Gene Ordering in Microarray Data Using Parallel Memetic Algorithms

Alexandre Mendes[1], Carlos Cotta[2], Vinícius Garcia[3], Paulo França[3] and Pablo Moscato[1]

[1]Newcastle Bioinformatics Initiative, University of Newcastle, Callaghan, NSW, 2308, Australia
[2]ETSI Informática, University of Málaga, Campus de Teatinos, 29071 - Málaga, Spain
[3]DENSIS–FEE, Universidade Estadual de Campinas C.P. 6101, 13083-970, Campinas, Brazil

Abstract

This paper addresses the Microarray Gene Ordering problem. It consists in ordering a set of genes, grouping together the ones with similar behavior. This behavior can be measured as the gene's activity level across a number of measurements. The Gene Ordering problem belongs to the NP-hard class and has strong implications in genetic and medical areas. The method employed is a Memetic Algorithm, which is a variant of the well known Genetic Algorithms. The algorithm employs several features like population structure, problem-specific crossover and mutation operators, local search, and parallel processing. The instances utilized are extracted from the literature and represent real systems with 106 up to 979 genes. The algorithm has a superior performance, successfully grouping the genes. Moreover, in this paper we evaluate the impact of parallel processing in the performance of the algorithm, especially for the larger instances, which required more computational effort.

1. Introduction

The use of the microarray technology [10] has constituted one of the most impressive breakthroughs in Molecular Biology. With it, we can monitor the activity of a whole genome in a single experiment. Enormous amounts of data are thus increasingly becoming available thanks to the utilization of these microchips. Interpreting the underlying relationships among the genes whose expression is being monitored poses a formidable challenge from a computational point of view. Consider that microarray experiments involve from hundreds up to tens of thousands of genes, with usually tens of measurements per gene. Clearly reduction techniques are in order in this context (as an illustration of the need for reduction, it is believed that genes are influenced on average by no more than eight to ten other genes [3].) Such reduction techniques try to group together genes with related expression patterns since such genes are likely to regulate each other, or be co-regulated. Clustering techniques can be used for this purpose (see e.g. [5, 11, 12, 15].) However, there is still much room for improvement in the solutions they provide.

In this work we propose the utilization of memetic algorithms (MAs) [18] as a tool for aiding in this process. More precisely, we consider the use of MAs for finding a high-quality re-arrangement 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. As it has been shown by [8], these high-quality re-arrangements can be very useful for improving the performance of clustering algorithms. This use of MAs results in a computation-intensive application whose execution in a sequential environment may require untenable computational resources. For this reason, we have considered the parallelization of the algorithm. This has been done using a master-slave model deployed on a local network of computers.

2. Microarray Data Analysis

DNA microarrays consists of large numbers of DNA molecules spotted in a systemic order on a solid substrate. A microarray experiment consists of exposing these DNA spots to complementary DNA (cDNA) –a process called hybridization– obtained from messenger RNA (mRNA). These mRNA molecules are marked with fluorescent dye visible under a microscope (usually red and green for a target and a reference sample, respectively.) Due to the complementary nature between DNA and cDNA molecules, they couple together by means of base-pairing. Of course, cDNA molecules not corresponding to any gene in the microarray will be uncoupled, and can be easily removed. After this process, the microarray is scanned, measuring the red/green fluorescence of each spot. This fluorescence value indicates the level of expression of the corresponding gene with respect to each sample.

The result of a microarray experiment can be expressed as a matrix $G = \{g_{ij}\}_{j=1\cdots m}^{i=1\cdots n}$, where n is the number of genes, and m is the number of experiments per gene. These experiments correspond to the state of the gene under different conditions or at different time points. The goal now is finding an optimal order of genes such that genes with similar expression patterns are close in this order. Implicit in this definition of the problem is a notion of "distance" among genes. A number of distance measures have been proposed in the literature. For example, centered Pearson correlation has been used by [19]. Other correlation measures such as Kendall's τ correlation or Spearman Rank correlation can be used as well. In this work we have considered a simple distance metric: the Euclidean distance. As usual, this metric is defined as $D[g_i, g_j] = \sqrt{\sum_{k=1}^{m} (g_{ik} - g_{jk})^2}$. At any rate, notice that the methodology presented in this work does not rely on a particular choice of distance measure.

Once a distance measure has been defined, and matrix D has been computed, several approaches can be used to find an adequate gene order. For example, a hierarchical clustering algorithm can be used in the first place. A popular approach for doing this clustering is arranging all genes in a list of singletons $S_i = \{g_i\}$, $1 \leq i \leq n$, and iteratively merge the two "closest" groups until only one group remains. The order in which these groups are joined determines the topology of the tree. Common methods for measuring the distance $c(S, S')$ among groups S and S' are the single-link, average-link, and complete-link (see [12].)

After having obtained a hierarchical clustering, a gene ordering can be found in different ways. On one hand, internal nodes can be flipped so as to obtain a good ordering consistent with the tree topology [4]. [6] have proposed a polynomial algorithm for finding the best of such orderings for a TSP-like distance measure (the optimality criterion is minimizing the sum of distances between adjacent genes in the sequence,) given the hierarchical clustering structure. This TSP-like optimality criterion has been also used by [14] in the context of a self-organizing map algorithm, and by [19] in the context of a genetic algorithm. This simple optimality criterion can serve as a first approximation to the best ordering, but it is not capable of grasping the global aspect of the resulting sequence as shown below.

3. Memetic Algorithms

In this section we will discuss the implementation of the MA. The MA is a population-based algorithm that uses analogies to natural, biological and genetic concepts, very similar to a Genetic Algorithm (GA). The 'Memetic' terminology refers to 'memes' [9], or portions of cultural information that can be transmitted among the individuals of a population. In other words the MAs share characteristics of genetic evolution and cultural evolution. These algorithms are also known as 'Local Search GA' or 'Hybrid GA'. The MA employed in this work has some features that increases its performance. Among them, we should cite the hierarchical population structure, multiple population model, and a pairwise-type local search.

3.1 The Fitness Function

The fitness function plays an important role in the evolutionary process, by determining the quality of an individual. Therefore, it must be strongly attached to the characteristics wanted in high-quality solutions. For the Gene Ordering problem, a good solution will have similar genes grouped together, in clusters. As mentioned in the previous section, the most simple is to calculate the total distance between adjacent genes, similarly to what is done in the Traveling Salesman Problem when one tries to minimize the total distance. Let $\pi = \langle \pi_1, \pi_2, \cdots, \pi_n \rangle$ be the order of the n genes in a given solution. Then, the total distance between adjacent genes can be described as the $\sum_{i=1}^{n-1} D[\pi_i, \pi_{i+1}]$.

A drawback of this objective function is that, since it only uses information of adjacent genes, it has a very narrow vision of the solution. This makes some solutions, where genes are grouped in small disjoint sets, be classified as very good. For a better grouping of the genes, a wider vision is necessary. The use of 'moving windows' is a better choice in this case. The total distance becomes a two-term sum. The first one sums up the distances between the window's central gene and all others within the window's length. The second term sums up those partial distances as it moves the window along the entire sequence. This function can be represented as:

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

where $2s_w + 1$ is the window size (the number of genes involved in each partial distance calculation.) The use of such function gives higher ratings to solutions where genes are grouped in larger sets, with few discontinuities between them. Nevertheless, the distance between any two genes within the window has the same weight on the objective function value. This is not a good feature, since intuitively, the weight for closer genes should be higher than for farther ones. This variance of weight can be achieved by adding a term multiplying the distance, making the total distance be calculated as:

$$total_{dist}(\pi) = \sum_{l=1}^{n} \sum_{i=\max(l-s_w,1)}^{\min(l+s_w,n)} (s_w - |l-i| + 1) D[\pi_l, \pi_i] \qquad (2)$$

This function returned the best results and thus was used to evaluate the quality of the solutions in the rest of this

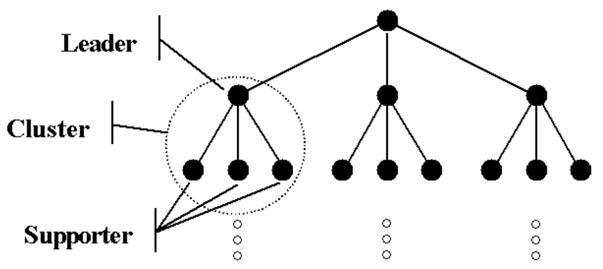


Figure 1. Diagram of the population structure.

work. Experiments focusing on the effect of the window size indicate that a size of window between $s_w = 5\%$ and $s_w = 10\%$ of the instance size is a good tradeoff between solution quality and computational cost [7]. It should be also emphasized that although the use of the 'moving windows' function was a major breakthrough, the correct adjustment of the MA also played an important role. If other methods utilized the same function, the results should also be good, but the extensive testing on local search techniques and crossover strategies were determinant for the quality of the results found.

3.2 Population Structure

The use of hierarchically structured populations boosts the performance of the GA/MA (see [13].) 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*.)

As shown in Fig. 1, each cluster consists of one *leader* and three *supporter* individuals. Any leader individual in an intermediate layer has both leader and supporter roles. 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.

The choice of the ternary tree structure was based mainly on empirical aspects. The first is motivated by the fact that any hierarchical tree behaves like a set of overlapping populations, as said before. Therefore, the dynamics is similar to four populations evolving in parallel - each cluster acts as an independent population - and exchanging individuals at a given rate. This individual exchange comes as a consequence of the tree re-structuring phase, carried out to maintain the hierarchical consistence.

A binary tree population would be formed by 3-individual clusters only, with only two recombinations possibilities. This would degrade the 'multiple population' character of the tree structure. Trees with a greater order - quaternary or more - increase the multiple population character, but initial tests indicated that the quality does not improve proportionally and moreover, the number of individuals rapidly jumps to prohibiting levels in terms of computational effort requirements. The best trade-off points to the selected ternary tree structure (see [13].)

3.3 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). It is pursued to have the MA exploiting this information, evolving sensible hierarchies of genes. This way, reproductive operators such as crossover and mutation can be less disruptive.

Selection is based on the position of the individuals in the ternary tree. 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 (the Prune-Delete-Graft operator, check [7]) thus selects any leader uniformly at random and then it chooses - also uniformly at random - one of the three supporters. Indirectly, this recombination is also 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 crossover rate, apparently high, is due to the offspring acceptance policy. The acceptance rule makes several new individuals be discarded. The acceptance of new solutions will be later discussed (see Subsection 3.5.) As to mutation, it is based on gene-sequence flip. A subtree is selected uniformly at random and the entire sequence belonging to it is flipped. This mutation preserves the gene grouping inside the subtree, thus creating very little noise. The tree structure is also flipped, to keep consistency with the gene sequence change

3.4 Local Search

The local search in this problem plays a 'fine-tune' role. While the recombination and mutation operators will be successful in finding the overall aspect of the sequence, subtle changes are usually out of reach of the genetic operators. For instance, if a given gene sequence can only be improved by a single swap of genes, the genetic operators will probably not find such a swap. Even if they are successful in swapping the genes, this operation will probably create too much noise in the other parts of the chromosome, erasing any chances for improvement. In such cases, a local search is the best choice for such fine-tuning movements. Generally, local searches utilize neighborhoods definitions to determine which movements will be tested.

A somewhat common neighborhood is the all-pairs. In our case, it should be equivalent to test all possible position swaps for every gene, i.e, for each gene, try to swap its position with all other ones, 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 needed a smaller neighborhood. The lightest swap local search type is the pairwise interchange, which only tests pairs of adjacent genes for swap. This neighborhood reduction results 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. The recombination operator could do the same job, but given its stochastic nature, it would require several trials to obtain a successful move, and depending on the size of the instance, such a success move could take simply too long to be achieved. Both local searches are applied once or twice, depending on whether or not in the first pass the solution was improved, and sequentially. The branch-inversion local search is applied on the initial solution and the resulting individual goes through the gene-swap local search.

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 is on the entire population, but only after its convergence. 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. Moreover, we noticed that when the population converges, the individuals are similar, but not equal, thus validating the application of the local search on all of them. The convergence criterion is discussed in next subsection.

3.5 Offspring Insertion into the Population

Every time a new offspring is created it is tested for insertion into the population. In this work, the acceptance of the new individuals will follow two rules:

- The offspring is inserted into the population replacing the supporter that took part in the recombination that generated it.
- The replacement occurs only if the fitness of the new individual is better than the supporter.

These acceptance rules are very restrictive and cause 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. Such hierarchy states that the fitness of the leader of a cluster must be lower than the fitness of the leader of the cluster just above it. 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 place.

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.

4. Parallelization Scheme

The performance if the MA is strongly dependent on the two local searches defined. Both of them require n evaluations of the individual, where n is the number of genes.

Since each evaluation is $O(kn)$, where k is the size of the window, we have a final complexity of $O(kn^2)$. This is extremely time-consuming considering the sizes of the instances used in this work. Actually, preliminary tests indicated that the local search was responsible for over 90% of the total CPU time during the search process, making it the key candidate for parallelization. This has been precisely the approach used in this work, performing several local searches in parallel by distributing them over a network of computers. The parallelization of other operators - like selection, recombination or mutation - is not interesting since they consume a negligible CPU time compared to the local search.

The distribution of the local search effort is simple. It employs a canonical master-slave architecture, with the master machine controlling the MA flow, assigning the processes to the slave-machines and receiving the optimized individuals that are sent by the slaves. Since the distribution routine was implemented using Java Threads, the sockets control, object serialization, etc., are directly managed by the Java system, thus simplifying the programming task. Overall traffic through the network is minimized because only the individual's chromosome is transferred. The local search routine and the instances are locally accessed by each machine, producing a lower overhead communication.

As said in the previous section, the local search phase takes place only when the population has converged, i.e., when all individuals present in the population are well-fitted. The optimization phase starts by putting all the individuals in a request queue, and each individual is sent to the next slave that is idle, one each time, through the correspondent socket. The sockets are opened at the first time the local search is called, and are not closed until the algorithm ends. When a slave finishes its job, it returns the resulting individual to the master machine through the same socket, which puts it in a response queue. When the request queue is empty and the response queue is complete, the local search phase ends and the MA continues. That means the next generation begins only when all the individuals have gone through local search and were received back by the master machine, in a 100%-synchronous master-slave architecture.

It must be noted that this parallelization is purely computational, and does not affect the behavior of the algorithm since local search is deterministic. This implies that the classic speedup measure can be used to assess the effects of the parallelization. This contrasts with other approaches for parallelizing evolutionary algorithms such as using multiple populations distributed over a network, with migrating individuals. In this latter case the behavior of the parallel algorithm would be clearly different from its sequential counterpart, and hence the classical measure can lead to anomalous results such as super-linear speedups [1].

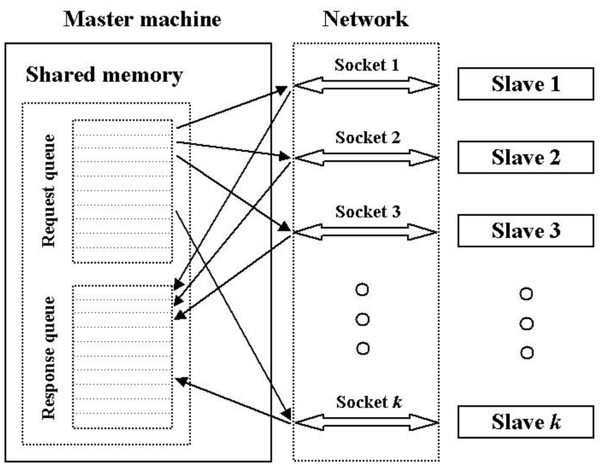


Figure 2. Master-Slave architecture employed to distribute the local search effort.

5. Computational Results

The computational tests evaluated the performance of the master-slave architecture, as well as the quality of the final solutions obtained. The experiments were realized using a Java implementation of the MA (build 1.4.0-b92). The algorithm was run on a network of SUN Workstations (UltraSparc Iii – 440Mhz, 256 Mb RAM, 48 Gb de HDD,) communicating through a 100-Mbit, Ethernet network, and running under UNIX (Solaris 2.8) operating system. Four instances were tested. These are the following:

- HERPES: this data set is taken from [17]. It comprises expression levels for 106 genes (21 experiments per gene.) These data were used to describe Kaposi's sarcoma-associated herpes virus gene expression during latency and after the induction of lytic replication.

- LYMPH: this data set is taken from [2]. It comprises gene expression levels for 380 genes (19 experiments per gene) corresponding to selectively expressed genes in diffuse large B-cell lymphoma.

- FIBRO: this data set is taken from [16]. It comprises expression levels for 517 genes (18 experiments per gene) corresponding to the response of human fibroblasts to serum.

- YEAST: this data set is taken from [4], which in turn extracted it from [11]. It comprises expression levels for 979 genes (79 experiments per gene) known

Table 1. CPU time utilized in the MA, CPU percentage required to apply local search (LS), and theoretically achievable asymptotical speedup.

	CPU time (s)	LS %	max. speedup
HERPES	106	66.22%	2.875
LYMPH	380	88.63%	8.794
FIBRO	517	91.42%	11.658
YEAST	979	95.72%	23.373

to participate in some complex creation in Yeast cell-cycle.

These instances cover a wide range of sizes, and allows testing the algorithm in different optimizations scenarios. In each case, the initial population of the MA has been fed with the solutions provided by the three clustering algorithms mentioned in Section 2. The tests were carried out using one up to 13 machines as slaves, which is the number of individuals in the population. The performance is measured in two ways. The first is the speed increment itself, in terms of speedup. The second is the quality of the final solution obtained, in terms of improvement over the sequential approach result with the same CPU time.

First, we show in Table 1, the instances' characteristics, the CPU time utilized for them and also the percentage of this time that is devoted to local search in a sequential run. This last information is relevant because the MA distributes the local search to the slaves, and the larger this percentage, the greater the theoretical speedup achievable. This theoretical speedup can be computed assuming no communication time, and infinite processors using the well-known formula:

$$max.\ speedup = \frac{1}{1-p} \tag{3}$$

where p is the percentage of the CPU time that can be parallelized (assumed to be infinitely divisible for distribution purposes.) Of course this is an overoptimistic estimation, but it can provide a first bound on the achievable results.

The times allotted to each instances are roughly proportional to n^2, where n is the number of genes, due to the growth trend of the local search cost. Notice the increasing percentage of CPU time devoted to local search as the number of genes increase.

The communication time between machines is very low. For the larger instance it is less than one second. That is due to the characteristics of the network and also to the amount of data navigating in the network. As only individuals are transported, the maximum size of a package sent from one machine to another is around 8KB - a YEAST-type individual with 979 genes, whose chromosome has 1957 positions, each of them represented by an integer-type value (32-bit). Next we present in Table 2 the relation between the number of machines and the number of individuals theoretically optimized by each one of them. The resulting theoretical speedup can then be calculated using the following formula:

Table 2. Description of the slaves' work load. As more slaves are used, each slave becomes responsible for the optimization of less individuals, increasing the theoretical maximum speedup value of local search (LS).

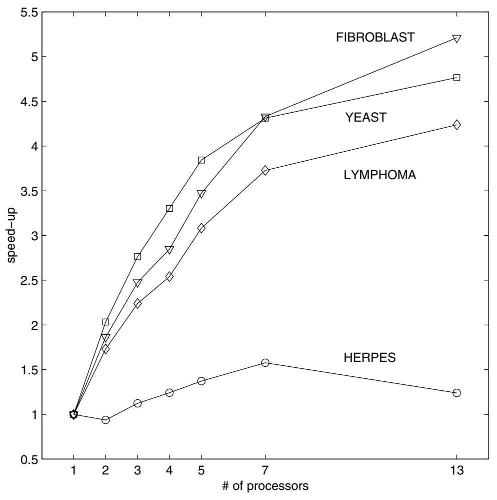
# proc	# individuals per proc (D)	$w(m)$
1	{13}	1.000
2	{6, 7}	1.857
3	{4, 4, 5}	2.600
4	{3, 3, 3, 4}	3.250
5	{2, 2, 3, 3, 3}	4.333
7	{1, 2, 2, 2, 2, 2, 2}	6.500
13	{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}	13.000

$$speedup(m) = \frac{1}{1-p+p/w(m)} \tag{4}$$

where p is the percentage of computational time devoted to local search, m is the number of machines, and $1/w(m)$ is the maximum fraction of the local search effort assigned to any machine ($w(m) = \frac{\Sigma D}{\max(D)}$.) This later factor accounts for the fact that the local search phase is not freely divisible, but can only be subdivided in the application of local search to isolated individuals. The actual values for $w(m)$ can be seen in Table 2.

Each configuration tested has a different maximum number of individuals evaluated by a machine. The testing of such cases only, aims to check how the algorithm took advantage of the workload reduction when the number of machines increased. It is important to consider that the theoretical LS speedup is calculated assuming negligible communication times, and more crucially, that all local searches have the same computational cost. This later assumption has turned out to be unrealistic, and hence a deviation from this speedup is expected. Of course, the actual speedup will be under these values due to the fact that there is a non-parallelized portion of the algorithm (recall Equation (4)).

Figure 3 (left) shows the speedup increase for each instance, illustrating these tests. The results are quite illustrative to explain how parallel processing works. The smallest instance has a very small speedup for two reasons: firstly, local search is only a small fraction of the CPU time for this instance, and hence the speedup cannot be very high (recall



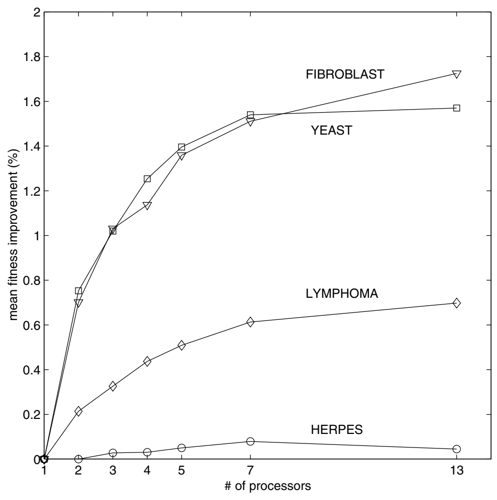


Figure 3. (Left) Speedup results (averaged for ten runs) for the four instances tested. (Right) Mean fitness improvement over the sequential algorithm.

Table 1); secondly, local search times rival with the communication time between machines. In fact, when 13 computers are utilized, the speedup even decreases due to the communication overhead. The next two instances returned expected results, with the speedup increasing smoothly up to the 13-machine configuration. The 979-gene instance, on the other hand, had a complex behavior. Its speedup results were worse than the ones obtained by the 517-gene instance for 7 and 13 machines despite the fact that local search takes a larger portion of the CPU time. The reason for this behavior is the different time required to apply local search to different individuals, as mentioned before. We verified that for the largest instance, the difference between the local search times required by two solutions might top 400%. For 7 and 13 individuals, such difference is very harmful, since the master-machine waits until the last process was completed to go on with the algorithm. This situation is very similar to those found in 'bin-packing'-related problems: when the "bins" (total time used by a slave to complete assigned tasks) are large with respect to the size of "items" (individual times for each task) a rather balanced solution can be found; as these quantities become closer, solutions tend to be more unbalanced. Nevertheless, the speedup was considerably good for up to five machines (these anomalies can even be positive sometimes for this small number of processors since more balanced workload distributions can be achieved,) with a relative performance decrease for the last two configurations only.

The second performance criterion was the improvement of the final solution obtained by the parallel approaches over the sequential one. The results are shown in Figure 3 (right). Although the percentage values obtained may appear to be small, the improvement is easily noticeable when we take into account the visual aspect of the solutions. Furthermore, they are about 10% better than basic clustering algorithms. Indeed, the MA did not find trouble in identifying large clusters. Moreover, an important aspect of these solutions is that the transition between different groups is very smooth, a result derived from the fitness function utilized, which penalizes rough transitions.

As expected, the improvement increases with the number of machines, except for the pathological cases of the 106-gene instance, which were already explained. The behavior of the curves is smoothly asymptotic, with few ups-and-downs, what is a sign of algorithmic robustness. Moreover, it also indicates that the network is very reliable, with efficient, fast communication between machines.

6. Conclusions

We have 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, and the definition of *ad-hoc* operators for manipulating this representation. Two different local search procedures have been used. As it has been shown in Section 5, the local search phase can take up to 95% of the computational time in a 979-gene instance. Based on this finding, a master-slave parallelization model of the algorithm has been proposed.

From the point of view of the quality of the results, the MA provides solutions with a smooth visual appearance.

Focusing on the efficiency of the parallelization, the computational results have been encouraging, since good speedups have been obtained with up to 5 processors. This good speedup is coupled with a sustained improvement of the solution quality on the basis of a constant computational time. When more processors are used, the efficiency of the parallel model starts to drop, due to inherent unbalance in the workload sent to the slaves. This suggests several lines for future developments.

First of all, the use of larger populations, or even multipopulation MAs can be considered. By doing so, it would be possible to achieve a more balanced distribution of tasks among slaves, since there would be a finer granularity in the parallelized portion of the MA. Additionally, and connected to the use of multiple populations, it was mentioned in Section 4 that an island-based distribution of a multi-population MA (i.e., distributing populations in different processors) is compatible with this master-slave parallelization model of the local search phase. Furthermore, we have shown that such island-based MAs can provide remarkable algorithmic speedups [7]. Thus, we could think of having the computer network organized in several groups of computers: in each of these groups a computer would be in charge of running a population, and the remaining ones would serve as slaves for the local search phase of this population. This way, it would be possible to combine two levels of parallelism, making a joint exploitation of the good speedups of the master-slave model for a moderate number of processors, and the diversified search performed by partially isolated populations. We are currently conducting further research in this line.

Acknowledgements

This work was partially supported by Spanish MCyT, and FEDER under contract TIC2002-04498-C05-02, and by Brazilian CNPq.

References

[1] E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, 82(1):7–13, 2002.

[2] 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] Z. Bar-Joseph, D. Gifford, and T. Jaakkola. Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics*, 17(1):22–29, 2001.

[5] A. Ben-Dor and Z. Yakhini. Clustering gene expression patterns. In *Proceedings of the ACM RECOMB'99*, pages 33–42, Lyon, France, 1999. ACM Press.

[6] T. Biedl, B. Brejova, E. Demaine, A. Hamel, and T. Vinar. Optimal arrangement of leaves in the tree representing hierarchical clustering of gene expression data. Technical Report 2001-14, University of Waterloo, 2001.

[7] C. Cotta, A. Mendes, V. Garcia, P. França, and P. Moscato. Applying memetic algorithms to the analysis of microarray data. In G. Raidl et al., editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 22–32. Springer-Verlag, Berlin, 2003.

[8] C. Cotta and P. Moscato. A memetic-aided approach to hierarchical clustering from distance matrices: Application to gene expression clustering and phylogeny. *Biosystems*, 72(1–2):75–97, 2003.

[9] R. Dawkins. *The Selfish Gene*. Clarendon Press, Oxford, 1976.

[10] J. DeRisi, V. Lyer, and P. Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278:680–686, 1997.

[11] M. Eisen, P. Spellman, P. 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.

[12] D. Fasulo. An analysis of recent work on clustering algorithms. Technical Report UW-CSEO1-03-02, University of Washington, 1999.

[13] P. França, A. 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.

[14] L. Gomes, F. von Zuben, and P. Moscato. Ordering microarray gene expression data using a self-organising neural network. In A. Lotfi, J. Garibaldi, and R. John, editors, *Proceedings of the 4th International Conference on Recent Advances in Soft Computing*, pages 307–313, Nottingham, UK, 2002. The Nottingham Trent University.

[15] E. Hartuv, A. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach, and R. Shamir. An algorithm for clustering cDNAs for gene expression analysis. In *Proceedings of the ACM RECOMB'99*, pages 188–197, Lyon, France, 1999. ACM Press.

[16] V. Iyer et al. The transcriptional program in the response of human fibroblasts to serum. *Science*, 283:83–87, 1999.

[17] R. Jenner, 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.

[18] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 105–144. Kluwer Academic Publishers, Boston MA, 2003.

[19] 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. Langdon et al., editors, *Proceedings of the 2002 Genetic and Evolutionary Computation Conference*, pages 610–617, San Francisco, CA, 2002. Morgan Kaufmann.