shown, not only that the geometric framework can be applied to permutations, so paving the way to a complete geometric unification of EAs, but also that our geometric interpretation sheds light on this representation, revealing a hidden and intimate connection between geometry of permutations, crossover and sorting algorithms. Most of the pre-existing crossover operators for permutations are designed around interpretations. We have shown that they fit, some exactly and some approximately, the geometric crossover definitions connected with permutations interpretations. The permutation representation also allows for a number of edit distances connected with various notions of mutation. Each notion of edit distance induces a notion of geometric crossover. Because of the distance duality, under a given edit distance a point on a segment between two permutations is on a minimal sorting trajectory connecting the two permutations. This allows implementing such crossovers using *sorting algorithms*. Some edit distances give rise to crossovers that can be implemented exactly and efficiently. Other edit distances give rise to crossovers can be implemented efficiently (in polynomial time) only in an approximate way. An experimental comparison on the n-queens problem of a number of geometric crossovers based on three classical sorting algorithms have been presented. The experiments corroborate the "good mutation, good crossover" rule-of-thumb proposed in chapter 3. Additionally, a theoretical analysis of the fitness landscape for the TSP problem based on reversals distance for circular permutations shows that this fitness landscape is smooth, hence the associated geometric crossover should perform well. Experimental results agree with the theory: the proposed geometric crossover for TSP beats edge recombination, which is one of the best crossover for this problem.

In chapter 6, we have applied the geometric framework to three related representations – sets, multisets and partitions – in their variable size and fixed size variants. For the variable size case we have considered the ins/del edit distance, that for sets corresponds to the symmetric distance, and its extensions to the case of multi-sets and partitions, for which it becomes the move edit distance. We have shown that the geometric crossovers associated to the ins/del edit distance for sets is a crossover that requires offspring to be supersets of the intersection of the two parent sets and subsets of their union. The geometric crossovers associated to the ins/del distance for multisets and partitions are simple extensions of the inter/union crossover for sets. For the fixed size case we have considered the substitution edit distance, that is equivalent to the ins/del edit distance when restricted to set of fixed size. Therefore, geometric crossover under substitution edit distance is a restricted version of the inter/union crossover where all offspring are required to have fixed size. The geometric crossover associated to multisets and partitions for the fixed size case are analogous to the restricted inter/union crossover for sets. We have proved a *duality* between geometric crossover for sets, multisets and partitions on the one hand, and binary strings, integer vectors, and permutations with repetitions on the other. Interestingly, this allows the interchangeable use of representations and operators, being equivalent in terms of search, but exploiting the differences of corresponding representations in terms of expressing constraints.

In chapter 7, we have shown that the geometric framework naturally connects the notions of homologous crossover, subtree mutation, hyperschema and structural distance for syntactic trees. We have also described the structure of the space of syntactic trees associated with these elements and argued that, when using the standard interpretation of syntactic trees as programs, the associated landscape is smoother, hence the homologous crossover is a good choice.

In chapter 8, we considered the question: is biological recombination geometric? We have been able to answer this question in the affirmative by extending the geometric framework to sequences under edit distance. This has the remarkable consequence that the theory of geometric crossover applies to biological crossover as well, bridging the gap between biological evolution and artificial evolution. Our results reveal a deep connection between crossover for binary strings and biological recombination, showing that standard EA crossover is less of a caricature than it appears at first sight. We showed that, in the case of sequences endowed with edit distances, geometric crossover is a form of homologous crossover which performs the alignment on sequence contents before mixing genetic material. We proved various properties of this crossover and extended it to weighted alignments and alignment with gaps, which are more realistic models of biological recombination. Finally, we argued that biological recombination is geometric and discuss the consequences of this.

In chapter 9, we have extended the geometric framework introducing the notion of product crossover. This is a very general result that allows one to build new geometric crossovers customized to problems with mixed representations by combining pre-existing geometric crossovers in a straightforward way. We have presented this notion in the more general setting of metric transformations and discussed how to generalize product crossover to structural composition of geometric crossovers. Using the product geometric crossover theorem (theorem 9.2.2), we have also shown that traditional crossovers for symbolic vectors and blend crossovers for integer and real vectors are geometric crossover. We have then designed new geometric crossovers for the Sudoku puzzle that deal in a natural way with its constraints. We have demonstrated the usage of the important notion of product geometric crossover to straightforwardly derive (i) new geometric crossovers for the entire grid obtained by employing simple geometric crossovers for each row and (ii) the distance functions associated with them. This has allowed us to analyze the geometric fitness landscape associated to the new geometric crossovers and tell *a priori*, based on how the fitness landscape is constructed, that the new crossovers are very well-suited to the Sudoku puzzle. Hence, they are likely to perform well. We extensively tested a number of geometric crossovers, mutations and hill-climbers and found that the operators associated with the row-swap distance are the best and produce consistently (near) optimal Sudoku grids.

In chapter 10, we have extended the geometric framework introducing the notion of quotient geometric crossover. This could be clearly understood using the concept of geometricity-preserving transformation. Quotient geometric crossover is a very general and versatile tool. We have given a number of interesting examples as its applications. As shown in applications, quotient geometric crossover is not only theoretically significant but also has a practical effect of making search more effective by reducing the search space or removing the inherent bias.

In chapter 11, we have applied the geometric framework to the multiway graph partitioning problem. This problem presents two challenges for crossover: feasibility and redundancy. In the chapter we have addressed these issues and made the following contributions. We have designed a new crossover (Cycle H-GX) for permutations with repetitions that naturally suits partition problems and addresses the feasibility problem. We have shown that the important notion of normalization before recombination, that is very effective on problems with redundant encodings, can be naturally cast in geometric terms using a distance that filters the redundancy of the encoding together with the formal definition of geometric crossover. This geometric point of view on normalization has allowed us to study its effect on the fitness landscape and explain, within the geometric framework, why normalization before recombination for redundant encodings is a good idea. The landscape analysis also provided evidence for the fact that the labeling-independent distance is more suitable for the solution space of multiway graph partitioning problem than the Hamming distance. We have designed another geometric crossover (5pt LI-GX) based on a labeling-independent distance that addresses the redundancy problem. We have then combined these two crossovers obtaining a new, much superior geometric crossover (Cycle LI-GX) that suits partition problems with redundant encodings. In extensive experimentation, we demonstrated that this crossover outperforms previously known methods, either providing new lower bounds or equalling known best lower bounds in graph partitioning benchmark problems. The chapter also shows that the theory of geometric crossover is not just theory for its own sake, but it can be put at work in practice and produce excellent results for hard combinatorial optimization problems. Redundancy and feasibility are important issues arising in practice in many problems. Geometric crossover can address them in a natural way within its general framework.

In chapter 12, we have extended the geometric framework with the notion of multi-parent geometric crossover that is a natural generalization of two-parental geometric crossover: offspring are in the convex hull of the parents. Then, using the geometric framework, we have shown an intimate relation between a simplified form of PSO, without the inertia term, and evolutionary algorithms. This has enabled us to generalize in a natural, rigorous and automatic way PSO for any type of search space for which a geometric crossover is known. We specialized the general PSO to Euclidean, Manhattan and Hamming spaces, obtaining three instances of the general PSO for these spaces: EPSO, MPSO and HPSO, respectively. We have performed extensive experiments with these new PSOs. In particular, we applied EPSO, MPSO and HPSO to standard sets of benchmark functions and obtained two surprising results. Firstly, our PSOs have performed really well, beating the canonical PSO most of the time. Secondly, they have done so right out of the box. That is, unlike the early versions of PSO which required considerable effort before a good general set of parameters could be found, with our PSO we have done very limited preliminary testing and parameter tuning, and yet they have worked well. This suggests that they may be quite robust optimisers. An important feature of the geometric PSO algorithm is that it allows one to automatically define PSOs for all spaces for which a geometric crossover is known. Since geometric crossovers are defined for all of the most frequently used representations and many variations and combinations of those, our geometric framework makes it possible to derive PSOs for all such representations. Geometric PSO is rigorous generalization of the classical PSO to general metric spaces. In particular, it applies to combinatorial spaces. We have demonstrated how simple it is to specify the general geometric particle swarm optimization algorithm to the space of Sudoku grids (vectors of permutations). This is the first time that a PSO algorithm has been successfully applied to a non-trivial combinatorial space. This shows that geometric PSO is indeed a natural and promising generalization of classical PSO.

In chapter 13, we have extended the geometric framework with the generalization of one-point crossover to any metric space. This extension is interesting for a number of reasons. Firstly, the definition of one-point crossover seems so tightly dependent on the fact that the underlying representation is a vector, that one may believe that its generalization to generic metric spaces, hence to any representation, is simply not possible. This would constitute a natural limit to the geometric interpretation of crossover. The fact that one-point crossover indeed generalizes and has a surprisingly simple geometric characterization gives further depth to the geometric interpretation of crossover. Secondly, there are a number of recombination operators that extend the notion of one-point crossover to different solution representations. These extensions are based on analogy and intuition and are adaptations of the original one-point crossover to the special characteristics of the new representation. One-point geometric crossover makes rigorous the notion of one-point crossover across representations and formalizes the intuition behind it. This leads us to the third point. Since one-point geometric crossover is well-defined for any representation, it can be used to generate new one-point crossovers for new representations without involving any element of arbitrariness. Fourthly, since we now have a formal definition of one-point crossover that holds for all representations, we can use it to derive properties common to all one-point crossovers.

In chapter 14, we have shown that the abstract definition of geometric crossover induces two *non-empty* representation-independent classes of recombination operators: geometric crossovers and non-geometric crossovers. This is a fundamental result that puts a programme of unification of evolutionary algorithms and a general representation-independent theory of recombination operators on a firm ground. Because of the peculiarity of the definition of geometric crossover, proving non-geometricity of a recombination operator, hence the existence of the non-geometric crossover class, is a task that at first looks impossible. This is because one needs to show that the recombination considered is not geometric under *any distance*. However, taking advantage of the different possible ways of looking at the definition of geometric crossover we have been able to develop some theoretical tools to prove non-geometricity in a straightforward way. We have then used these tools to prove the non-geometricity of three well-known operators for real vectors, permutations, and syntactic trees representations. Also, we have used a similar technique to prove that the class of one-point geometric crossover is a proper subclass of geometric crossover.

In chapter 15, we have started developing a general theory of evolutionary algorithms endowed with geometric operators. We have defined the important notion of abstract behavior and used it to prove our main result on convex evolutionary search. This is a very general result that applies to virtually all types of evolutionary algorithms. We have then investigated the origin of the differences of evolutionary algorithms. A first source of differences is the representation-geometry duality that tells us that the evolutionary search can be equivalently described in geometric terms and representation terms. However, whereas the geometric description is the same for all spaces, the description based on representation is necessarily representation-specific. We have shown this for the cases of Hamming space and Euclidean space. Different evolutionary algorithms not only look different but, up to a certain extent, behave differently. What is the origin of their behavioral differences? Thank to the formal definition of formal metric evolutionary algorithm, we can now compare rigorously the same algorithm on different representations. We have shown that peculiar properties of the underlying search space are reflected into peculiar properties in the behavior of the search algorithm. As an example, we have shown that the property of isotropy of the search space is reflected into unbiased search by the search algorithm. This naturally has led us to lay the foundations for a behavioral taxonomy of evolutionary algorithms based on a taxonomy of their underlying search spaces based on their properties. We then have discussed the issue of how fitness landscape and the search done by the formal metric evolutionary algorithm relate in the light of NFL theorem. We claimed that the ultimate reason evolutionary algorithms work is fundamentally the same and it is to be found at an abstract level.

In summary the most important achievements of this thesis are:

Geometric framework: developed a formal geometric framework for the unification of EAs.

Generalization, simplification and clarification: shown that this framework helps to rethink, simplify and clarify various familiar aspects of evolutionary algorithms in a very general setting.

EAs Unification: shown that the unification is possible and many well-known operators for the most frequently used representations fit the framework.

Natural evolution: shown that natural evolution at molecular level fits this framework too.

Crossover principled design: shown that the formal framework can be used to do crossover principled design for any representation.

Synthesis of theories: shown that the geometric framework can bridge naturally very different theories such as those for continuous spaces and those for combinatorial spaces.

Towards a general theory: shown that the framework forms a solid basis for a representation-independent theory that applies to all evolutionary algorithms that fit the framework.

This work answers fully the research questions. Now there is no doubt about the possibility and utility of unification. The work explores the main research directions the unification opens up discovering new territories that now are ready to be conquered. For its own nature, a unification project can be evaluated only at the end when the whole picture is laid down, since each block reinforces the significance of unification as a unity only when put side by side with all the others. The author hopes that the overall picture that emerges looks like a harmonious balance between all aspects of the unification.

16.2 Limitations and future work

The framework presented in this thesis emphasizes the geometric aspect of the relationship between representations, search operators and fitness landscape. The major limitation of this framework in its present form is that it is incomplete in the following two senses:

- Whereas the specific probability distributions of the search operators are defined, we have not used them directly, neither in the design of operators, nor in the description of the dynamics of evolutionary algorithms with geometric operators. The underlying assumption here is that, *up to a certain extent*, the exact probability distributions are of secondary importance, and the underlying distance of a search operator is what matters. So, the framework presented in this thesis is, with respect to the probability distributions of the search operators, only a qualitative framework.

- In chapter 3, we introduced a notion of smooth fitness landscape and gave an initial explanation on the reason why geometric operators work well on this type of landscapes. Then, in the rest of the thesis, we used the notion of smooth landscape as a rule-of-thumb for the design of search operators, or as a basis for other claims. In order to provide solid foundations to the geometric framework, it is, therefore, necessary to define more rigorously the class of smooth fitness landscapes and prove results on the performance of evolutionary algorithms with genetic operators on this type of landscapes.

Not looking into the previous two topics, however, has been a wary choice. As highlighted in our research questions, we have preferred giving priority to showing that a unification of evolutionary algorithms is possible and useful before refining its details. The two issues risen above, however, constitute important future work. In the following, we present for each chapter, further topics of future work linked with the contents of the chapter.

In chapter 3, we have introduced the notions that we have then developed in the thesis. In future work we expect to further extend the applications of our framework to other representations and operators, to study the connections between this theory and other evolutionary computation theories (including those based on the notions of schema) and to investigate the links with generalized notions of convexity and continuity for the fitness landscape. In particular, we want to formalize the notion of smooth fitness landscape using random fields and derive the average performance of evolutionary algorithms with geometric operators.

In chapter 4, we have built new geometric crossovers associated with non-traditional metrics associated with continuous spaces with the aim of removing the search bias. In future work we want to implement the new crossovers presented and test them on problems for which we expect them to perform well from the study of their fitness landscapes. Another important reason to consider non-traditional metrics associated with real spaces is their potential use in specific applications. For example, we could consider vectors in polar coordinates instead of Cartesian coordinates and define metrics and geometric crossovers suited to them. These crossovers are likely to perform well on problems that are naturally expressed in terms of polar coordinates such as problems naturally defined on a sphere, for example, finding locations of antennas to maximize the signal coverage on the surface of the earth respecting some constraints. In general, building geometric operators associated with non-standard metrics for the real-coded representation well-suited to specific problems is a potentially powerful and straightforward way of embedding problem knowledge in the search. This possibility has been neglected in the specific case of continuous optimization for which the Euclidean distance has been almost always implicitly chosen.

In chapter 5, we have considered geometric crossovers for permutations. A permutation can be seen as a total order on its elements. In future work, we want to consider geometric crossover for partial orders and test it on scheduling problems such as job-shop and open-shop which solutions are naturally represented as partial orders.

In chapter 6, we have considered geometric crossover for sets and related representations. We showed that there is a interesting duality between sets and vectors. Although the geometric crossovers for these two representations are equivalent, in fact the two representations may be better suited to different types of problems. In future work we want to exploit this duality and solve problems that are better suited to one or the other representation.

In chapter 7, we have dealt with geometric crossover for GP syntactic trees. In future we will be looking at other distances for syntactic trees and other types of trees, and corresponding spaces and operators. In particular, we will focus on component-wise distances and grammatical distances, that arise by considering the syntactic tree as a collection of sub-components, and as a syntactic object based on a formal grammar, respectively.

In chapter 8, we have used the geometric framework to established a connection between biological evolution and evolutionary algorithm. We think this is a very promising result that deserves particular attention. In section 16.3 we outline a future research plan elaborating on this connection.

In chapter 9, we have introduced the notion of product geometric crossover that allows us to build new, more complex geometric crossovers, by combining together in a vector known geometric crossovers. We want to look at more complex combinations of geometric crossovers involving other container structures such as for example trees, graph and sequences, and find how they combine the distances of the composing geometric crossovers. Also, we want to explore the use of metric transformations for other purposes than actually constructing new geometric crossovers. Indeed, we are currently using some of these transformations to attack the following important open issue regarding geometric crossover. Given a geometric crossover, there is in general more than one distance for which the crossover is geometric (for example, cycle crossover is geometric under Hamming distance and swap distance). The question is: is there a distance that can be said to be the best distance to consider for a specific geometric crossover? If so, what is this distance? The answer relies heavily on the notion of metric transformation.

In chapter 10, we have introduced the notion of quotient geometric crossover. We have shown that this concept is very versatile and we have given many examples of its potential applications. In future work we want to do a thorough theoretical analysis and applications for each case. In particular, we have already started working on geometric crossovers for graphs and functions (finite state machines).

In chapter 11, we have applied the geometric framework to the graph partitioning problem and obtained solutions of very good quality. In future work, we want to apply geometric crossover design to more problems and devise a practical principled design methodology based the geometric framework. We discuss this more thoroughly in our research plan in section 16.3.

In chapter 12, we have shown that the geometric framework allowed us to generalize Particle Swarm Optimization to generic solution representations. In the future, we want to specialize this framework to solution representations and spaces very different from continuous spaces such as, for example, the space of genetic programs. The aim is to test the scope of the practical applicability our generalization of PSO. Geometric PSO does not include the inertia term. To generate the new position of the particle outside the convex hull between its current position, its local best and swarm best we used mutation. However, mutation captures only one aspect of the effect of inertia. Since inertia seems to be very important in the case of traditional PSO for continuous spaces, we will extend the geometric PSO with a proper generalization of inertia. From preliminary study, this seems to be possible using the geometric notion of extension ray that is well-defined for any metric space. Differential evolution is a very effective optimization method for continuous spaces and it is related to PSO. In future work, we will use the geometric framework to generalize to generic metric space differential evolution using similar techniques we have used to generalize PSO.

In chapter 13, we have introduced the notions of one-point crossover and multi-point crossover. In future work we want to refine the notion of multi-point crossover. Also, we want to understand when the class of one-point geometric crossover has an advantage over the class of geometric crossover.

In chapter 14, we have used the metric axioms to prove general properties of geometric crossover. In future work we want to derive more properties of the class of geometric crossover and its subclasses such as one-point crossover and complete crossover. This will allows us to obtain a classification of geometric crossovers according to their properties. In the chapter we briefly discussed the relationship between the geometric framework proposed in this thesis and Forma analysis showing that they do not coincide, but they are related. In future work we want to reveal more commonalities and differences between these two frameworks, and, if possible, integrate them in a single framework. In the chapter, given a recombination operator, we have asked if a distance satisfying the metric axioms existed so that the operator in question fits the definition of geometric crossover. Another very important and related question is: given a geometric crossover, what is the distance that fits it best and in what sense? This question makes sense because there are geometric crossover that are non-trivially geometric under more than a distance such as, for example, the cycle crossover for permutations that is geometric under Hamming distance and swap distance. Answering this question is very important in practice because it would tell us which fitness landscape associated with the geometric crossover to consider to predict the performance of an evolutionary algorithm using that operator. In future work we will address this important question.

In chapter 15, we have shown that evolutionary algorithms endowed with geometric crossover do convex search. This way of searching is linked with the schema theorem via the notion of convex sets. Convex sets that are invariant under geometric crossover[1] are a natural geometric generalization of schemata that are invariant under traditional crossover. In future work, we want to devise a general schema theorem for geometric crossover. We also want to specify schemata and schema theorem for a number of representations, different from binary strings, to develop an understanding of what it tells us about these representations. Convex evolutionary search may relate also with that part of mathematical optimization, called convex optimization. This is very interesting because convex optimization has a strong theory with performance guarantee. We would like to investigate the relationship between convex optimization and evolutionary search: this might lead to a strong theory for evolutionary algorithms and a generalization of convex optimization to more general search spaces. In evolutionary computation, existing theories for real-vector representation (e.g., evolutionary strategy theory) are very different from theories for binary strings (e.g., schema theorem). In future work we would like to recast pre-existing theories for different representations in terms of distance and generalize them to generic metric spaces. This would allow us to have a general and coherent theory of evolutionary algorithms.

[1]This means that when both parents belong to a convex set, also all their offspring, obtained by the application of geometric crossover, belong to that convex set.

16.3 Future research plan

The aim of the PhD thesis was to show that the unification of evolutionary algorithms in a general geometric formal framework is possible and useful. The natural second phase of research is to exploit the full potential of this framework in three different directions:

General computational complexity: it is well-known (NFL theorem) that over all problems on average every algorithm performs the same as pure random search. That is, it performs very poorly. Therefore, from a global optimization viewpoint, the ultimate goal of a theory of EAs is to find the largest class of problems for which an EA is guaranteed to approximate the optimum within a given expected error in polynomial time. The geometric unification of evolutionary algorithms has shown that, independently from the solution representation, all evolutionary algorithms perform the same type of search. This paves the way to finding a common defining property of a truly general class of problems on which all evolutionary algorithms with any representation perform provably well.

Formal principled design: evolutionary algorithms have been shown to be powerful tools to address a vast array of optimization, search, learning and design problems. However, designing a good evolutionary algorithm for a new problem is more of an art than a science. Preexisting theories of evolutionary algorithms do not give any guidance for the choice of solution representation and search operators design for new problems. The application of the geometric framework to a specific representation gives a formal recipe to build new search operators, mutations and crossovers, for any representation and it indicates what is likely to be a good solution representation for virtually any problem.

Algorithmic biological evolution: from an engineering viewpoint living creatures are marvelously complex devices perfectly fit to their environment. How could evolution be able to design such complex devices in a relatively short time contrasted with the magnitude of the space of all possible designs? Current biological theories of evolution such as population genetics and molecular evolution are not able to explain this. Casting a computational perspective on biological evolution could be the key to understand why biological evolution is able to do "intelligent design" so effectively and efficiently without intelligence. Evolutionary algorithms are only caricatures of biological evolution. Yet, they present a surprising design ability. This ability may have origin in the same algorithmic reason in both cases. The geometric framework naturally encompasses biological representations - variable-length DNA sequences – and biological operators, in particular homologous recombination which aligns sequences on contents before swapping genetic material. This makes this framework the perfect candidate to deliver a common explanation to the design ability to both artificial and biological evolution.

16.4 A final word

Unification is a delicate matter: besides proving that unification is possible in principle, one has to explain the consequences of the new angle on a network of related concepts. Plus, one has to show that many interesting cases are actually encompassed, but one also needs to go deep to show that the unification does capture some fundamental aspects common to all evolutionary algorithms and not only trivialities can be inferred for all evolutionary algorithms. Moreover one needs also to show that unification is not only theory for its own sake but that has practical advantages too. This needed to be done within a time-window of a PhD study. To different extents all the previous points have been addressed. Naturally, since a complete and thorough unification is a monumental task, the focus has been on particular topics that the author felt had priority and showed that a particular important territory is encompassed by the unification. By no means, each topic covered was exploited as much as it would have deserved. Indeed, each topic would have taken a PhD to be exhaustively addressed alone. Since the focus is the unification, for each topic it was shown instead how the geometric framework reveals its connection with all the others and this in turn has shed light on the specific topic itself.

Bibliography

[1] E. Aarts and J. K. Lenstra (eds.), *Local search in combinatorial optimization*, Wiley-Interscience, 1997.

[2] A. Aguilera and R. Rothstein (eds.), *Molecular genetics of recombination*, Springer, 2007.

[3] L. Altenberg, *The evolution of evolvability in genetic programming*, Advances in Genetic Programming (K. Kinnear, ed.), The MIT Press, 1994, pp. 47–74.

[4] ______, *The schema theorem and price's theorem*, Proceedings of the Workshop on the Foundations of Genetic Algorithms, 1995, pp. 23–49.

[5] P. J. Angeline, *Genetic programming's continued evolution*, Advances in Genetic Programming, vol. 2, The MIT Press, 1996, pp. 89–110.

[6] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation: Combinatorial optimization problems and their approximability properties*, Springer Verlag, 1999.

[7] D. Avis, *A survey of heuristics for the weighted matching problem*, Networks **13** (1983), 475–493.

[8] T. Bäck, D. B. Fogel, and T. Michalewicz (eds.), *Evolutionary computation 1: Basic algorithms and operators*, Institute of Physics Publishing, 2000.

[9] T. Back, D. B. Fogel, and Z. Michalewicz (eds.), *Handbook of evolutionary computation*, Oxford press, 1997.

[10] R. Battiti and A. Bertossi, *Greedy, prohibition, and reactive heuristics for graph partitioning*, IEEE Transactions on Computers **48** (1999), no. 4, 361–385.

[11] J. Blanton and R. Wainwright, *Multiple vehicle routing with time and capacity constraints using genetic algorithms*, Proceedings of the International Conference on Genetic Algorithms, 1993, pp. 452–459.

[12] K. D. Boese, A. B. Kahng, and S. Muddu, *A new adaptive multi-start technique for combinatorial global optimizations*, Operations Research Letters **15** (1994), 101–113.

[13] H. J. Bremermann, J. Rogson, and S. Salaff, *Global properties of evolution processes*, Natural Automata and Useful Simulations (H. H. Pattee, ed.), 1966, pp. 3–42.

[14] T. N. Bui and B. R. Moon, *Genetic algorithm and graph partitioning*, IEEE Transaction on Computers **45** (1996), no. 7, 841–855.

[15] A. Caprara, *Sorting by reversals is difficult*, Proceedings of the 1st Annual International Conference on Computational Molecular Biology, 1997, pp. 75–83.

[16] S. S. Choi and B. R. Moon, *Normalization in genetic algorithms*, Genetic and Evolutionary Compatation Conference, 2003, pp. 862–873.

[17] M. Clerc, *Discrete particle swarm optimization, illustrated by the traveling salesman problem*, New Optimization Techniques in Engineering, Springer, 2004, pp. 219–239.

[18] M. Clerc and J. Kennedy, *The particle swarm - explosion, stability, and convergence in a multidimensional complex space*, IEEE Transaction in Evolutionary Computation **6** (2002), 58–73.

[19] C. Colbourn, *The complexity of completing partial latin squares*, Discrete Applied Mathematics **8** (1984), 25–30.

[20] J. Cong and S. K. Lim, *Multiway partitioning with pairwise movement*, Proceedings of the International Conference on Computer-Aided Design, 1998, pp. 512–516.

[21] J. Cong, S. K. Lim, and C. Wu, *Performance driven multi-level and multiway partitioning with retiming*, Proceedings of the ACM/IEEE-CAS/EDAC Design Automation Conference, 2000, pp. 274–279.

[22] G. Cormode, *Sequence distance embeddings*, Ph.D. thesis, University of Warwick, 2003.

[23] N. A. C. Cressie, *Statistics for spatial data (revised edition)*, Wiley, 1993.

[24] J. C. Culberson, *Mutation-crossover isomorphism and the construction of discriminating functions*, Evolutionary Computation Journal **2** (1995), 279–311.

[25] L. Davis, *Applying adaptive algorithms to epistatic domains*, Proceedings of the International Joint Conference on Artificial Intelligence, 1985, pp. 162–164.

[26] ______, *Handbook of genetic algorithms*, Van Nostrand Reinhold, NY, 1991.

[27] K. DeJong, *Evolutionary computation: a unified approach*, The MIT Press, 2006.

[28] R. Descartes, *Discourse on method*, 1637.

[29] M. Deza and T. Huang, *Metrics on permutations, a survey*, Journal of Combinatorics, Information and System Sciences **23** (1998), 173–185.

[30] M. Deza and M. Lauren, *Geometry of cuts and metrics*, Springer, 1991.

[31] C. DiChio, A. Moraglio, and R. Poli, *Geometric particle swarm optimization on binary and real spaces: from theory to practice*, Proceedings of the Genetic and Evolutionary Computation Conference - Workshop on Particle Swarms: The Second Decade, 2007, pp. 2659–2666.

[32] R. Dorne and J. K. Hao, *A new genetic local search algorithm for graph coloring*, Proceedings of the Fifth Conference on Parallel Problem Solving from Nature, 1998, pp. 745–754.

[33] S. Dutt, *New faster kernighan-lin-type graph-partitioning algorithms*, Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 1993, pp. 370–377.

[34] A. E. Eiben, *Solving constraint satisfaction problems using genetic algorithms*, WCCI1 proceedings, 1994, pp. 542 – 547.

[35] A. Ekart and S. Z. Nemeth, *A metric for genetic programs and fitness sharing*, Genetic Programming, Proceedings of EuroGP'2000, 2000, pp. 259–270.

[36] W. J. Ewens, *Mathematical population genetics*, Springer, 2004.

[37] E. Falkenauer, *Genetic algorithms and grouping problems*, John Wiley & Sons, 1998.

[38] C. Fiduccia and R. Mattheyses, *A linear time heuristics for improving network partitions*, 19th ACM/IEEE Design Automation Conference, 1982, pp. 175–181.

[39] D. B. Fogel, *Evolutionary computation: Toward a new philosophy of machine intelligence*, IEEE press, 1995.

[40] D. B. Fogel, *Evolutionary computation: the fossil record*, IEEE Press, 1998.

[41] L. J. Fogel, A. J. Owen, and M. J. Walsh, *Artificial intelligence through simulated evolution*, Wiley, 1966.

[42] B. R. Fox and M.B. McMahon, *Genetic operators for sequencing problems*, Proceedings of the Workshop on the Foundations of Genetic Algorithms, 1991, pp. 284–300.

[43] E. Galvan and R. Poli, *An empirical investigation of how and why neutrality affects evolutionary search*, Proceedings of the Genetic and Evolutionary Computation Conference, 2006, pp. 1149–1156.

[44] M. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, 1979.

[45] M. Giritli, *From 3-sat to 2+p, 3-sat*, Ph.D. thesis, University of Amsterdam, the Netherlands, 2001.

[46] P. Gitchoff and G. P. Wagner, *Recombination induced hypergraphs: A new approach to mutation- recombination isomorphism*, Journal of Complexity **2** (1996), 37–43.

[47] F. Glover, M. Laguna, and R. Marti, *Fundamentals of scatter search and path relinking*, Control and Cybernetics **29** (2000), no. 3, 653–684.

[48] F. W. Glover (ed.), *Handbook of metaheuristics*, Kluwer, 2002.

[49] D. E. Goldberg, *Genetic algorithms and walsh functions : Part i, a gentle introduction*, Complex Systems **3** (1989), 129–152.

[50] ______, *Genetic algorithms and walsh functions : Part ii, deception and its analysis*, Complex Systems **3** (1989), 153–171.

[51] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, 1989.

[52] ______, *The design of innovation*, Kluwer Academic, 2002.

[53] D. E. Goldberg and R. Lingle, *Alleles, loci, and the traveling salesman problem*, Proceedings of the First International Conference on Genetic Algorithms and Their Applications, 1985, pp. 154–159.

[54] J. J. Grefenstette, *Deception considered harmful*, Proceedings of the Workshop on the Foundations of Genetic Algorithms, 1992, pp. 75–91.

[55] D. Gusfield, *Algorithms on strings, trees and sequences*, Cambridge University Press, 1997.

[56] S. Gustafson and L. Vanneschi, *Operator-based distance for genetic programming: Subtree crossover distance*, Proceedings of the 8th European Conference on Genetic Programming, 2005, pp. 178–189.

[57] R. B. Heckendorn and D. Whitley, *Predicting epistasis directly from mathematical models*, Journal of the ACM **45** (1999), no. 4, 563–750.

[58] C. Hohn and C. R. Reeves, *Graph partitioning using genetic algorithms*, Proceedings of the 2nd International Conference on Massively Parallel Computing Systems, 1996, pp. 31–38.

[59] J. H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, 1975.

[60] T. C. Hu and E. S. Kuh, *VLSI circuit layout theory and design*, IEEE Press, New York, 1985.

[61] I. Hwang, Y. H. Kim, and B. R. Moon, *Multi-attractor gene reordering for graph bisection*, Proceedings of the Genetic and Evolutionary Computation Conference, 2006, pp. 1209–1215.

[62] T. Jansen, *On classification of fitness functions*, Tech. Report CI–76/99, University of Dortmund, 1999.

[63] D. S. Johnson, C. Aragon, L. McGeoch, and C. Schevon, *Optimization by simulated annealing: An experimental evaluation, Part 1, graph partitioning*, Operations Research **37** (1989), 865–892.

[64] T. Jones, *Evolutionary algorithms, fitness landscapes and search*, Ph.D. thesis, University of New Mexico, 1995.

[65] L. Kallel, B. Naudts, and A. Rogers (eds.), *Theoretical aspects of evolutionary computing*, Springer, 2001.

[66] S. J. Kang and B. R. Moon, *A hybrid genetic algorithm for multiway graph partitioning*, Proceedings of the Genetic and Evolutionary Computation Conference, 2000, pp. 159–166.

[67] G. Karypis and V. Kumar, *Multilevel k-way partitioning scheme for irregular graphs*, Journal of Parallel and Distributed Computing **48** (1998), no. 1, 96–129.

[68] J. Kececioglu and D. Sankoff, *Exact and approximate algorithms for sorting by reversals, with applications to genome rearrangement*, Algorithmica **13** (1995), 180–210.

[69] J. Kennedy and R. C. Eberhart, *A discrete binary version of the particle swarm algorithm*, IEEE Transactions on Systems, Man, and Cybernetics **5** (1997), 4104–4108.

[70] ______, *Swarm intelligence*, Morgan Kaufmann, 2001.

[71] B. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell Systems Technical Journal **49** (1970), 291–307.

[72] H. Y. Kim, Y. Yoon, A. Moraglio, and B. R. Moon, *Geometric crossover for multiway graph partitioning*, Proceedings of the Genetic and Evolutionary Computation Conference, 2006, pp. 1217–1224.

[73] J. P. Kim and B. R. Moon, *A hybrid genetic search for multi-way graph partitioning based on direct partitioning*, Proceedings of the Genetic and Evolutionary Computation Conference, 2001, pp. 408–415.

[74] Y. H. Kim and B. R. Moon, *Lock-gain based graph partitioning*, Journal of Heuristics **10** (2004), no. 1, 37–57.

[75] M. Kimura., *Neutral theory of molecular evolution*, Cambridge University Press, 1983.

[76] D. E. Knuth, *Dancing links*, Tech. Report Preprint P159, Stanford University, 2000.

[77] John R. Koza, *Genetic programming: On the programming of computers by means of natural selection*, The MIT Press, 1992.

[78] H. W. Kuhn, *The Hungarian method for the assignment problem*, Naval Res. Logist. Quart. **2** (1955), 83–97.

[79] W. Langdon and R. Poli, *Foundations of genetic programming*, Springer-Verlag, 2002.

[80] G. Laszewski, *Intelligent structural operators for the k-way graph partitioning problem*, Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, pp. 45–52.

[81] P. Merz and B. Freisleben, *Fitness landscapes and memetic algorithms*, New ideas in optimization (D. Corne, M. Dorigo, and F. Glover, eds.), McGraw-Hill, 1999.

[82] P. Merz and B. Freisleben, *Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning*, Evolutionary Computation **8** (2000), no. 1, 61–91.

[83] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*, Springer, 1996.

[84] M. Mitchell, S. Forrest, and J. H. Holland, *The royal road for genetic algorithms: Fitness landscapes and ga performance*, Proceedings of the First European Conference on Artificial Life, 1992.

[85] E. H. Moore, *Definition of limit in general integral analysis*, Proceedings of the National Academy of Sciences of the United States of America, vol. 1, 1915, pp. 628–632.

[86] A. Moraglio, *Geometric unification of evolutionary algorithms*, Bulletin of the European Association of Theoretical Computer Science, 2005, p. 251.

[87] ———, *Geometric unification of evolutionary algorithms*, Proceedings of the European Conference on Genetic Programming - Graduate Student Workshop on Evolutionary Computation, 2006, pp. 45–58.

[88] A. Moraglio, C. Di Chio, and R. Poli, *Geometric particle swarm optimization*, European Conference on Genetic Programming, 2007, pp. 125–136.

[89] A. Moraglio, H. Y. Kim, Y. Yoon, and B. R. Moon, *Geometric crossovers for multiway graph partitioning*, Evolutionary Computation Journal (2007), (to appear).

[90] A. Moraglio, H. Y. Kim, Y. Yoon, B. R. Moon, and R. Poli, *Cycle crossover for permutations with repetitions: application to graph partitioning*, Proceedings of Parallel Problem Solving from Nature conference - Workshop on Evolutionary Algorithms: Bridging Theory and Practice, 2006.

[91] ______, *Cycle crossover for permutations with repetitions*, Tech. Report CSM-454, Computer Science department, University of Essex, UK, 2006.

[92] ______, *Generalized cycle crossover for graph partitioning*, Proceedings of the Genetic and Evolutionary Computation Conference, 2006, pp. 1421–1422.

[93] ______, *Geometric crossover for permutations with repetitions: application to graph partitioning*, Tech. Report CSM-448, Computer Science department, University of Essex, UK, 2006.

[94] A. Moraglio and R. Poli, *Topological crossover for the permutation representation*, Tech. Report CSM-408, Computer Science department, University of Essex, UK, 2004.

[95] ______, *Topological interpretation of crossover*, Proceedings of the Genetic and Evolutionary Computation Conference, 2004, pp. 1377–1388.

[96] ______, *Abstract geometric crossover for the permutation representation*, Tech. Report CSM-429, Computer Science department, University of Essex, UK, 2005.

[97] ______, *Geometric landscape of homologous crossover for syntactic trees*, Proceedings of IEEE congress on evolutionary computation, 2005, pp. 427–434.

[98] ______, *Geometric landscape of homologous crossover for syntactic trees*, Tech. Report CSM-430, Computer Science department, University of Essex, UK, 2005.

[99] ———, *Topological crossover for the permutation representation*, Proceedings of the Genetic and Evolutionary Computation Conference - Workshop on Adaptive Representations, 2005, pp. 332–338.

[100] ———, *Topological crossover for the permutation representation*, Proceedings of the Congress of the Italian Association for Artificial Intelligence - Italian worshop on Evolutionary Computation, 2005.

[101] ———, *Geometric crossover for sets, multisets and partitions*, Proceedings of Parallel Problem Solving from Nature conference, 2006, pp. 1038–1047.

[102] ———, *Inbreeding properties of geometric crossover and non-geometric recombinations*, Proceedings of the European Conference on Artificial Intelligence - Italian worshop on Evolutionary Computation, 2006.

[103] ———, *Product geometric crossover*, Proceedings of Parallel Problem Solving from Nature conference, 2006, pp. 1018–1027.

[104] ———, *Product geometric crossover*, Tech. Report CSM-447, Computer Science department, University of Essex, UK, 2006.

[105] ———, *Inbreeding properties of geometric crossover and non-geometric recombinations*, Proceedings of the workshop on the Foundations of Genetic Algorithms, 2007, (to appear).

[106] ———, *Topological crossover for the permutation representation*, Journal of the Italian Association for Artificial Intelligence (2007), (to appear).

[107] A. Moraglio, R. Poli, and R. Seehuus, *Geometric crossover for biological sequences*, Proceedings of the European Conference on Genetic Programming, 2006, pp. 121–132.

[108] ———, *Geometric crossover for biological sequences*, Proceedings of the Genetic and Evolutionary Computation Conference - Workshop on Adaptive Representations, 2006.

[109] A. Moraglio and J. Togelius, *Geometric pso for the sudoku puzzle*, Proceedings of the Genetic and Evolutionary Computation Conference, 2007, pp. 118–125.

[110] A. Moraglio, J. Togelius, and S. Lucas, *Product geometric crossover and the sudoku puzzle*, Proceedings of IEEE congress on evolutionary computation, 2006, pp. 470–476.

[111] R. Motwani and P. Raghavan, *Randomized algorithms*, Cambridge University Press, 1995.

[112] H. Mühlenbein, *Parallel genetic algorithms in combinatorial optimization*, Computer Science and Operations Research: New Developments in Their Interfaces, 1992, pp. 441–456.

[113] H. Mühlenbein, *The breeder genetic algorithm - a provable optimal search algorithm and its application*, IEE Colloquium on Applications of Genetic Algorithms **15** (1994), 5/1 – 5/3.

[114] H. Mühlenbein and D. Schlierkamp-Voosen, *Predictive models for the breeder genetic algorithm I: Continuous parameter optimization*, Evolutionary Computation **1** (1993), no. 1, 25–49.

[115] S-H. Nienhuys-Cheng, *Distance between herbrand interpretations: a measure for approximations to a target concept*, Proceedings of the 7th International Workshop on Inductive Logic Programming, 1997, pp. 213–226.

[116] T. Ohta, *The nearly neutral theory of molecular evolution*, Annual Review of Ecology and Systematics **23** (1992), 263–286.

[117] I. Oliver, D. Smith, and J. Holland, *A study of permutation crossover operators on the traveling salesman problem*, Proceedings of the Second International Conference on Genetic Algorithms, 1987, pp. 224–230.

[118] A. L. Olsen, *An evolutionary algorithm to solve the joint replenishment problem using direct grouping*, Computers and Industrial Engineering **48** (2005), no. 2, 223–235.

[119] Una-May O'Reilly, *Using a distance metric on genetic programs to understand genetic operators*, IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation, vol. 5, 1997, pp. 4092–4097.

[120] C. H. Papadimitriou, *Computational complexity*, Addison-Wesley, 1994.

[121] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*, Prentice-Hall, 1982.

[122] P. M. Pardalos and M. G. C. Resende (eds.), *Handbook of applied optimization*, Oxford University Press, 2002.

[123] M. D. Platel, M. Clergue, and P. Collard., *Maximum homologous crossover for linear genetic programming*, Proceedings of the European Conference on Genetic Programming, 2003, pp. 194–203.

[124] A. Prugel-Bennett, *Modelling evolving populations*, Journal of Theoretical Biology **185** (1997), 81–95.

[125] N. Radcliffe, *Equivalence class analysis of genetic algorithms*, Complex Systems **5** (1991), 183–205.

[126] ———, *Forma analysis and random respectful recombination*, Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, pp. 31–38.

[127] N. J. Radcliffe, *Genetic set recombination*, Proceedings of the Workshop on the Foundations of Genetic Algorithms, 1992, pp. 203 – 219.

[128] ———, *Nonlinear genetic representations*, Proceedings of Parallel Problem Solving from Nature Conference, 1992, pp. 259–268.

[129] ———, *The algebra of genetic algorithms*, Annals of Maths and Artificial Intelligence **10** (1994), 339–384.

[130] S. Rana, R. B. Heckendorn, and D. Whitley, *A tractable walsh analysis of sat and its implications for genetic algorithms*, AAAI-98 conference proceedings, 1998, pp. 392–397.

[131] C. E. Rasmusen and C. Williams, *Gaussian processes for machine learning*, the MIT Press, 2006.

[132] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution*, Frommann-Holzboog, 1973.

[133] J. Reed, R. Toombs, and N. A. Barricelli, *Simulation of biological evolution and machine learning*, Journal of theoretical biology **17** (1967), 319–342.

[134] C. R. Reeves and J. E. Rowe, *Genetic algorithms - principles and perspectives: A guide to ga theory*, Springer, 2002.

[135] C. M. Reidys and P. F. Stadler, *Combinatorial landscapes*, SIAM Review **44** (2002), 3–54.

[136] G. Ronald, M. Grtschel, and L. Lovsz, *Handbook of combinatorics*, vol. 2, The MIT Press, 1995.

[137] F. Rothlauf, *Representations for genetic and evolutionary algorithms*, Springer, 2002.

[138] J. E. Rowe, M. D. Vose, and A. H. Wright, *Group properties of crossover and mutation*, Evolutionary Computation Journal **10** (2002), no. 2, 151–184.

[139] G. Rudolph, *Convergence properties of evolutionary algorithms*, ICEC-99 conference proceedings, 1999, pp. 646–651.

[140] S. Russell and P. Norvig, *Artificial intelligence: A modern approach*, Prentice Hall, 2003.

[141] L. A. Sanchis, *Multiple-way network partitioning*, IEEE Transactions on Computers **38** (1989), no. 1, 62–81.

[142] H. P. Schwefel and G. Rudolph, *Contemporary evolution strategies*, Advances in Artificial Life, Springer, 1995, pp. 893–907.

[143] R. Seehuus and A. Moraglio, *Geometric crossover for protein motif discovery*, Proceedings of the Genetic and Evolutionary Computation Conference - Workshop on Adaptive Representations, 2006.

[144] O. J. Sharpe, *Towards a rational methodology for using evolutionary search algorithms*, Ph.D. thesis, University of Sussex, UK, 2000.

[145] A. Solomon, *Sorting by bounded permutations*, Ph.D. thesis, Virginia Polytechnic Institute, 1997.

[146] C. R. Stephens, R. Poli, A.H. Wright, and J.E. Rowe, *Exact results from a coarse-grained formulation of the dynamics of variable-length genetic algorithms*, Proceedings of the Genetic and Evolutionary Computation Conference, 2002, pp. 578–585.

[147] C. R. Stephens and J. Mora Vargas, *Effective fitness as an alternative paradigm for evolutionary computation i: General formalism*, Genetic Programming and Evolvable Machines **1** (2000), no. 4, 363–378.

[148] C. R. Stephens and H. Waelbroeck, *Schemata evolution and building blocks*, Evolutionary Computation Journal **7** (1999), no. 2, 109–124.

[149] C. R. Stephens and A. Zamora, *Ec theory: A unified viewpoint*, Proceedings of the Genetic and Evolutionary Computation Conference, 2003, pp. 1394–1405.

[150] R. Stephens and R. Poli., *Ec theory "in theory": Towards a unification of evolutionary computation theory*, Frontiers of Evolutionary Computation (A. Menon, ed.), Springer, 2004, pp. 129–155.

[151] P. D. Surry, *A prescriptive formalism for constracting domain-specific evolutionary algorithms*, Ph.D. thesis, University of Edinburgh, UK, 1998.

[152] W. A. Sutherland, *Introduction to metric and topological spaces*, Oxford University Press, 1975.

[153] G. Syswerda, *Uniform crossover in genetic algorithms*, Proceedings of the Third International Conference on Genetic Algorithms, 1989, pp. 2–9.

[154] M. van de Vel, *Theory of convex structures*, Elsevier, Amsterdam, 1993.

[155] C. van Hoyweghen, B. Naudts, and D. E. Goldberg, *Spin-flip symmetry and synchronization*, Evolutionary Computation **10** (2002), 317–344.

[156] L. Vanneschi, M. Tomassini, P. Collard, and M. Clergue, *Fitness distance correlation in structural mutation genetic programming*, Proceedings of the European Conference on Genetic Programming, 2003, pp. 455–464.

[157] J. P. C. Vergara, *Sorting by bounded permutations*, Ph.D. thesis, Virginia Polytechnic Institute, 1997.

[158] M. D. Vose, *The simple genetic algorithm: Foundation and theory*, MIT press, 1999.

[159] G. P. Wagner and L. Altenberg, *Complex adaptations and the evolution of evolvability*, Journal of Evolution **50** (1996), no. 3, 967–976.

[160] E. D. Weinberger, *Correlated and uncorrelated fitness landscapes and how to tell the difference*, Biological Cybernetics **63** (1990), 325–336.

[161] D. Whitley, *A genetic algorithm tutorial*, Journal of Statistics and Computing **4** (1994), 65–85.

[162] ———, *Functions as permutations: Implications for no free lunch, walsh analysis and statistics*, Proceedings of Parallel Problem Solving from Nature Conference, 2000, pp. 169–178.

[163] D. Whitley, T. Starkweather, and D. Shaner, *Travelling salesman and sequence scheduling: quality solutions using genetic edge recombination*, Handbook of Genetic Algorithms (L. Davis, ed.), Van Nostrand Reinhold, 1991, pp. 350–372.

[164] Darrell Whitley, *An overview of evolutionary algorithms: practical issues and common pitfalls*, Information and Software Technology **43** (2001), no. 14, 817–831.

[165] D. H. Wolpert and W. G. Macready, *No free lunch theorems for optimization*, IEEE Transaction on Evolutionary Computation **1** (1996), no. 1, 67–82.

[166] ———, *No free lunch theorems for search*, Tech. Report SFI-TR-95-02-010, The Santafe Institute, 1996.

[167] X. Yao and Y. Liu, *Fast evolutionary programming*, Proceedings of the Conference on Evolutionary Programming, 1996, pp. 451–460.

[168] T. Yato, *Complexity and completeness of finding another solution and its application to puzzles*, Master's thesis, University of Tokyo, Japan, 2003.

[169] Y. Yoon, H. Y. Kim, A. Moraglio, and B. R. Moon, *Geometric crossover for real-vector representation*, Tech. Report CSM-466, Computer Science department, University of Essex, UK, 2007.

[170] ———, *Quotient geometric crossover*, Tech. Report CSM-467, Computer Science department, University of Essex, UK, 2007.