The Local Searcher as a Supplier of Building Blocks in Self-Generating Memetic Algorithms

N. Krasnogor, S. Gustafson
Automated Scheduling, Optimization and Planning Group
School of Computer Science and IT
University of Nottingham {nxk,smg}@cs.nott.ac.uk

Abstract

In this paper we implement a Self-Generating Memetic Algorithm for the *Maximum Contact Map Overlap Problem (MAX-CMO)*. We demonstrate how the optimization of solutions can be done simultaneously with the discovering of useful local search strategies. In turn, the evolved local searchers act as suppliers of building blocks for the evolutionary algorithm.

1 Introduction

A vast number of very successful applications of Memetic algorithms (MAs) have been reported in the literature in the last years for a wide range of problem domains. The majority of the papers dealing with MAs are the result of the combination of highly specialized **pre-existing** local searchers and usually purpose-specific genetic operators. Moreover, those algorithms require a considerable effort devoted to the tuning of the local search and evolutionary parts of the algorithm.

In [10] and [11] we propose the so called "Self-Generating Metaheuristics". Self-Generating MAs are able to **create** their own local searchers and to co-evolve the behaviors it needs to successfully solve a given problem. In Self-Generating Memetic Algorithms two evolutionary processes occur. On one hand evolution takes place at the chromosome level as in any other Evolutionary Algorithm; chromosomes and genes represent solutions and features of the problem one is trying to solve. On the other hand, evolution also happens at the memetic level, that is, the behaviors that individuals or agents will use to alter the survival value of their chromosomes. As the memes (i.e. local search strategies) are propagated, mutated and are selected in a Darwinian sense, the Self-Generating MAs we propose are closer to R. Dawkins concept of memes than previous works on memetic algorithms (e.g. [5],[14],[15],[2]).

In [10],[11],[16] it was proposed and demonstrated that the concept of Self-Generating Memetic algorithms can be implemented and, at least for the domains considered in those papers, beneficial. In the context of SGMAs, memes specify sets of rules, programs, heuristics, strategies, behaviors, or move operators the individuals in the population can use in order to improve their own fitnesses (under a given metric). Moreover the interactions between genes and memes are indirect and mediated by the common carrier of both: individuals (sometimes also called agents).

L.M. Gabora in [6] mentions three phenomena that are unique to cultural (i.e. memetic) evolution, those are, *Knowledge-based*, *imitation* and *mental simulation*. It is these three phenomena that our Self-Generating Memetic Algorithm implements and by virtue of which it can produce its own local searchers. The representation of the low level operators (in this paper the local searchers) includes features such as the acceptance strategy (e.g. next ascent, steepest ascent, random walk, etc), the maximum number of neighborhood members to be sampled, the number of iterations for which the heuristic should be run, a decision function that will tell the heuristic whether it is worth or not to be applied on a particular solution or on a particular region of a solution and, more importantly, the move operator itself in which the low level heuristic will be based[10].

The role played by local search in both Memetic and Multimeme algorithms has traditionally been associated to that of a "fine tuner". The evolutionary aspect of the algorithms is expected to provide for a global exploration of space while the local searchers are assumed to exploit current solutions and to fine tune the search in the vicinity of those solutions (i.e. exploita-

tion)

The goal of this paper is to demonstrate that local searchers can be evolved in the realm of a graph theory combinatorial problems and, more importantly, to suggest a new role for local search in evolutionary computation in general and memetic algorithms in particular: *the local searcher not as a fine-tuner but rather as a supplier of building-blocks*. For an overview of selectorecombinative evolutionary algorithms from a building-blocks perspective please refer to [7].

2 The Maximum Contact Map Overlap Problem

In this paper we explore the local searcher as a supplier of building block in the context of a problem drawn from computational biology. A *contact map* is represented as an undirected graph that gives a concise representation of a protein's 3D fold. In this graph, each residue[1] is a node and there exists an edge between two nodes if they are neighbors. Two residues are deemed neighboors if their 3D location places them closer than certain threshold. An *alignment* between two contact maps is an assignment of residues in the first contact map to residues on the second contact map. Residues that are thus aligned are considered equivalents. The value of an alignment between two contact maps is the number of contacts in the first map whose end-points are aligned with residues in the second map that, in turn, are in contact (i.e. the number of size 4 undirected cycles that are made between the two contact maps and the alignment edges). This number is called the *overlap* of the contact maps and the goal is to maximize this value. The complexity of Max CMO problem was studied in [8] and later in [10].

3 Self-Generating Memetic Algorithms for MAX-CMO

In a genetic algorithm for *Max CMO*[13], a chromosome is represented by a vector $c \in [0, \ldots, m]^n$ where m is the size of the longer protein and n the size of the shorter. A position j in c, $c[j]$, specifies that the j^{th} residue in the longer protein is aligned to the $c[j]^{th}$ residue in the shorter. A value of -1 in that position will signify that residue j is not aligned to any of the residues in the other protein (i.e., a structural alignment gap). Unfeasible configurations are not allowed, that is, if $i < j$ and $v[i] > v[j]$ or $i > j$ and $v[i] < v[j]$ (e.g., a crossing alignment)then the chromosome is discarded. It is simple to define genetic operators that preserve feasibilities based on this representation. Two-point crossover with boundary checks was used in [13] to mate individuals and create one offspring. Although both parents are feasible valid alignments the newly created offspring can result in invalid (crossed) alignments. After constructing the offspring, feasibility is restored by deleting any alignment that crosses other alignments.

The mutation move employed in the experiments is called a sliding mutation. It selects a consecutive region of the chromosome vector and adds, slides right, or subtracts, slides left, a small number. The phenotypic effect produced is the tilting of the alignments. In [13] a few variations on the sliding mutation were described and used.

In our previous work[3] we employed a multimeme algorithm that, besides using the same mutation and crossover as the mentioned GA, had a set of 6 Human-designed local search operators. Four of the local searchers implemented were parameterized variations of the sliding operator. The direction of movement, left or right sliding, and the tilting factor, i.e., the number added or subtracted, were chosen at random in each local search stage. The size of the window was taken from the set $\{2, 4, 8, 16\}$. Two new operators were defined: a "wiper" move and a "split" move.

Details of the operators described here can be found in [13],[10] and [3].

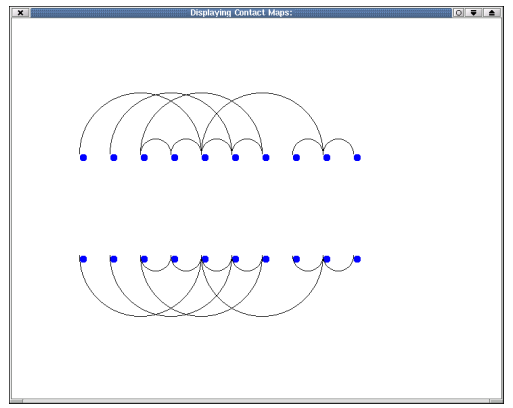
3.1 Description of Memes

As mentioned in previous sections, we seek to produce a metaheuristic that creates from scratch the appropriate local searcher to use under different circumstances. The embodiment of local searcher is done as *memeplexes*[1]. A memeplex is a co-adapted complex of memes. A meme represents one particular way of doing local search. Memes can adapt through changes in their parameter set or through changes in the actions they perform. The local search involved can be very complex and composed of several phases and processes. In the most general case we want to be able to explore the space of all possible memes. One can achieve this by using a formal grammar that describes memeplexes and by letting a genetic programming[9] based system to evolve sentences in the language generated by that grammar[10]. The sentences in the language generated by this grammar represent syntactically valid complex local searchers and they are the instructions used to implement specific search behaviors and strategies. For space limitations we do not describe here the grammar used to represent valid

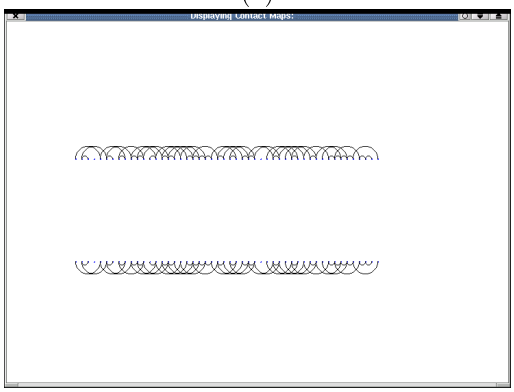
[1] A residue is a constitutent element of a protein.

memes. For details on how to achieve that the reader is referred to [10].

We concentrate instead only on the representation used to evolve the move operator itself; we resort to a few examples. In figure 1 we can see two contact maps ready to be aligned by our algorithm.To simplify the exposition, both contact maps are identical (i.e. we are aligning a contact map with itself) and have a very specific pattern of contacts among their residues. In the present example (with a given probability) a residue is connected to either its nearest neighbor residue, to a residue that is 4 residues away in the protein sequence, or to both. In 1(a) the contact map is 10 residues long, while in (b) it is 50 residues long (but with the same connectivity patterns).



(a)



(b)

Figure 1: Two contact maps snapshots. In (a) the two randomly generated proteins have 10 residues, while in (b) the patterns of contacts are maintained but the protein is 50 residues long.

This contact pattern can be represented by the string $1-4$, meaning that the residue which occupies the ith position in the protein sequence is in contact in the native state with residues $(i+1)$th and $(i+4)$th. That is, the pattern $1-4$ is a succint representation of a possible building block which, if matched by the local searcher, could be propagated later on by crossover into other solutions. An appropriate move operator for a local searcher acting in any of the contact maps on figures 1(a) and (b) would be one that iterates through every residue in one of the contact maps, checking which residues on the lower contact map fulfills the pattern of connectivity and making a list of them. The same procedure would be applied to the top contact map producing a second list of residues. The local searcher then would pair residues of one list with residues of the second list thus producing a new and correct alignment which includes that building blocks. The number of residues that verifies the pattern in each list puts an upper bound on how expensive the local search move operator can be. If the size of the first list is L_1 and that of the second L_2 and without loss of generality we assume that $L_1 \leq L_2$ then there are at most $\sum_{i=1}^{i=L_1} \frac{L_2!}{(L_2-i)!}$. Clearly this number is too big to be searched exhaustively, this is why the previous grammar allows for the adaptation of the sample size. Moreover, although it is well known that real proteins present these contact patterns[4] it is impossible to know a priori *which* of these patterns will provide the best fitness improvement for a particular pair of protein structures. Hence, the Self-Generating MA needs to discover this itself. If the graphs to be aligned were different (in the previous cases a graph was aligned with itself for the sake of clarity), then a move operator able to account for that variation in patterns must be evolved. The move operator thus defined induces a neighborhood for every feasible alignment. If an alignment s is represented as explained above and L_1, L_2 are the list of vertices that matches the move operator, then every *feasible* solution that can be obtained by adding to s one or more alignments of vertices in L_1 with vertices on L_2 is a neighbor of s. The other components of a meme will then decide how to sample this neighborhood and which solutions to accept as the next one. As this paper is an account of the initial investigations we performed on the use of SGMA, we fixed several aspects of the memes that could otherwise be evolved. That is, in this paper all memes employ first improvement ascent strategy and they are applied after crossover. The sample size was either 50 or 500 and the local search was iterated 2 times.

As described in the introduction, there were three memetic processes, namely, imitation, innovation and mental simulation. Upon reproduction, a newly created offsprings inherited the meme of one of its parents accordingly to the simple inheritance mechanism described in [12]. In addition to this mechanism, and

with a certain probability (called "imitation probability"), an agent could choose to override its parental meme by copying the meme of some successful agent in the population to which it was not (necessarily) genetically related. In order to select from which agent to imitate a search behavior, a tournament selection of size 4 was used among individuals in the population and the winner of the tournament was used as role model and its meme copied. Innovation was a random process of mutating a meme's specification by either extending, modifying or shortening the pattern in a meme (either before or after the $\rightarrow$). If during 10 consecutive generations no improvement was produced by either the local search or the evolutionary algorithm a stage of mental simulation was started. During mental simulation, each individual (with certain probability) will intensively mutate its current meme, try it in the solution it currently holds, and if the mutant meme produces an improvement, both the newly created solution and the meme will be accepted as the next state for that agent. That is, mental simulation can be considered as a guided hill-climbing on memetic space. If ten mental simulation cycles finished without improvements, then metal simulation was terminated and the standard memetic cycle resumed.

4 Experimental Setting

We designed a random instance generator with the purpose of parameterizing the complexity of the contact map overlap problems to be solved. The input to the random instance generator is a list of the form:

$r\ d\ n\ p_1\ pr_1\ p_2\ pr_2\ \ldots\ p_n\ pr_n$ where r is the number of residues in the randomly generated contact map, d is the density of random edges (i.e. noise) and n is the number of patterns in the contact map. For each of the n patterns two numbers are available, p_i and pr_i, where p_i specifies that a residue j is connected to residue $j + p_i$ with probability pr_i for all $i \in [1, n]$. That is, every pattern occurs with certain probability in each residue, thus an upper bound on the expected number of contacts is given by $r * d + r * \sum_{i=1}^{i=n} pr_i \leq r * (n + d)$. In our experiments $r \in \{10, 50, 100, 150, 200, 250\}$, $d = 0.01$ and $n \in \{1, 2, 3, 4\}$, that is, contact maps as short as 10 residues and as long as 250 residues were considered. For each contact map length, every possible number of patterns was used, this gives rise to 24 pairs of (r, n) values. For each pair, 5 random instances were generated spanning from low density contact maps to high density contact maps[2]. A total of 120 instances were generated. From all the possible parings of contact maps we randomly choose a total of 96 pairs to be aligned by means of 10 runs each.

We present next comparisons of the performance of a Genetic Algorithm versus that of the SGMA. In this experiment we would like to elucidate whether the overhead of learning suitable local searchers is amortized along the run and whether our proposed approach is ultimately useful. In order to run the experiments we implemented a GA as described previously. We were able to reproduce the results of [13] and [3] hence we considered our implementations as equivalent to the earlier ones. The difference between the GA and the SGMA are described below. In graphs 2,3,4 and 5 we compare the overlap values[3] against the *first hitting times*. First hitting time (FHT) is the time (in number of fitness evaluations) at which the best value of a run was encountered. Each graphs presents the results for 1,2,3 and 4 patterns respectively and for a range of contact maps sizes. The particular parameters used in the GA are 0.15, 0.75 for mutation and crossover probabilities, and a $(50, 75)$ replacement strategy. The Self-Generating MA uses 0.15,0.75,1.0,1.0,1.0,1.0 for the probabilities of mutation, crossover,local search, imitation, mental simulation and innovation respectively. The algorithms uses the same replacement strategy and for both local search and mental simulation a cpu budget of 50 samples is allocated.

The graphs in 2,3,4 and 5 are good representatives of the results obtained with the two types of algorithms. That is, under a variety of changes to the parameter values mentioned above the results remain equivalent to those shown here.

From figures 2,3,4 and 5 we can see that the Self-Generating Memetic Algorithm produces a much better amortized overlap value than the simple GA. That is, if enough time is given to the SGMA, it will sooner or later discover an appropriate local searcher move that will supply new building blocks. In turn, this will deliver an order of magnitude better overlaps than the Genetic Algorithm. Also, it seems that the GA is oblivious to the size (i.e. residues number) of the contact maps as it seems to produce mediocre local optima solutions even when given the maximum cpu time allocation (in these experiments $2 * 10^5$ fitness evaluations) for the whole range of 10 to 250 residues. The GA converges very soon into local optima, this is seen in the graphs by bands parallel to the $x - axis$ over the range of energy evaluations for low overlap values. On the contrary, as the SGMA continuously improves

[2]The program to generate random contact maps was written in java 1.1.8 as is available by request from the author.

[3]A higher overlap value means a better structural alignment.

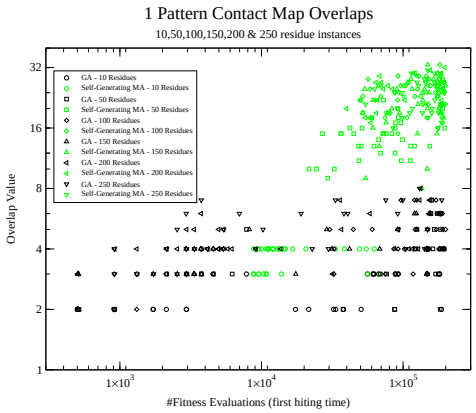


Figure 2: *Comparison of the first hitting times and the quality of overlaps obtained for GA and SGMA on increasingly difficult randomly generated instances. Complexity increases as a function of residues number. Contact maps present one pattern.*

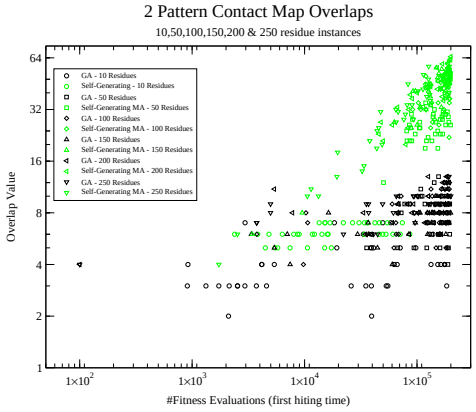


Figure 3: *Comparison of the first hitting times and the quality of overlaps obtained for GA and SGMA on increasingly difficult randomly generated instances. Complexity increases as a function of residues number. Contact maps present two patterns.*

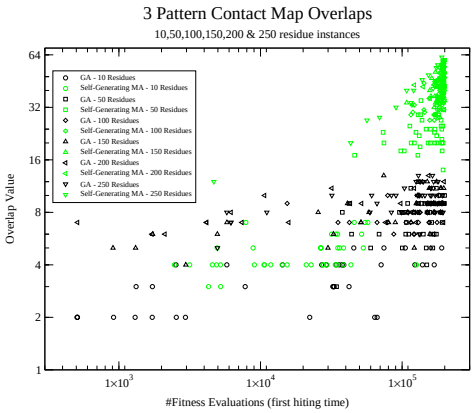


Figure 4: *Comparison of the first hitting times and the quality of overlaps obtained for GA and SGMA on increasingly difficult randomly generated instances. Complexity increases as a function of residues number. Contact maps present three patterns.*

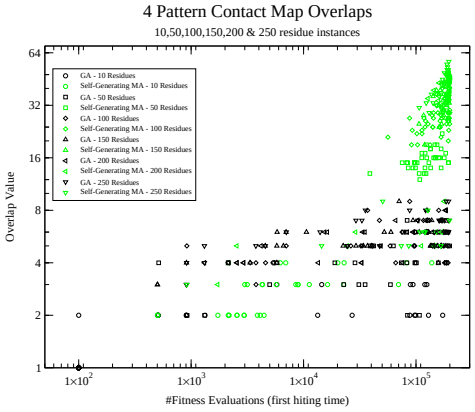


Figure 5: *Comparison of the first hitting times and the quality of overlaps obtained for GA and SGMA on increasingly difficult randomly generated instances. Complexity increases as a function of residues number. Contact maps present four patterns.*

its solutions, it is not until very late in the execution (i.e. to the right of the $x-axis$) that the best solutions are found. In contrast to the GA, the SGMA (as expected) is sensitive to the number of residues in the contact maps involved, that is, longer contact maps require larger cpu time to come up with the best value of the run (which is seen in the graph in the clustering patterns for the different residues number). Another important aspect to note is that both the $x-axis$ and the $y-axis$ are represented in logarithmic scales. Taking this into consideration it is evident that the quality of the overlaps produced by the SGMA are much better than those produce by the GA. As it is evident from the graphs, for sufficiently small instances (e.g all the 10 residues long and some of the 50 residues long) it is not worth using the SGMA as it requires more cpu effort to produce same quality of overlaps as the GA. On the other hand, as the number of residues increases beyond 50, then instances are sufficiently complex to allow for the emergence of suitable local searchers in time to overtake and improve on the GA results. Also, as the number of patterns that are present in the instances increases both algorithms, as expected, require larger amounts of CPU to come up with the best solution of a run. However, it is still seen that the GA is insensitive to the number of residues, while the SGMA is clustered in the upper right corner (of figure 5). This indicates that during all its execution the algorithm is making progress toward better and better solutions, the best of which is to be found near the end of the

run. Moreover, this behavior indicates that the SGMA is not prematurely trapped in poor local optima as is the GA.

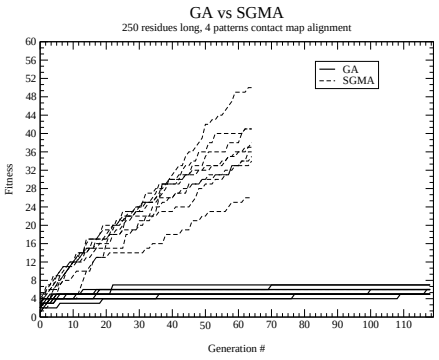


Figure 6: Representative example of GA and SGMA runs for a 250 residues and 4 patterns instance.

The ability of the SGMA to overcome local optima comes from the fact that the evolved local searchers will introduce good building-blocks that match the particular instance. This supply of building-blocks is essential for a synergistic operation of both the local searcher and the genetic operators. That is, using Goldberg's notation [7], we have that for the SGMA the *take over time* t^* is greater than the *innovation time* t_i, which allows the algorithms to continuously improve. In figure 6 10 runs of the GA are compared agaisnt 10 runs of the SGMA. It can be seen that the GA runs get trapped very early (around the 20th generation) in poor local optima while the SGMA keeps improving durin all the run. All the runs in figure 6 use the same total number of fitness evaluations.

5 Conclusions and Future Work

In this paper we introduced the concept of "Self-Generating Metaheuristics" and we exemplified its use in a hard combinatorial graph problem. The particular implementation of Self-Generating Metaheuristics used in this paper was based on Memetic Algorithms. Unlike commonly held views on Memetic Algorithms and Hybrid GAs, we do not resort here to human-designed local searchers but rather we allowed the SGMA to discover and assemble *on-the-fly* the local searcher that best suits the particular situation. A similar strategy could be use in other metaheuristics (e.g. Simulated Annealing, Tabu Search, Ant Colonies, GRASP, etc) where more sophisticated GP implementations might be needed to co-evolve the used operators. One of the reasons for the success of the SGMA is that the evolved local searchers act as a (low and medium order) building block supplier. These continuous supply of building blocks aids the evolutionary process to improve solutions continuously by producing a more synergistic operation of the local searchers and the genetic operators.

References

[1] S. Blackmore. *The Meme Machine*. Oxford University Press, 1999.

[2] E.K. Burke and A.J. Smith. A memetic algorithm for the maintenance scheduling problem. In *Proceedings of the ICONIP/ANZIIS/ANNES '97 Conference, Dunedin, New Zealand*, pages 469–472. Springer, 24-28 November 1997.

[3] B. Carr, W.E. Hart, N. Krasnogor, E.K. Burke, J.D. Hirst, and J.E. Smith. Alignment of protein structures with a memetic evolutionary algorithm. In *GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufman, 2002.

[4] T. E. Creighton, editor. *Protein Folding*. W. H. Freeman and Company, 1993.

[5] P.M. França, A.S. Mendes, and P. Moscato. Memetic algorithms to minimize tardiness on a single machine with sequence-dependent setup times. In *Proceedings of the 5th International Conference of the Decision Sciences Institute, Athens, Greece*, July 1999.

[6] L.M. Gabora. Meme and variations: A computational model of cultural evolution. In Ed by L.Nadel and D.L. Stein, editors, *1993 Lectures in Complex Systems*, pages 471–494. Addison Wesley, 1993.

[7] David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, 2002.

[8] D. Goldman, S. Istrail, and C. Papadimitriou. Algorithmic aspects of protein structure similarity. *Proceedings of the 40th Annual Symposium on Foundations of Computer Sciences*, pages 512–522, 1999.

[9] J.R. Koza, F.H. Bennet, D. Andre, and M.A. Keane. *Genetic Programming III, Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, 1999.

[10] N. Krasnogor. http://slater.chem.nott.ac.uk/~natk/public/papers.html. In *Studies on the Theory and Design Space of Memetic Algorithms*. Ph.D. Thesis, University of the West of England, Bristol, United Kingdom., 2002.

[11] N. Krasnogor and S. Gustafson. Toward truly "memetic" memetic algorithms: discussion and proof of concepts. In W. Hart J. Knowles N. Krasnogor R. Roy J.E. Smith D. Corne, G. Fogel and A. Tiwari, editors, *Advances in Nature-Inspired Computation: The PPSN VII Workshops*. PEDAL (Parallel, Emergent and Distributed Architectures Lab). University of Reading. ISBN 0-9543481-0-9, 2002.

[12] N. Krasnogor and J.E. Smith. Emergence of profitable search strategies based on a simple inheritance mechanism. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2001.

[13] G. Lancia, R. Carr, B. Walenz, and S. Istrail. 101 optimal pdb structure alignments: a branch-and-cut algorithm for the maximum contact map overlap problem. *Proceedings of The Fifth Annual International Conference on Computational Molecular Biology, RECOMB 2001*, 2001.

[14] P. Moscato. Memetic algorithms: A short introduction. In D. Corne, F. Glover, and M. Dorigo, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.

[15] P.A. Moscato. Problemas de otimizacão np, aproximabilidade e computacão evolutiva:da prática à teoria. *Ph.D Thesis, Universidade Estadual de Campinas,Brasil*, 2001.

[16] J.E. Smith. The co-evolution of memetic algorithms for protein structure prediction. In W. Hart J. Knowles N. Krasnogor R. Roy J.E. Smith D. Corne, G. Fogel and A. Tiwari, editors, *Advances in Nature-Inspired Computation: The PPSN VII Workshops*. PEDAL (Parallel, Emergent and Distributed Architectures Lab). University of Reading. ISBN 0-9543481-0-9, 2002.