Applying Memetic Algorithms to the Analysis of Microarray Data

C. Cotta[1], A. Mendes[2], V. Garcia[2], P. França[2], P. Moscato[3]

[1] Dept. Lenguajes y Ciencias de la Computación, University of Málaga,
ETSI Informática, Campus de Teatinos, 29071 – Málaga, Spain

[2] Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas C.P. 6101, 13083-970, Campinas, Brazil

[3] School of Electrical Engineering and Computer Science
University of Newcastle, Callaghan, NSW, 2308, Australia

Contact email: `ccottap@lcc.uma.es`

Abstract. This work deals with the application of Memetic Algorithms to the Microarray Gene Ordering problem, a NP-hard problem with strong implications in Medicine and Biology. It consists in ordering a set of genes, grouping together the ones with similar behavior. We propose a MA, and evaluate the influence of several features, such as the intensity of local searches and the utilization of multiple populations, in the performance of the MA. We also analyze the impact of different objective functions on the general aspect of the solutions. The instances used for experimentation are extracted from the literature and represent real biological systems.

1 Introduction

Due to the huge amount of data generated by the Human Genome Project, as well as what is being compiled from other genomic initiatives, the task of interpreting the functional relationships between genes appears as one of the greatest challenges to be addressed by scientists [13]. The traditional approach of Molecular Biology was based on the "one gene in one experiment" scenario. This approach severely limits understanding "the whole picture", making the ramified paths of gene function interactions hard to track. For this reason, new technologies have been developed in the last years. In this sense, the so-called *DNA microarray* technique [5, 7] has attracted tremendous interests since it allows monitoring the activity of a whole genome in a single experiment.

In order to analyze the enormous amount of data that is becoming available, reduction techniques are clearly necessary. As a matter of fact, it is believed that genes are influenced on average by no more than eight to ten other genes [3]. To achieve such a reduction, and allow molecular biologists concentrate on a sensible subset of genes, clustering techniques such as k-means, or agglomerative methods can be used (see [8, 9] for example). However, there is still much room for improvement in the solutions they provide.

Memetic algorihms [16] (MAs) have been recently proposed as a tool for aiding in this process [15]. In this work we consider the application of MAs to a related variant of this clustering problem: the Gene Ordering problem. It consists of finding a high-quality rearrangement of gene-expression data, such that related (from the point of view of their expression level) genes be placed in nearby locations within a gene sequence. This problem is motivated by the usual bidimensional representation of microarray data, and hence it combines aspects of clustering and visualization. Here, we concentrate our efforts in analyzing the influence of several features of the MA –such as the intensity of local searches and the configuration of the population– in the performance of the algorithm. The impact of different objective functions on the quality of the solutions is also thoroughly tested. This is a critical issue, especially because the ordering quality is a relative attribute, strongly connected to the visual aspect.

The remainder of the article is organized in three sections: first, the definition of the problem, and the application of MAs to this domain are detailed in Section 2; next, the results of an extensive computational experimentation involving the parameters of the algorithm are reported in Section 3; finally, a brief summary of the main contributions of this work, and some prospects for future work are provided in Section 4.

2 Memetic Algorithms

In this section we will discuss the implementation of the Memetic Algorithm. The MA is a population-based algorithm that uses analogies to natural, biological and genetic concepts, very similarly to Genetic Algorithms (GAs) [4]. As in the latter, it consists of making a population of solutions evolve by mutation and recombination processes. Additionally, a distinctive feature of MAs is the use of local search operators in order to optimize individual solutions. The best fitted solutions of the population will survive and perpetuate their information, while the worse fitted will be replaced by the offspring. After a large number of generations, it is expected that the final population would be composed of highly adapted individuals, or in an optimization application, high-quality solutions of the problem at hand. Next, we briefly explain some specific MA aspects.

2.1 The Fitness Function

Before getting into the details of the fitness function, some comments must be made about the nature of the data defining the problem. For our purposes, the output of a microarray experimentation can be characterized by a matrix $G = \{g_{ij}\}$, $1 \leq i \leq n$, $1 \leq j \leq m$, where n is the number of genes under scrutiny, and m is the number of experiments per gene (corresponding to measurements under different conditions or at different time points). Roughly speaking, entry g_{ij} (typically a real number) provides an indication on the activity of gene i under condition j.

Now, as mentioned in Section 1, a good solution in the Gene Ordering problem (i.e., a good permutation of the genes) will have similar genes grouped together, in clusters. A notion of distance must thus be defined in order to measure similarity among genes. We have considered a simple measure, the Euclidean distance (other options are possible; see e.g [18]). We can thus construct a matrix of inter-gene distances. This matrix will be used for measuring the goodness of a particular gene ordering.

The most simple way of doing this is calculating the total distance between adjacent genes, similarly to what is done in the Traveling Salesman Problem. A drawback of such an objective function is that, since it only uses information of adjacent genes, it has a very narrow vision of the solution. For a better grouping of the genes, the use of 'moving windows' is a better choice. The total distance thus becomes a two-term sum, instead of a single-term one. The first term sums up the distances between the window's central gene and all others within the window's length. The second one sums up those partial distances as the window moves along the entire sequence. Moreover, in order to give more weight to the genes that are closer a multiplying term was added to the function. Let $\pi = \langle \pi_1, \pi_2, \cdots, \pi_n \rangle$ be the order of the n genes in a given solution. Then, the fitness function is:

$$fitness(\pi) = \sum_{l=1}^{n} \sum_{i=\min(l-s_w,1)}^{\max(l+s_w,n)} w(i,l) D[\pi_l, \pi_i] \tag{1}$$

where $2s_w + 1$ is the window size (the number of genes involved in each partial distance calculation), and $w(i,l)$ is a scalar weighting the influence of the gene located at position l on the gene located at position i. We have considered weights proportional to $s_w - |l - i| + 1$ (i.e., linear in the distance between the genes), and normalized so as to have the sum of all weights involved in each application of the inner sum in Eq. (1) be 1. The use of this function gives higher ratings to solutions where genes are grouped in larger sets, with few discontinuities between them. As a matter of fact, the function makes in some sense the assumption that every gene could be the center of a cluster, subsequently measuring the similarity of nearby genes. The global optimization of this function is thus expected to produce robust solutions. Section 3 will provide results regarding the use of this function, and the effect of varying the window size.

2.2 Representation and Operators

The representation chosen for the Gene Ordering problem takes some ideas from hierarchical clustering. To be precise, solutions are represented as a binary tree whose leaves are the genes. By doing so, solutions are endowed with extra information regarding the level of relationship among genes, information that would be missing in other representations that only concentrated on the actual leaf order (e.g., permutations). This way, reproductive operators such as recombination and mutation can be less disruptive.

We have used a pre-order traversal of the tree to store it into individuals. More precisely, the chromosome is a string of integers in the $[-1, n-1]$ interval, where n is the number of genes to be ordered. Each of the genes (leaves of the tree) is identified with a number in $[0, n-1]$, and internal nodes of the tree with the value -1 (all of them are indistinguishable). It is easy to see that the length of the chromosome will thus be $2n-1$ genes.

Having defined this representation, adequate operators must be defined to manipulate it. Considering firstly the recombination operator, we have used an approach similar to that used in the context of phylogeny inference [6]. To be precise, we perform recombination by selecting a subtree T from one of the parents, removing any leaf present in T from the second parent (to avoid duplications), and inserting T at a random point of the modified parent. As to mutation, it is based on gene-sequence flip. A subtree is selected uniformly at random, and its structure is flipped. This way, the entire sequence belonging to it is flipped too. This mutation preserves the gene grouping inside the subtree, introducing some disruption in its boundaries.

2.3 Population Structure

The use of hierarchically structured populations boosts the performance of the GA/MA (see [10]). There are two aspects that should be noticed. First, the placement of the individuals in the population structure according to their fitness. And second, a well-tailored selection mechanism that selects pairs of parents for recombination according to their position in the population. In our approach, the population is organized following a complete ternary tree structure. In contrast with a non-structured population, the complete ternary tree can also be understood as a set of overlapping 4-individual sub-populations (that we will refer to as *clusters*).

Each cluster consists of one *leader* and three *supporter* individuals. The leader individual always contains the best solution of all individuals in the cluster. This relation defines the population hierarchy. The number of individuals in the population is equal to the number of nodes in the complete ternary tree, i.e. we need 13 individuals to make a ternary tree with 3 levels, 40 individuals to have 4 levels, and so on. The general equation is $(3^n - 1)/2$, where n is the number of levels. Previous tests comparing the tree-structured population with non-structured approaches are present in [10]. They show that the ternary-tree approach leads to considerably better results, and with the use of much less individuals.

2.4 Reproduction

The selection of parents for reproduction is an important part of the algorithm, since the quality of the offspring will depend basically on this choice. Based on the assumption that a ternary tree is composed of several clusters, we adopted the restriction that recombination can only be made between a leader and one of its supporters within the same cluster. The recombination procedure thus selects any leader uniformly at random and then it chooses –also uniformly at random– one of the three supporters. Indirectly, this recombination is fitness-biased, since individuals situated at the upper nodes are better than the ones at the lower nodes. Therefore, it is unlikely that high-quality individuals recombine with low-quality one, although it might happen a few times.

The number of new individuals created every generation is equal to the number of individuals present in the population. This recombination rate, apparently high, is due to the offspring acceptance policy. The acceptance rule makes several new individuals be discarded. In our implementation, a new individual is inserted into the population only if it is better than one of its parents. This is a very restrictive rule and causes a quick loss of diversity among the population. Nevertheless, the impact on the MA of this scheme was noticeable, making the algorithm reach much better solutions using less CPU time. After each generation the population is restructured to maintain the hierarchy relation among the individuals. The adjustment is done comparing the supporters of each cluster with their leader. If any supporter is found to be better than its respective leader, they swap their places.

It must be emphasized that this scheme must be used together with a check procedure for premature population convergence, in order not to waste CPU time. The check procedure implemented verifies the number of generations without improvement of the incumbent solution. If more than 200 generations have passed and no improvement was obtained, we conclude that the population has converged and apply local search on all individuals. Moreover, the local search is also carried out every time a new incumbent solution is found through recombination/mutation. Such an event signalizes a new starting point, where the application of a local search could improve that incumbent solution even more. In this second case, it is also very likely that the rest of the population is well-fitted too, since it also usually requires several generations to find, just by recombination and mutation, a better solution than the incumbent, especially if the incumbent has already gone through at least one local search process.

2.5 Local Search

Generally, local searches utilize neighborhood definitions to determine which movements will be tested. A somewhat common neighborhood for sequences is the *all-pairs* [10]. In our case, it should be equivalent to test all possible position swaps for every gene, keeping the movements that improve the fitness. This local search turns out to be very computationally expensive since the evaluation of the fitness is costly. We thus tried the lightest swap local search type, the pairwise interchange, which only tests pairs of adjacent genes for swap. This neighborhood resulted in a reasonable computational cost, with an acceptable improvement in terms of solution quality.

The other local search implemented acts at the clustering tree structure level, by inverting the branches of every subtree present in the solution, much like it is done during mutation. Such an inversion is a good choice to make radical changes in the chromosome without loosing information about the gene grouping. As a matter of fact, this local search is very successful in joining together separated groups that contain similar genes. Both local searches are applied sequentially: the branch-inversion local search is applied on the initial solution, and the resulting individual goes through the gene-swap local search. Notice that each local search can be iterated a number of times until no change is detected in the individual (i.e., it is at a local optimum). However, notice that the fitness improvement achieved by performing repeated passes might not be worth the associated computational cost. For this reason, the number of passes each local search performs on an individual is an important parameter of the MA that has been studied in Section 3.

Another important matter is to decide which individuals should go through local search. In the Gene Ordering problem, for instance, the application of local search on every new individual is simply too costly and the algorithm wastes a lot of time optimizing individuals that are not worth it. In our implementation, the application of the local search takes place only after convergence of the population. This reduces the number of local searches and guarantees that most individuals are promising, since the population must have evolved for many generations before converging. Again, the influence of the number of individuals that are subject to local optimization will be empirically studied in Section 3.

2.6 Island Model

The use of the so-called island model [17] is known to be advantageous for several reasons: firstly, the use of semi-isolated populations helps preserving diversity, and increases the chances of escaping from local optima; secondly, it leads to a natural parallelization scheme, in which different populations are assigned to different processors. For these reasons, we have studied the influence of using multiple populations in the MA performance for this application. To be precise, we consider the effect of having an increasing number of populations cooperating with each other.

For the study of multi-population MAs, we had to define how individuals migrate from one population to another. The migration policy adopted states that populations are arranged in a ring structure. Moreover, migration occurs in all populations and the best individual of each one migrates to the population right next to it, replacing a randomly chosen individual (except the best one). This way, every population receives only one new individual, after all populations have converged and the local search phase has ended.

3 Computational Results

First of all, Table 1 describe the instances used in this work. They were extracted from real gene sequences available in the literature. Although the high number of configurations analyzed has precluded including larger instances in the test-suite, the different sizes of these instances allowed testing the algorithm in diverse optimizations scenarios.

Table 1. Instances considered in this work.

instance	genes	experiments	description
HERPES	106	21	Kaposi's sarcoma-associated herpesvirus gene expression [12]
LYMPHOMA	380	19	Selectively expressed genes in diffuse large B-cell lymphoma [2]
FIBROBLAST	517	18	Response of human fibroblasts to serum [11]

The times allotted to these instances were 30s, 300s, and 500s respectively. These times are roughly proportional to n^2, where n is the number of genes, due to the growth trend of the local search cost. In each case, the initial population of the MA has been fed with the solutions provided by three classical clustering algorithms: complete-linkage, average-linkage, and single-linkage clustering (see [9] for example). The experiments have been done using the NP-Opt Framework [14], running on a PC (Pentium IV - 1.7 Ghz, 256 MB RAM) under Windows XP and Java build 1.4.1_01-b01.

The first study addresses the different window sizes (s_w), which were fixed at 1, 1%, 5%, 10% and 20% of the instance size. These tests tried to adjust the window size in order to find solutions with a good general aspect. Figure 1 shows some results for the FIBROBLAST data set. The results for the remaining data sets reflect the same behavior.

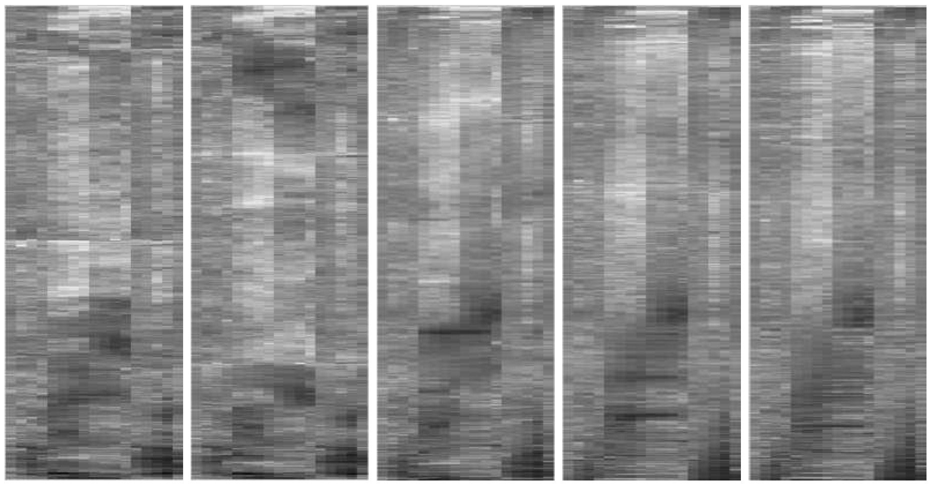

Fig. 1. Median results (over ten runs of the MA) for different sizes of the window (FIBROBLAST instance). From left to right, window size = 1, 1%, 5%, 10%, and 20%. Light (respectively dark) dots represent data points in which a particular gene is expressed (respectively non-expressed).

The tests indicated that the TSP-like fitness function, with a window size of 1, provide poor results. The problem is that, as said before, this function has a narrow view of the instance, what makes similar genes be grouped apart. This behavior can also be seen in the 1% window size. The best results came when the window is at least 5% of the instance. Up from this value, groups begin to be well-formed, with soft transitions between them. The larger window sizes provide good results, but we observed a light increase in the overall noise for very large window sizes, partly because the higher computational cost of the fitness function precludes a finer optimization within the allotted time. Given these results, we decided to use a window fixed at 5% of the instance size in the remaining experiments, as a good tradeoff between quality and computational cost.

The next tests were oriented to set the best policy for applying the local search methods. Given that the population is structured as a tree, we firstly considered to which levels local search was going to be applied. Secondly, we had to decide the number of passes n_p for each local searcher. Subsequently, the tests included applying local search according to the following configurations:

– Only on the root (i.e., just on the best individual), n_p =1, 4, and 13.
– On the two first levels (i.e., on the best four individuals), n_p =1, and 3.
– On all three levels (i.e., on the entire population), $n_p = 1$.

The local search was thus tested on a total of six configurations, being applied only when the population converges. The number of passes indicates the number of times that the gene-swap and the tree-swap local searches would be sequentially applied on the individual, at maximum. Logically, if a pass does not improve the present individual, the local search ends promptly. Since we used the same CPU time for all configurations, the maximum number of local search passes was fixed. Following this, every time a population converges, 13 passes were carried out, at maximum. The results are shown in Table 2.

Table 2. Results (averaged for ten runs) of the MA for different intensities of local search.

	1 level			2 levels		3 levels
Instance	$n_p = 1$	$n_p = 4$	$n_p = 13$	$n_p = 1$	$n_p = 3$	$n_p = 1$
HERPES	600.102	603.930	604.335	600.011	599.893	598.769
LYMPHOMA	2609.274	2607.666	2609.619	2610.593	2620.339	2622.640
FIBROBLAST	1376.804	1386.560	1390.353	1382.191	1398.143	1407.402

The configuration offering the best overall tradeoff was to apply the local search only on the best individual, making a single pass. This configuration is the one with less local search effort, leaving more CPU time for the recombination and mutation phases. This indicates that genetic operators are very important for the problem and contribute in a strong manner to find good solutions.

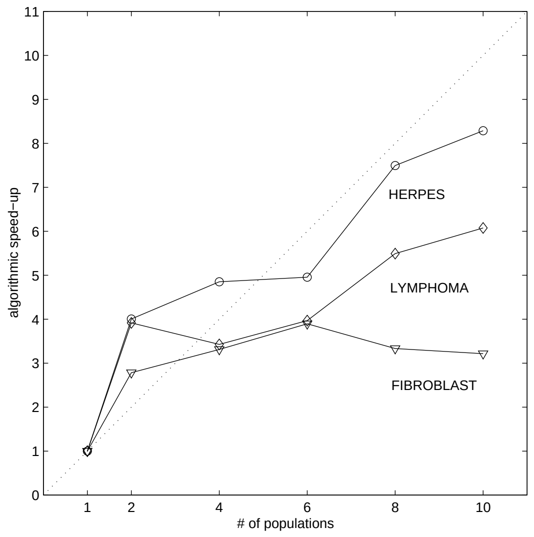

Fig. 2. Algorithmic speedup of the MA for different number of populations.

The last parameter to be tested was the number of populations. The tests were carried out for 2, 4, 6, 8 and 10 populations. These tests were performed in a sequential environment, and their focus was on the study of the algorithmic speedup of the MA. By algorithmic speedup we refer to an upper bound of the actual computational speedup that could be achieved were the MA deployed on a physically distributed environment. This bound is computed as follows: firstly, a desired fitness level is selected. subsequently, all configurations of the MA are run until a solution at least as good as desired is found. The corresponding times (in sequential model, i.e., by sequentially running each population for one step until a synchronization point is reached) are measured. By normalizing these times dividing by the number of populations we obtain the ideal time the MA would require to produce such a solution in a distributed environment, i.e., assuming negligible communication times. As mentioned before, this provides an upper bound of the achievable speedup with respect to the single-population MA, since in practice communications will have non-zero cost[4]. The results are shown in Fig. 2

As it can be seen, excellent algorithmic speedups can be obtained for up to four populations. Beyond this point, the performance starts to degrade for the larger instances, specially for FIBROBLAST. Nevertheless, very good speedups are possible for HERPES, and at least moderately good for LYMPHOMA. This suggests that migration plays an important role in the search process. In fact, migration helps in the way it reduces the effect of premature convergence. More-

[4] We do not consider here other factors that may affect this speedup in a positive way. For example, the distribution of populations can result in a better use of local caches, thus making distributed populations run faster that sequential ones.

over, given the ‘genetic-drift’ effect, larger portions of the search space can be scanned in a clever way. Notice also that for some configurations there is super-linear speedup. This is a known effect due to the different behavior of a multi-population evolutionary algorithm (EA) with respect to a single-population EA [1]. As a matter of fact, such as behavior is precisely one of the reasons supporting the use of these non-panmictic MAs.

4 Conclusions

This work presented a memetic algorithm for finding the best gene ordering in microarray data. The main features of this MA are the use of a fitness function capturing global properties of the gene arrangement, the utilization of a tree representation of solutions, the definition of *ad-hoc* operators for manipulating this representation and the use of multiple populations in a island model migration scheme. Two different local search procedures have also been used.

Several configurations for the MA parameters were tested. The window size, used by the fitness function to measure the distance between a specific gene and its neighbors; the local search policy, which defined how local search should be applied on the population; and finally the number of populations utilized by the MA.

Besides showing how the MA can improve over classical agglomerative clustering algorithms, the results indicate that the window size should be proportional to the instance size. Moreover, the local search should be applied in a light way, giving time for the algorithm to complete several generations. This result reinforces the quality of the genetic operators, which apparently play important roles in the evolutionary process. Finally, the use of multiple populations results in better overall performance over the single population approach.

Future work will be directed to test the MA in a physically distributed environment, in order to confirm the results presented. This is important with respect to the deployment of the algorithms on larger problem instances too. We are actively working in this topic. Preliminary results support the scalability of the approach. We are also interested in using some smart weighting scheme within the fitness function, so as to giving lower relevance to outliers. Statistical tools should be used to this end. In this line, the possibility of utilizing some sort of *annealing* scheme to change the window size dynamically could be considered as well.

Acknowledgements

Carlos Cotta is partially supported by Spanish MCyT, and FEDER under contract TIC2002-04498-C05-02. Alexandre Mendes is supported by *Fundação de Amparo à Pesquisa do Estado de São Paulo* (FAPESP - Brazil). This work was also partially supported by *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq - Brazil).

References

1. E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, 82(1):7–13, 2002.
2. A.A. Alizadeh et al. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403:503–511, 2001.
3. A. Arnone and B. Davidson. The hardwiring of development: Organization and function of genomic regulatory systems. *Development*, 124:1851–1864, 1997.
4. T. Bäck, D.B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, New York NY, 1997.
5. P.O. Brown and D. Botstein. Exploring the new world of the genome with DNA microarrays. *Nature Genetics*, 21:33–37, 1999.
6. C. Cotta and P. Moscato. Inferring phylogenetic trees using evolutionary algorithms. In J.J. Merelo et al., editors, *Parallel Problem Solving From Nature VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 720–729. Springer-Verlag, Berlin, 2002.
7. J.L. DeRisi, V.R. Lyer, and P.O Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278:680–686, 1997.
8. M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences of the USA*, 95:14863–14868, 1998.
9. D. Fasulo. An analysis of recent work on clustering algorithms. Technical Report UW-CSEO1-03-02, University of Washington, 1999.
10. P.M. França, A.S. Mendes, and P. Moscato. A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*, 132(1):224–242, 2001.
11. V.R. Iyer et al. The transcriptional program in the response of human fibroblasts to serum. *Science*, 283:83–87, 1999.
12. R.G. Jenner, M.M. Alba, C. Boshoff, and P. Kellam. Kaposi's sarcoma-associated herpesvirus latent and lytic gene expression as revealed by DNA arrays. *Journal of Virology*, 75:891–902, 2001.
13. E.V. Koonin. The emerging paradigm and open problems in comparative genomics. *Bioinformatics*, 15:265–266, 1999.
14. A.S. Mendes, P.M. França, and P. Moscato. NP-Opt: An optimization framework for NP problems. In *Proceedings of POM2001 - International Conference of the Production and Operations Management Society*, pages 82–89, 2001.
15. P. Merz. Clustering gene expression profiles with memetic algorithms. In J.J. Merelo et al., editors, *Parallel Problem Solving From Nature VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 811–820. Springer-Verlag, Berlin, 2002.
16. P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics.* Kluwer Academic Publishers, Boston, 2002.
17. R. Tanese. Distributed genetic algorithms. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–439, San Mateo, CA, 1989. Morgan Kaufmann.
18. H.-K. Tsai, J.-M. Yang, and C.-Y. Kao. Applying genetic algorithms to finding the optimal gene order in displaying the microarray data. In W.B. Langdon et al., editors, *Proceedings og the 2002 Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, 2002.