Distributed Problem Solving by Memetic Networks

[Extended Abstract]

Ricardo M. Araujo
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre RS, 91501-970 Brazil
ricardo.araujo@gmail.com

Luis C. Lamb
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre RS, 91501-970 Brazil
LuisLamb@acm.org

ABSTRACT

This paper illustrates the use of a novel class of population-based optimization algorithms namely *Memetic Networks*. These algorithms make use of an underlying network to structure information flow between multiple individuals representing points in the search space. Memetic Networks have as a fundamental characteristic the possibility to aggregate several solutions in order to compose new ones. Network properties allow to control how information is spread among the population. We apply these algorithms to several real-valued benchmark optimization problems and the TSP and report results from extensive simulations. We show how some network properties can influence the algorithm's performance and illustrate the effectiveness of this new class of algorithms.

Categories and Subject Descriptors: I.2.8 Problem Solving, Control Methods, and Search: Heuristic methods ; I.2.11 Artificial Intelligence: Multiagent systems

General Terms: Algorithms,Experimentation

Keywords: Meme, Network, Multi-parent recombination

1. INTRODUCTION

Many evolutionary-based algorithms have as a central idea the concept of recombination between individuals in a population [4]. Recombination allows for parallel searches for solutions to communicate with each other, effectively transferring information between them that may help guide further searches. Most evolutionary algorithms that use some form of recombination implement it between two individuals (parents) selected from the population, which mimics the natural processes. It is not clear that recombination between pairs is the best approach and several models for multi-parent recombination have been proposed (e.g. [3]). As far as our knowledge goes, all previous approaches consider the number of parents as a parameter of the algorithm or fixed for all individuals of any generation.

We present a novel class of population-based optimization algorithms that extends the concept of mating to include a variable number of parents. We do so by constraining the exchange of information by an underlying network, whose nodes represent individuals and edges represent interaction possibilities. Individuals act as autonomous agents and are able to connect to other nodes following a specified rule and then exchange local information. This way, each next-generation "offspring" can be the result of the recombination of a different number of individuals at each step of the algorithm. Since our class of algorithms resembles the process of memetic evolution [1] in social networks more closely than genetic evolution, we name it *Memetic Networks*.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-130-9/08/07.

2. ON MEMETIC NETWORKS

A *Memetic Network Algorithm* (MNA) is a population-based stochastic optimization algorithm. It is composed of an ordered set of N individuals $(n_1, n_2, ..., n_N)$, each encoding a complete solution for the optimization problem, a binary $N \times N$ matrix representing possible connections between individuals and a set of rules specifying how the matrix is formed and how interaction between individuals take place. Thus an MNA's structure is a directed unlabeled graph whose nodes represent solutions to an optimization problem (individuals) and edges represent connections between these solutions. The rules are described as follows:

Connection rule: Specifies how individuals will connect to and disconnect from each other. This rule guides the construction of the network structure. The connection rule is executed at every step of the algorithm, thus the network is dynamic and connections may change at each step. It defines the dynamics *of* the network.

Aggregation rule: Given a connection, this rule specifies how information is to flow through it. It guides how the solution contained in each node is to be modified as a function of the connected nodes. It defines the dynamics *on* the network.

Appropriation rule: After information has been transmitted, this rule specifies any local changes to the information contained in a node.

Therefore, the algorithm makes use of a dynamic network during the optimization process to explicitly represent the exchange of information between several parallel searches. By defining the above rules, several types of networks can be created. This setup allows for a single node to be connected to several other nodes, thus a possible solution can influence and be influenced by several other solutions during a search. In an MNA, an unspecified and dynamic number of solutions may contribute to create a new solution, as defined by the number of connections (degree) of a node. By doing so, a more explicit and wider use of the multiple parallel searches is carried out - not only the number of "parents" is variable across generations, but it is also across offsprings.

3. PROBLEM SOLVING WITH MNA

In order to assess the performance and instantiate the proposed class, we define two simple instances of MNAs. For both versions we use the same Connection Rule:

Connection rule: node n_1 connects to node n_2 if and only if $f(\vec{x_1}) < f(\vec{x_2})$; the connection is unidirectional - information may flow from n_2 to n_1 but not conversely.

An algorithm based on this connection rule will often induce *hubs* - agents that have a higher than average number of connections. This is likely to happen because all agents will connect to a few best evaluated ones, while many badly evaluated agents will have a few (if any) incoming connections, but many outgoing connections. The first instance (henceforth *Continuous MNA*) is designed to optimize real-valued continuous functions. Each individual is composed of a vector of real-values that specify a complete solution for a function. We specify the following additional rules:

Aggregation rule: let $x_{i,j}$ be the ith component of the vector contained in node j and $C(j)$ be the set of all nodes node j connects to. Then, $x_{i,j}$ is modified to represent the average of components in position i for nodes in $C(j)$.

Appropriation rule: noise is added with probability p_n to every component of the solution in the form of a gaussian distribution with mean zero.

The second instance (henceforth *Discrete MNA*) is tailored to optimize a discrete problem, namely the Travelling Salesman Problem (TSP). For this instance, we specify the following additional rules:

Aggregation rule: the next city to compose the local path is chosen from among connected nodes to be the most frequent city that follows the previous selected city and that is not already present in the partially formed path. For example, if the first city to be visited is 1, then we query all connected nodes and count which city follows 1 most frequently in these nodes. This city will follow 1 in the new solution and the procedure is repeated until a full path has been formed.

Appropriation rule: each city in the path is swapped with a random position with probability p_n. This is akin to a simple mutation operator applied to TSP in GAs.

4. PERFORMANCE TESTS AND ANALYSIS

In order to validate the Continuous MNA, we have tested it over some functions typically used as benchmarks [2]. The chosen functions cover combinations of features regarding *modality* and *separability*. Function f_1 is the Sphere function, which is unimodal and separable; f_2 is the Rosenbrock function, which is unimodal and non-separable; f_3 is the Rastrigin function, which is multimodal and separable and f_4 is the Ackley function, which is multimodal and non-separable. All functions are 10-dimensional.

We compared our algorithm to a fairly-standard real-valued GA [5]. Each algorithm was run for a maximum of 10000 runs and we compared the average over 50 independent runs. Except for f_4, the MNA showed faster convergence rates and a better ability to escape local optima. For the unimodal functions, the MNA required about half the number of rounds to find the global optimum. For f_3, the MNA was able to find the global optimum in almost all runs, while the GA became often trapped in local optima. For f_4, however, both algorithms became trapped in local optima, but the GA found (on average) better solutions.

The Discrete MNA was applied to the *bays29* dataset taken from *TSPLib*. Even though the presented algorithm is unsophisticated, it displayed a good ability to find good solutions. Over an average of 10 independent runs and after 400 rounds, the best found tour was only 8% worse than the best possible tour. While this is not competitive with the best known techniques applied to TSP, the results shows that this general setup can be successfully applied to discrete scenarios.

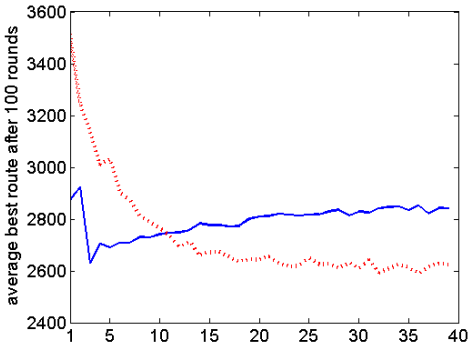

Figure 1: Best solution found after 100 rounds for different values of α averaged over all values of β (dashed line) and different values of β averaged over all values of α (solid line).

5. ANALYSING THE EFFECTS OF HUBS

Two questions arise from analyzing the aggregation of information of multiple individuals through a network. First, is it always good to aggregate as much information as possible? Second, is it good to distribute good information as much as possible? These two questions are easily translated to a network context: they ask whether it is advantageous to have nodes with a high in-degree and out-degree, respectively. In order to address these issues, we experimented with Discrete MNA over the TSP problem. We have changed our algorithm slightly by imposing limits on the in and out-degree of each node. We let α to be the maximum number of connections a node can receive (in-degree) and β as the maximum number of nodes a node can connect to (out-degree).

Our results (Fig. 1) indicate that allowing agents to receive connections from many agents (high α) is always beneficial. When severely limiting α, the ability to find good solutions degrades. However, after some value, no further improvements result from increases in this parameter. A different picture is presented when modifying the out-degree of nodes. We observe that small β values are associated to poor performance, but so are high values. The best case happens for some intermediate value (in this particular scenario, $\beta = 3$). This shows that allowing too large hubs in the system can influence negatively general performance.

Acknowledgments: This work was partly supported by CNPq-Brazil.

6. REFERENCES

[1] R. Dawkins. *The Selfish Gene*. Oxford U. P., 1976.
[2] K. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, U. of Michigan, 1975.
[3] A. E. Eiben, P.-E. Raué, and Z. Ruttkay. Genetic algorithms with multi-parent recombination. In Y. Davidor, H.-P. Schwefel, and R. Männer, ed., *PPSN III*, 1994. Springer.
[4] D. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE, 2000.
[5] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.