On The Design of Genetic Algorithms for Geographical Applications

S. van Dijk
PO Box 80.089
3508 TB Utrecht
The Netherlands
steven@cs.uu.nl

D. Thierens
PO Box 80.089
3508 TB Utrecht
The Netherlands
dirk@cs.uu.nl

M. de Berg
PO Box 80.089
3508 TB Utrecht
The Netherlands
markdb@cs

Abstract

In many geographical optimization problems, the linkage (which determines the structure of building blocks) is determined by the spatial relationships between the components of a solution. Therefore the linkage can be identified easily, unlike in most other problems. Based on this observation, we develop a hybrid GA that uses a geometrically local optimiser—one that computes good solutions to subproblems that are local in the geometric sense. One of the main advantages of our method is that it leads to GA's that are easily adapted to slight changes in the problem definition, without the need to tune many parameters in the fitness function. We apply our method to the map labeling problem, (placing as many names as possible on a map, without overlap), where it leads to good results.

1 Introduction

A geographical information system (GIS) stores geographical data like the shape of countries, the height of mountains, the course of rivers, rainfall in different regions, and so on. This information can be extracted, displayed, and analyzed in various ways. Some of the problems that a GIS has to solve are optimization problems. A well known example that arises in the display of maps is the *map labeling problem*. Here one wants to label cities and rivers with their name, areas in a precipitation map with the amount of rainfall they get, etc. The problem is to place the labels strategically, so that none of them overlap. If this is not possible, one would like to maximize the number of labels that can be placed without introducing overlap. Even in very restrictive settings (see below), these problems are already NP-complete. There are also all kinds of other constraints, however, making them even harder. Some of these are hard (like: the state capitals should always receive a label), others are soft (like: if possible, it is preferred that the label of a city is placed to its right and above it).

Genetic algorithms are a good candidate to tackle such difficult problems. Unfortunately, there are some problems when GA's are developed in a standard manner. One of the main problems is that the fitness function will contain many parameters, which are used to weigh the various constraints. These parameters need to be carefully tuned, which is a time-consuming process. Moreover, any slight change of the problem definition leads to a new tuning phase.

We propose a framework for designing GA's for geographical applications that alleviates this problem, and leads to GA's with a good performance. It is based on the use of local optimizers. In itself this is nothing new: it is well known that hybrid GA's are often the best option in practical applications—see for instance the book by Davis [3], where many examples of hybrid GA's are given. Our local optimizers are different from the standard ones, however. Normally a local optimizer tries to improve the solution by only looking at a neighborhood in the fitness landscape. Our local optimizer does this as well, but it is unusual in the sense that it is also a *geometrically local optimiser*. This is quite useful, for the following reason. In most problems, the *linkage*—the non-linear interactions between various components of the solution—is unknown. In geographical problems, however, the linkage is usually determined by the spatial relationships between the components: in the map labeling problem, for instance, labels of cities that are close may interact whereas labels of cities that are far apart do not interact. Thus building blocks are also spatially local components (small regions of the map). Hence, a geometrically local optimiser can produce building

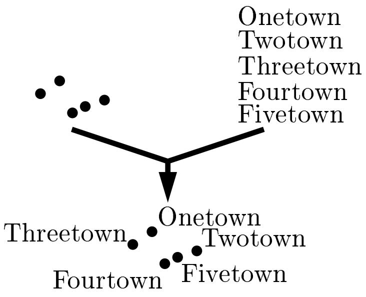

Figure 1: The map labeling problem.

blocks, allowing the GA to mix directly at the level of building blocks. Another advantage of the use of geometrically local optimisers is that they can be used to deal with geometrically local constraints (like: every state capital should receive a label). Such constraints are often difficult to incorporate into the fitness function. In fact, we only need to put the combinatorial constraints in the fitness function; local and soft constraints can be handled by the geometrically local optimiser. This way the number of tunable parameters in the fitness function is reduced significantly.

This paper is organized as follows. In Section 2 we give some background on the problem we use to illustrate our ideas, namely the map labeling problem. In Section 3 we outline our framework for designing GA's. This is followed by Section 4, which is devoted to the geometrically local optimisers. There we describe the advantages in three different Subsections: Section 4.1 discusses the benefits for searching in the search space, Section 4.2 describes the increased robustness and extendibility of the algorithm, and Section 4.3 shows how our technique reduces the number of parameters in the fitness function. We show how our techniques take care of the need for robustness that geographical problems have in Section 5. We present experimental results for the map labeling problem in Section 6. Section 7 discusses our techniques further, and we close the paper in Section 8 with a short summary.

2 The map labeling problem

This paper uses the geographical problem of map labeling as a running example to illustrate our ideas. This is a hard problem which still has not been solved satisfactorily. The problem consists of placing labels near certain features on a given map (see Figure 1). The number of labels placed should be maximal and the map should be readable, aesthetically pleasing, and give easy access to the information it is offering. We shall concentrate on a relatively simple version of the problem, where the only features to be labeled are points (cities on a large-scale map, for instance), and the label of each point can be placed in four different locations: with its top right corner at the point, with its top left corner at the point, with its bottom right corner at the point, and with its bottom left corner at the point. This is called the *map labeling problem for point features in the four-position model.* The problem is to choose the position of each point in such a way that the number of labels without overlap is maximized. Even this seemingly simple problem is NP-complete [9]. In practice, there are many other constraints: certain positions of a label are preferable to others (usually the position to the right and above the point is considered best), certain points (cities) are more important to label than others, and so on.

The map labeling problem for point features in the four-position model has been studied by various authors. Techniques that have been used include enumerative methods (various rule based methods have been proposed, for example by Cook *et al.* [2] and recently a method based on a branch and bound algorithm was devised by Verweij *et al.* [14]), greedy algorithms (see for example Langran *et al.* [8] and Yoeli [15]), local search (a gradient descent approach was described in the article by Christensen *et al.* [1]), heuristic search (see for example Hirsch [7] and Feigenbaum [5]), 0/1 linear programming (by Zoraster [16, 17]), simulated annealing (proposed by Christensen *et al.* [1]), and genetic algorithms (different approaches were proposed by Djouadi [4] and Verner *et al.* [13][1]). So far the best results were obtained by the simulated annealing algorithm of Christensen *et al.* [1] and the GA of Verner *et al.* [13]—see the paper of Christensen *et al.* or the technical-report version of the present paper [12] for extensive comparisons. Our algorithm is competitive with these two algorithms with respect to performance (see Section 6), but we believe it is much easier to adapt to slight changes in the problem definition. Hence, we believe it will be the method of choice in practical situations, where various soft and local constraints have to be taken into account.

3 Outline of the Genetic Algorithm

The generic genetic algorithm for solving geographical problems uses for the encoding a string of genes where the alleles come from a finite alphabet. The selection scheme uses the Elitist Recombination Scheme as de-

[1]Very recently, a genetic algorithm for map labeling was proposed by Raidl [10]. It uses an interesting, but more traditional approach then the one which is described in this paper. We did not have the time to do comparisons with this approach.

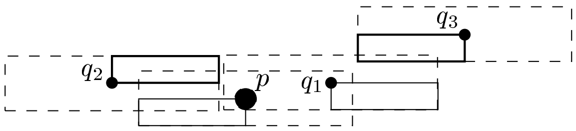

Figure 2: A local region can be defined as a point and its *rivals*. A rival is a point whose label can intersect with the chosen point. Thus, the local region of point p contains the points p, q_1 and q_2. The local region of point q_1 contains the points q_1, p and q_3.

scribed by Thierens *et al.* [11]. In this scheme two parents are randomly chosen, two children are generated and from this family of four the two best individuals (as measured by the fitness function) replace the two parents. Advantages of this scheme are preservation of good solutions (elitism on the family level), simplicity, constant selection pressure and robustness. No mutation is used. Crossover and geometrically local optimisers are the only operators left, which both work on the level of building blocks.

What a building block is depends on the problem, but here it is easy to determine since the problem is a geometrical one. Each building block spans a (geometrically) local region and local regions can overlap.

Running example: For the map labeling problem these local regions can consist of a city with its neighbors. Since each neighbor defines its own local region, it follows that these regions overlap. See Figure 2 for an graphical example.

Crossover works by randomly choosing local regions until the total of the amount chosen is roughly half the size of the total encoding. Then the union of these local regions if copied from the first parent to the first child and the complementary part is taken from the other parent. The second child is generated likewise. It is expected that this kind of crossover will generate new conflicts since local regions overlap. So after crossover, places where new conflicts can have arisen are checked for conflicts and if so, the geometrically local optimiser is applied in an attempt to fix the conflict.

The fitness function should be as simple as possible to avoid having many tunable weighing factors and as many aspects of the problem as possible should be handled in the geometrically local optimiser (this is called the *principle of maximal delegation*). The fitness function should only contain those aspects of the problem which are combinatorially difficult. We will discuss this further in Section 4.3.

Running example: The map labeling problem is hard because it has to maximize the number of free labels, which is an NP-complete problem (see for a proof Marks *et al.* [9]). So the fitness function should count the number of free labels, and nothing more. All other aspects of the problems (aesthetical constraints, etc.) do not make it so hard, so they should only belong in the geometrically local optimiser.

4 Geometrically local optimisers

A geometrically local optimiser is a procedure which takes a geometrically bounded part of the solution and tries to improve that part without regard for the rest of the solution. As such, geometrically local optimisers can be used for any problem for which the scope of a subpart can be strictly defined. In geographical problems the scope can be determined in a geometrical way (another class of problems which are suitable are graph-problems in general, such as graph coloring). As such, a subpart can be considered as defining a partition and the task of the local optimizer is to find the building block within that partition.

Running example: Consider a geometrically local optimiser for the problem of labeling a map, which is applied to a certain point. If the label intersects another label, the local solution can be optimized by moving the label to a position where it is free. The scope of this geometrically local optimiser consists of the point it is applied to and its rivals (see Figure 2).

Geometrically local optimisers are often easy to construct, since in geographical problems it is not hard for a local part of the map to see how to improve it. Since the genetic algorithm works with a randomized population, for a specific part of the problem the geometrically local optimiser will be applied several times in different contexts. The local solution which eventually dominates the population is the local solution which fits also into the global solution. Another way of looking at it is by viewing the geometrically local optimisers as creating building blocks, which compete and eventually the global solution is formed out of building blocks which work well together.

A geometrically local optimiser therefore should have the following properties:

- It should be able to improve or at least not degrade a part of the solution with limited scope.
- When possible, it should choose randomly from multiple local optima to provide diversity for the global selection mechanism of the genetic algorithm.

- It should be fast, since it will be applied often.

Geometrically local optimisers have the following uses:

- They are used to resolve conflicts which are introduced during crossover as a result of building blocks overlapping.
- They construct locally good solutions from which the GA can construct a globally good solution.
- They allow for efficient searching of the search space by adaptively reducing the cardinality of the effectively processed alphabet. (See Section 4.1 for more details.)
- They make it possible to easily extend the problem with new constraints without significant performance loss. (See Section 4.2 for more details.)
- They allow for the handling of precedences in geographical problems. When a certain aspect can only be considered when another aspect is satisfied (for example, make sure a capital is labeled, and then consider if a label can be placed in a preferred position), this difference in importance is hard to respect in a standard GA. (See Section 4.2 for more details.)
- They allow for the use of the maximal delegation principle which removes aspects of the problem from the fitness function which are not combinatorially difficult. (See Section 4.3 for more details.)

The next Subsections explain some of the issues mentioned above in more detail.

4.1 Efficient searching

The encoding of the solution is a string of characters taken from some finite alphabet. The size of the search space is A^n, with the cardinality of the alphabet denoted by A and the length of the string denoted by n. It follows that reducing the cardinality of the alphabet reduces the size of the search space and will make the GA run faster. This is confirmed by the work of Harik *et al.* [6] who showed that for a GA working on an alphabet of cardinality A and a building block size of k, the proper population size is proportional to A^k. A smaller population size means a faster GA. Reducing the cardinality of the alphabet is desirable, but it should still be possible to express the optimal solution. It is usually very difficult, if not impossible to find a new encoding which meets these conditions (smaller cardinality and equal power of expression).

Geometrically local optimisers adaptively reduce the cardinality of the *effectively processed* alphabet, while maintaining the same power of expression. Since the operators of the GA (crossover and the geometrically local optimiser) work on the level of building blocks, the actual alphabet which is processed (we will call this the *meta-alphabet*) is different from the alphabet which is used to encode the solution with. Its characters consist of the configurations which can arise in the partition a building block is part of.

Running example: The cardinality of the alphabet which is used to encode the solution of the map labeling problem is four, the number of positions a label can have (if labels can be deleted, an extra position becomes available). Geometrically local optimisers work on a city and a small neighborhood. This local region becomes optimized and therefore the label of the city loses some positions in which it can be placed. Each configuration of the local region the geometrically local optimiser works on corresponds with a character in the meta-alphabet. The cardinality of the meta-alphabet is the number of configurations that local region can be in. Since the number of possible configurations is reduced by the appliance of the geometrically local optimiser, the cardinality of the meta-alphabet is reduced.

The only two operators that change solutions are crossover and geometrically local optimisers. After crossover, the geometrically local optimiser is applied to the parts of the solution which can have changed (and therefore may have re-introduced configurations which had disappeared). That way it can be guaranteed that certain configurations which are not allowed will not re-appear, since the geometrically local optimiser will always change the configuration to one which is allowed.

The second condition (equal power of expression) can be met by ensuring that the geometrically local optimiser only makes those configurations unreachable that are clearly non-optimal. This is easy to do for geographical problems, since in a local sense it is usually obvious if a configuration is non-optimal.

4.2 Extendibility

When solving geographical problems, one often needs to take many different types of constraints into account. Some of these are hard (combinatorial) constraints, and others are local or soft (e.g. aesthetical) constraints. Moreover, when these problems have to be solved by a geographical information system, the constraints may be specified by the user. Depending on how much detail of the map the user wants to consider, more constraints are added and are therefore not known beforehand. This means that algorithms used in a GIS need to be robust in the sense that they should work well for a class of variations of some problem, not for one specific problem definition.

Geometrically local optimisers offer a mechanism to make the algorithm extendible, which increases robustness (other techniques to increase robustness are discussed in Section 5). Separating the problem into a part which is combinatorially hard and into a part consisting of soft constraints, we can put the combinatorially hard constraints in the fitness function and all other constraints in the local optimizer. Now suppose we want to change the problem slightly by adding an extra constraint. Depending on the nature of the new constraint, it should be placed in either the fitness function or the local optimizer. Since most new constraints will fall in the class which ends up in the local optimizer, extendibility is preserved since constructing a optimizer for a local situation is usually relatively easy.

> **Running example:** The problem is to label a map and maximize the number of free labels. This is the constraint which makes the problem hard. We can complicate the problem by demanding that the label is placed in a preferred position. Since this is a soft constraint, the only thing we have to do is provide a way to handle this constraint in the local optimizer. Nothing else changes in the algorithm. A way to handle this is to determine a set of positions where the label can be placed without intersecting other labels. From this set of positions we choose a position in order of decreasing preference and place the label in that position.
>
> Now suppose we wish to change the problem definition by stipulating that certain points (state capitals, for instance) must receive a label that does not overlap other labels. Such a local but hard constraint is quite difficult to incorporate in the fitness function, since all solutions which have their capitals labeled should have high enough fitness not to be selected against. This lead to large differences in fitness between solutions with and without labeled capitals. As a result, the GA may suffer from premature convergence or hitchhiking. By applying a geometrically local optimiser to such a point we can enforce the solution to satisfy this constraint, and there is no need to try and put it in the fitness function.

The latter example shows that geometrically local optimisers also make it easy to have constraints which are more important than other constraints (labeling a capital is more important than any number of labels placed in a preferred position). Such a relationship would be hard to express in the fitness function (see also the next Subsection).

4.3 Avoiding tuning of the GA

As noted before, geographical problems often consist of two types of aspects: those which make it combinatorially hard and those that include other, soft constraints. In general, one does not want to degrade the solution for the combinatorial part in order to make the solution more pleasing to the eye. So how should these two types of aspects be combined?

> **Running example:** In the case of the problem for map labeling, the aspect which makes it hard is finding a solution which maximizes the number of free labels. The aspects which are aesthetical or local constraints are the additional constraints such as placing the label in a preferred position or guaranteeing that a city (for example a capital) is labeled.

Combining different aspects of a problem is often done by making a different fitness function for every different aspect and summing these to obtain the global fitness. This approach has the disadvantage that each partial fitness function needs a weighing factor to determine what its contribution to the global fitness should be. This leads to all kinds of problems when used in genetic algorithms for geographical applications. Firstly, there is no guarantee that the combinatorial part is not deteriorated by the aesthetical constraints and secondly, these weighing factors need to be tuned to find the optimal setting of the weighing factors. Tuning requires costly GA runs to find the right values, and the GA probably only works well on that specific problem instance. Furthermore, since these runs are often done on small scale problems, it is questionable if they extend to large scale problems.

Most GA's need to be tuned. Several tuning parameters can be eliminated using other techniques like using Elitist Recombination to eliminate the need of setting P_c (since disrupted building blocks lower the fitness of the child and therefore it looses the family competition). The weighing factors of the fitness function need to be tuned also when you consider multiple problem aspects. Using geometrically local optimisers alleviates this problem since soft constraints can be considered there. This approach can be summarized as the *principle of maximal delegation*: put as much as possible into the local optimizer, leaving only the combinatorial aspects in the global fitness function.

5 Robustness

As described in the Introduction and Section 4.2, for geographical problems the notion of robustness is important: the algorithm should be able to keep performing well when the problem instance is extended or changed in some other way. To be robust an algorithm should be extendible, minimize the number of parameters which have to be tuned (and thus make the algorithm specific to some instance), avoid behavior which makes the genetic algorithm unreliable (such as hitchhiking and genetic drift) and provide a way to efficiently search any search space.

The algorithm which we propose has the following points which account for its robustness:

- P_c, the crossover probability, can safely be set to 1 because of the elitist recombination scheme.
- P_m, the mutation probability, can be set to 0.0 because geometrically local optimisers are used.
- No tuning of fitness weights is necessary because only the combinatorially hard aspect of the problem is measured in the fitness function.
- Hitchhiking is prevented because crossover mixes on the level of building blocks.
- Genetic drift is avoided because of the constant selection pressure which acts on all parts of the solution.
- The cardinality of the alphabet which is effectively processed is reduced using the local optimizers so the search space remains tractable.
- Extending the problem with new constraints is easy since the algorithm is designed in such a way to allow this.

6 Comparisons

We implemented a GA using the techniques in this paper to solve the map labeling problem. We also implemented the two algorithms which reported the best results in literature: the genetic algorithm of Verner *et al.* [13][2] and the simulated annealing algorithm of Christensen *et al.* [1][3]. The maps which had to be labeled were randomly generated according to a procedure from the article of Christensen *et al.*: a number of cities was randomly placed on a grid of fixed dimensions (792 by 612) with labels also of fixed dimensions (30 by 7). The GA of Verner *et al.* used an eight-position model and the SA of Christensen *et al.* used a four-position model. We compared our algorithm using the appropriate position model. The results are shown in Figure 3. As shown, the results of our algorithm are superior to the results of the algorithm by Verner *et al.* The SA algorithm and the GA had comparable results. On large maps, the SA algorithm was faster because the GA had to use a higher population size. We also experimented with adding extra constraints and this lead to the GA producing better results then the SA algorithm. However, we did not change the cooling schedule, but it is plausible that the scheduling could be tuned to obtain better results.

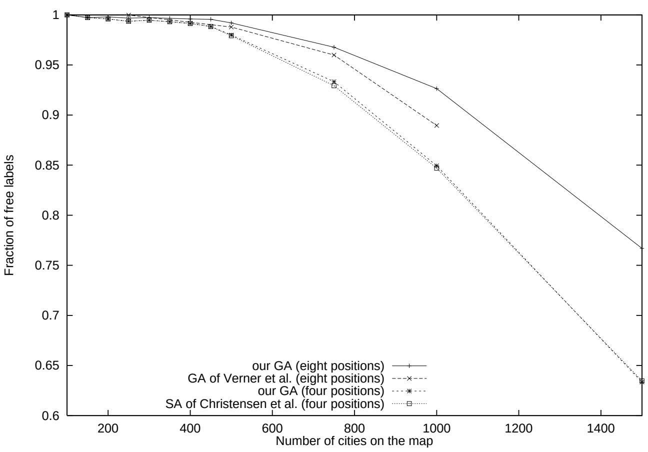

Figure 3: Our GA compared with the best known algorithms.

7 Discussion

The approach which is taken in this paper is to take explicitly into consideration the specific characteristics of geographical problems, which are:

1. Geographical problems are often encountered in GIS-use. The designer should therefore take into account the fact that the constraints fall into (combinatorially) hard and local/soft (e.g. aesthetical) constraints.
2. These kind of problems have geometric properties which provide clear linkage.

[2]Unfortunately, we could not reproduce the results with our implementation. We compare against the reported results instead.

[3]The GA of Raidl [10] was unknown to us at that point and was not considered. Comparisons are not possible without an implementation since the paper considers the problem of minimizing the number of conflicts (pairs of overlapping labels) instead of maximizing the number of free labels.

Both points were addressed in this paper. The fact that the problems contain different types of constraints is dealt with using the principle of maximal delegation and the use of local optimizers. The second point is acknowledged by the fact that the algorithm is mixing on the level of building blocks (since crossover chooses local regions and the local optimizer tries to generate building blocks).

This paper described a technique using geometrically local optimisers for solving geographical problems. For a general geographical problem, we recommend following the following technique:

1. Analyze the problem to determine its geometrical structure and decide upon the scope of the local region which will be mixed.

   Running example: The structure of the problem of maximizing the number of free labels for point feature map labeling can be expressed using a graph which has the point features as its nodes, and has an edge between two nodes if the corresponding cities are rivals (see Figure 2). The scope of the local region is a chosen city with its rivals.

2. Use Elitist Recombination. Apart from the advantage of family elitism (good solutions can never be replaced by worse ones), Elitist Recombination allows for the safe setting of P_c at 1.0. The parameter P_c has to balance the mixing of building blocks with the rate of disruption by crossover. This is not necessary with Elitist Recombination since disrupted strings will not win the family competition. Eliminating the need to set P_c makes the GA more robust.

3. Perform crossover by randomly picking local regions until the total combined area exceeds half the size of the map. Then the chosen region can be copied to the one child while the complementary part is copied to the other child. This way the GA mixes on the level of the building blocks.

4. After crossover conflicts may have arisen since the local regions overlap. Call the geometrically local optimiser on the cities which can have a new conflict.

5. Analyze the problem to determine how it can be decomposed in different aspects. Of each aspect it should be clear whether it makes the problem combinatorially hard. If it does, put it in the fitness function. Do not put anything else in the fitness function, handle it in the geometrically local optimiser instead. To avoid that the geometrically local optimiser changes local regions in a way which is opposed to the selection pressure induced by the fitness function, all aspects should be handled in the geometrically local optimiser.

   Running example: Suppose the problem is to maximize the labels on a map (first aspect), while placing labels preferably in the top right position (second aspect) and always placing the labels of capitals (third aspect). The first aspect is an NP-complete problem, so we count the number of free labels in the fitness function. We do not consider the other two aspects in the fitness function. The geometrically local optimiser does the following. First, it determines where the label can be placed. If the city is a capital, then it can be placed anywhere (and the labelings of the surrounding cities will adapt to that). If it is a regular city, the label can be placed where it does not overlap any other label. Now we can choose the position which is most preferred. Note that the geometrically local optimiser locally maximizes the number of free labels, while still considering the other aspects. This is done however in a way which guarantees that the number of free labels will be optimal. In contrast, the traditional approach is to put all these aspects in the fitness function.

This approach works well, as can be verified by examining the case study of map labeling (see for extensive detail Van Dijk *et al.* [12]), which includes several extensions to the basic problem of maximizing free labels, without any significant performance decrease. In terms of speed it is outperformed by the simulated annealing algorithm of Christensen et al. [1] for the basic problem on large maps. However, when the problem includes more extensions the GA starts performing better.

A map labeled using a genetic algorithm which was designed with the techniques from this paper is shown in Figure 4.

8 Conclusion

This paper described a general technique for geographical optimization problems. The technique uses geometrically local optimisers to reduce the search space, avoid tuning, include soft constraints safely and make the GA robust and extendible. Using the case study of the problem of putting names on maps it was shown

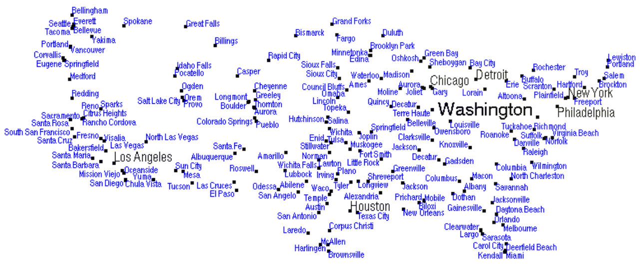

Figure 4: A map of the United States labeled with the GA. The capital and large cities are guaranteed to be placed. Different font sizes are used. Only a subset of labels was selected (by the GA) for placement since there were to many to place without overlap. Labels have preferred positions, with the top right being the most preferred position.

that the technique is useful for these kind of problems.

Future work includes the application of this technique for other problems (for example map labeling involving line and area features) and improvements in applying the geometrically local optimiser to avoid optimizing an area which already is good enough.

References

[1] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, 14(3):203–232, 1995.

[2] A. C. Cook and C. B. Jones. A Prolog interface to a cartographic database for name placement. In *Proceedings Fourth International Symposium on Spatial Data Handling*, pages 701–710, 1990.

[3] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[4] Y. Djouadi. Cartage: A cartographic layout system based on genetic algorithms. In *Proc. EGIS*, pages 48–56, 1994.

[5] M. Feigenbaum. Method and apparatus for automatically generating symbol images against a background image without collision utilizing distance-dependent attractive and repulsive forces in a computer simulation, 1994. Assigned to Hammond Inc., Maplewood, New Jersey. U.S. Patent filed 11/5/93, received 10/11/94.

[6] G. Harik, E. Cantú-Paz, D. Goldberg, and B. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *Proc. IEEE Int. Conf. On Evolutionary Computation*, pages 7–12, 1997.

[7] S. A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5–17, 1982.

[8] G. E. Langran and T. K. Poiker. Integration of name selection and name placement. In *Proc. Auto-Carto 8*, pages 50–64, 1986.

[9] J. Marks and S. Shieber. The computational complexity of cartographic label placement. Technical Report TR-05-91, Harvard CS, 1991.

[10] G. Raidl. A genetic algorithm for labeling point features. In *Proc. of the Int. Conference on Imaging Science, Systems, and Technology*, pages 189–196, Las Vegas, NV, July 1998.

[11] D. Thierens and D. Goldberg. Elitist recombination: An integrated selection recombination GA. In *Proc. IEEE Int. Conf. on Evolutionary Computation*, pages 508–512. IEEE Service Center, Piscataway, NJ, 1994.

[12] S. van Dijk, D. Thierens, and M. de Berg. Robust genetic algorithms for high quality map labeling. Technical Report TR-1998-41, Utrecht University, 1998.

[13] O. Verner, R. L. Wainwright, and D. A. Schoenefeld. Placing text labels on maps and diagrams using genetic algorithms with masking. *INFORMS Journal of Computing*, 9(3), 1996.

[14] A. Verweij and K. Aardal. An optimisation algorithm for maximum independent set with applications in map labelling. In *Proc. 7th Annu. European Sympos. on Algorithms*, 1999. (to appear).

[15] P. Yoeli. The logic of automated map lettering. *The Cartographic Journal*, 9:99–108, 1972.

[16] S. Zoraster. Integer programming applied to the map label placement problem. *Cartographica*, 23(3):16–27, 1986.

[17] S. Zoraster. The solution of large 0-1 integer programming problems encountered in automated cartography. *Operations Research*, 38(5):752–759, 1990.