Efficient Evolutionary Algorithms for the Clustering Problem in Directed Graphs

C. Rodrigo Dias
I.C. - Universidade Federal Fluminense
156 Rua Passo da Pátria, Bloco E, 3 andar
Niterói, RJ, Brasil 24210-240
cdias@ic.uff.br

Luiz S. Ochi
I.C. - Universidade Federal Fluminense
156 Rua Passo da Pátria, Bloco E, 3 andar
Niterói, RJ, Brasil 24210-240
satoru@dcc.ic.uff.br

Abstract- This paper presents improvements in the performance of standard genetic algorithms (GAs) as regards the solution of highly complex combinatorial optimization problems. These improvements are related to some modifications in the GA, including local search and/or diversification procedures. The performance of each proposed version is evaluated through a graph partitioning problem. Extensive computational experiments show that our evolutionary algorithms outperform a genetic algorithm proposed in the literature, by significantly improving the quality of the final solutions with similar computational times.

1 Introduction

The clustering problem corresponds to the process of grouping elements in a set so that the formed groups, or clusters, represent a configuration in which each element is more similar to any element from the same group than to the elements from the other groups. There are several applications for clustering in several research fields and it has been exhaustively approached in the literature and, more recently, it has been used in data mining (Berkhin 2002), biotechnology (Hartuv et al. 1999), software engineering (Doval et al. 1999), and others.

There are several clustering techniques in the literature and some are more suitable for certain types of applications. Among the most explored techniques are the hierarchical and partitioning methods.

In hierarchical clustering algorithms the clusters are formed progressively by the grouping or disassociation of elements or clusters, respectively, forming a cluster hierarchy which is represented through a cluster tree. The process of cluster formation is interrupted when the number of wanted clusters is obtained (if this number has been predefined) or in case any other stopping condition happens. The lack of refinement in the grouping or disassociation process usually gives a greedy character to the traditional hierarchical method.

On the other hand, in the clustering algorithms that use any partitioning method, the initial set of elements is divided into several subsets and the obtained configuration is evaluated. If the evaluation of the configuration indicates that it does not solve the problem in question, then a new configuration is obtained through the migration of elements among the clusters and the process continues in an iteractive way until some stopping criterion is reached.

In the partitioning methods, the evaluation of all the possible cluster configurations is computationally unfeasible, thus limiting the exclusive use of exact methods for its solution. Therefore, heuristics or approximated methods have been frequently proposed and, more recently, the use of evolutionary algorithms in the application to clustering problems (Cole 1998; Chiun & Lan 2001) has been growing, especially the genetic algorithms (GAs). Although GAs are often used in the literature in problems of combinatorial optimization, the performance of these, in their traditional form, is worse than other metaheuristics. With the aim of attempting to improve the performance of GAs, different versions of a traditional GA have been proposed.

Among the most used variations of GAs, we can mention the proposals of adding new genetic operators and new mechanisms for the creation of the initial population, as well as for the selection process and the individuals' reproduction. Therefore, a significant improvement in the quality of the obtained solutions has been reached by the evolution of the GAs (Drummond et al. 1997; Drummond et al. 1998a; Drummond et al. 1998b; Ochi & Rocha 2000; Ochi et al. 2001).

In that evolution, Glover (Glover 1977) introduced the evolutionary approach entitled Scatter Search, which presents similar and complementary characteristics to GAs, and which has been shown to be promising in solving optimization problems (Laguna 1999; Glover & Laguna 2000; Martí et al. 2002). In the Scatter Search approach, the main idea is to work with a more deterministic version of GAs, by using the idea of linear combination of good solutions (élite solutions) obtained during algorithm iterations. In another proposal of evolution of GAs, Moscato (1989) introduced the memetic algorithms (MAs), whose formal representation was presented by Radcliffe & Surry (1994). In MAs, a local search is carried out in order to be applied to the whole population or only to some individuals alongside the algorithm iterations. Because of the high cost of application of a local search to a great number of individuals, MAs are better used when the local search is occasionally applied to a small number of individuals only, and in case the fitness function can be decomposed. Thus, a small alteration in an individual during the local search will not need to update the

individual's fitness as a whole, only in regard to the part affected by the alteration.

Another variation of GAs, proposed by Lorena & Furtado (2001), corresponds to the Constructive Genetic Algorithms (CGA), which add new characteristics to the traditional proposal of GA. Among the main aggregated characteristics is the association of a rank to each individual, bearing in mind that an individual can be constituted of blocks of partial solutions or a complete solution, and the use of a bi-objective fitness function, which is used to evaluate the complete solutions and the blocks of partial solutions that are called schemata.

A bottleneck of GAs is the required computational time that is usually longer than the time required for other heuristics. Some of the alternatives to reduce this limitation is to gauge its parameters appropriately, like the size of the population, or to work with parallel versions of GAs (Drummond et al. 1997; Drummond et al. 1998a; Drummond et al. 1998b; Ochi et al. 2001).

The main goal of this work is to present procedures so as to obtain a more efficient GA. In order to achieve that, a traditional GA, as proposed by Doval et. al. (1999) for clustering in graphs, was initially implemented and then different fittings were suggested and new modules to GA were incorporated in an attempt to improve the obtained solutions without a significant increase in the execution time. The contribution of this paper is the investigation of the extent to which an GA can be improved by alterations that do not compromise its global acting.

This article is organized as follows: in Section 2 the graph partitioning problem is described and the genetic algorithm is presented, as proposed by Doval et. al. (1999). In Section 3, the new proposals are described. Section 4 presents information about the implementation, the obtained computational results and their analysis. Conclusions and future proposals are presented in Section 5, followed by the bibliography.

2 Graph Partitioning

The clustering process applied to graphs is also reported in the literature as graph partitioning problem. In this case, the problem concerns the grouping of the nodes of a graph in disjoint clusters, so that the nodes that are more strongly connected are gathered in a same cluster, while the connections among the nodes of different clusters are minimized (Doval et al. 1999; Hartuv & Shamir 1999; Maini et al. 1994). The clustering problem in graphs is a NP-COMPLETE problem, what justifies the use of heuristics, with special attention to GAs.

The traditional GA implemented in this work, and referred to in this paper as basic GA (BGA), was proposed by Doval et. al. (1999) in order to automatically obtain a good partitioning (or clustering) of a module dependency graph (MDG).

A MDG is a way used in software design to make complex systems more comprehensible. It corresponds to a directed graph where the modules of a system are represented by the nodes, and the static dependencies among the modules are represented by the edges of the graph. In a MDG it is possible to have an edge connecting a node to itself.

In contrast with most of the methods used for clustering, in the proposed BGA it is not necessary to previously specify the number of clusters into which the MDG should be partitioned. The choice of the best number of clusters is part of the process of the problem solution. Therefore, this variant of the graph partitioning problem is more complex than the models by which the number of clusters is previously defined.

During the execution of BGA, each individual of the population corresponds to a feasible solution, even with different number of clusters in each of them. In order to evaluate the quality of a solution corresponding to a directed graph, the authors present a measure that takes the several connections among the nodes of the graph into consideration.

The measure, denominated modularization quality (*MQ*), is used as the fitness function of the GA. The objective of the GA is to find a good (possibly the best) partitioning through the maximization of *MQ*, which is defined as

$$MQ = \begin{cases} \frac{\sum_{i=1}^{k} A_i}{k} - \frac{\sum_{i,j=1}^{k} B_{i,j}}{\frac{k(k-1)}{2}} & \forall k > 1 \\ A_i & k = 1 \end{cases}$$

where A_i is the intra-connectivity of cluster i, $B_{i,j}$ is the inter-connectivity between clusters i and j and k is the total number of clusters of the solution.

The intra-connectivity A_i of a cluster i is a measure that considers the total number of edges inside each one of the clusters of the solution (density of the cluster), being defined as

$$A_i = \frac{\mu_i}{N_i^2}$$

where μ_i is the total number of internal edges of the cluster i and N_i is the total number of its nodes.

The inter-connectivity $B_{i,j}$ between clusters i and j is a measure that considers the total number of edges between the pairs of clusters in a solution. Considering a couple of clusters i and j, ε_{ij} being the total edges from cluster i to cluster j, N_i and N_j being the total nodes of clusters i and j, respectively, the measure of inter-connectivity $B_{i,j}$ of the pair of clusters is given by

$$B_{i,j} = \begin{cases} 0 & if \quad i = j \\ \frac{\varepsilon_{ij}}{2N_i N_j} & if \quad i \neq j \end{cases}$$

The values of the intra-connectivity and inter-connectivity vary between 0 and 1 and a good quality clustering has a high value for the sum of the intra-connectivity of all clusters and a low value for the sum of the inter-connectivity of every possible pair of clusters.

The value of *MQ* corresponds to the difference between the intra-connectivity average and the inter-connectivity average and can vary between –1 and 1. The function *MQ* rewards the accomplisment of clusters with high intra-connectivity value and penalizes partitionings with many dependencies among its clusters (high inter-connectivity value), i.e., the higher the value of *MQ*, the better the clustering.

As an example, Figure 1 indicates the value of *MQ* for two different clusterings for a same graph with 8 nodes, where clustering A is better than clustering B.

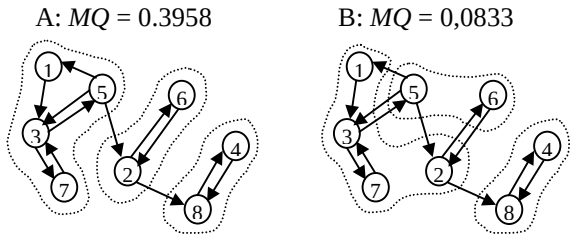

Figure 1: two different samples of clustering

The representation used in order to make possible to associate each BGA individual to a solution for the clustering problem is the encoding presented as group-number in Cole (1998) .

In the group-number scheme, the clustering of *n* objects is represented by an array of *n* integers where the *i*th integer indicates the number of the group which holds the *i*th element. Therefore, in the graph partitioning, an array of integers with size *n* is used for each individual, where *n* is the total number of nodes in the graph. Figure 2 shows an individual representing a clustering solution for a graph with 8 nodes.

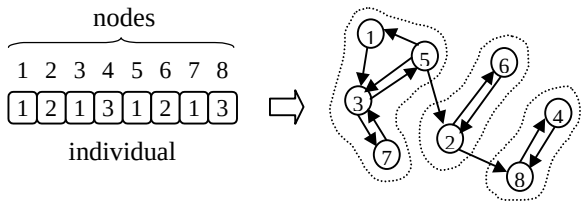

Figure 2: an example of group-number encoding

In BGA, Doval et. al. (1999) propose that the number of individuals from the populations alongside the generations is kept, with a total number of $10 \times n$ individuals, where *n* is the total number of nodes of the graph.

However, they only analyzed small graphs, with less than 100 nodes. By using this approach, the size of the population can become very large for graphs with a high number of nodes, as in the case of nodes in the order of hundreds or thousands.

Because of that, when implementing BGA in this work, the size of the population was assumed as being [*max*{100, *n*}], which empirically was showed to be a satisfactory value, in comparison with the initial proposal. As specified in the original BGA, the initial population is randomly generated in this work, being possible to find individuals that correspond to solutions with different number of clusters.

After the generation of the initial population, the evolutionary process begins, getting different individuals' populations alongside the generations. Each generation corresponds to an iteration of GA, when the selection and reproduction steps of a population first take place so as to generate a new population.

During the selection and reproduction of a population, *n* individuals are chosen from the current population according to their fitness values. The selection in BGA uses the roulette method, complemented by elitism, the latter being used in order to guarantee that the fittest individual of the current population is selected for the next one. These *n* selected individuals form the new population, which will be used in the next iteration of the GA. In BGA, Doval et. al. (1999) specify the total number of iterations carried out as being $200 \times n$, where *n* is the number of nodes of the graph. According to this definition, the number of iterations increases a lot as the graph grows. Because of that, in this work half of the proposal ($100 \times n$) was adopted - what has continued to produce satisfatory results.

Even after adjusting the number of iterations, this number can be high and the execution time can be long. For that reason, we adopted a second stopping criteria, which restricts the execution time of the algorithm to a maximum of 4 hours.

In each generation, immediately after the selection and reproduction, the crossover operator is applied to combine pairs of individuals with the aim of obtaining new individuals. Figure 3A presents an example of the application of the crossover operator, where the individuals use the group-number encoding.

The mutation operator is applied to each individual of the population after the application of the crossover operator. In the mutation operator, the value of each *i*th element of the individual, with $1 \leq i \leq n$, has the same probability of being changed to a random value *q*, so that $1 \leq q \leq n$, i.e., the maximum number of clusters will be the number of elements (i.e. nodes) of the related problem (i.e. the graph). The rate of application of the mutation operator is $0.004 \times \log_2(n)$. An example of the mutation operator is shown in Figure 3B.

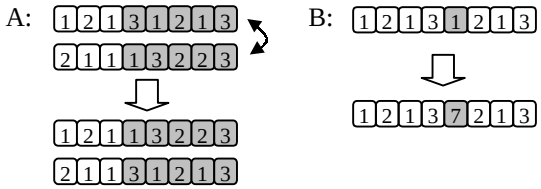

Figure 3: crossover (A) and mutation (B) operators

3 Improving the Performance of BGA

The results obtained from the execution of BGA in preliminary tests were shown to be unsatisfactory for directed graphs with some dozens of nodes, what can be observed in Section 4. Despite the unsatisfactory results of BGA, it was verified that its *MQ* function allows a good evaluation of the quality of a solution, even without a previous definition of the amount of clusters in the searched solution.

Based on the analysis of the behavior of BGA in this preliminary battery of tests, it was possible to identify its downside. Then, we propose alternatives to try to improve the performance of the BGA as regards the quality of the results obtained. The proposals consist in optimizing parameters and developing new modules to be added to the BGA.

The proposals are implemented in different versions of BGA, and we can note the following: a calibration mechanism in the number of clusters considered during the mutation operator; the insertion of a local search procedure; a routine to generate a new heterogeneous population based on the information held in the best solution found until the moment it is generated. Each of the new proposed versions is described as follows:

Version 1 (GA1)

It corresponds to a version of the BGA with the insertion of a local search procedure to be carried out in each iteration after the application of the mutation operator to the best individual of the population. The objective of the local search procedure is to attempt to refine the best individual in each generation, exchanging the nodes among the clusters.

The process of exchanging the nodes is simple and accomplished in each i*th* element of the array of the solution, through the substitution of its value. In such process each node is selected once and it is obtained the cluster to which it shares the most number of edges. If the obtained cluster is different from its current it is made an evaluation of the individual considering the exchanging of the node from the current cluster to the obtained cluster. If this exchange improves the individual's fitness it is accomplished, in other way, the node stays in its current cluster.

Because of the nature of the proposed solution for the partitioning graph problem, it is not possible to decompose the fitness function and, therefore, to each exchange it is necessary a new evaluation of the whole individual and not just of the altered element. This demands a high computational cost and, for this reason, the local search procedure will just be applied to an individual of each generation: the best one.

Version 2 (GA2)

This version is based on the GA1, with the introduction of a way to calibrate the number of clusters of a solution through the mutation operator. In order words, to each generation, considering *nc* as the number of clusters of the best solution obtained up to the current generation, the random value *q* of the cluster obtained for each selected element that will be altered in the operator mutation will be obtained in the range $1 \leq q \leq (1.1 \times nc)$.

The aim of this alteration is to allow a progressive adjustment in the maximum number of clusters of the solutions considered in each generation, which should converge to the optimal number of clusters, while the local search procedure is used to speed up this convergence.

Version 3 (GA3)

This version uses the local search procedure introduced in GA1 and the calibration of the number of clusters in the mutation process, introduced in GA2. However, in this version a modification was accomplished in the local search procedure. As it was exposed in GA1, to each application of the local search procedure to an individual, a node can be transposed from a cluster to another once only and, at the end of the local search procedure, a new clustering configuration can be achieved.

The proposal of GA3 is that, immediately after the conversion of the nodes in a local search, and consequent new clustering configuration, the local search procedure should be re-applied to verify if new exchanges of nodes among the clusters can get better solutions than that obtained in the previous application of the local search procedure.

In this new proposal, this re-application of the local search procedure should happen while the obtained solution improves. This process will be referred to from now on as *re-application of the local search procedure*.

Version 4 (GA4)

That version uses the local search introduced in the GA1 with the re-application proposed in the GA3, together with a new proposal. This new proposal consists in applying the local search procedure to the best individual of this population after the ramdom generation of the initial population.

After that, a new population is generated by a diversification module, in which the best solution obtained up to the current generation (called global best) is duplicated *m* times (where *m* is the size of the population) and in each copy of the global best, we select one or more windows where the size and the positions of these are randomly chosen. Then, each element into the windows is changed to values randomly selected. In the example of the Figure 4, a population with five individuals is created with copies of the global best, inserting random values in the windows of random sizes and positions.

The aim of this procedure is to investigate an area of search space close to the best individual obtained up to the current iteration (called global best), though without losing the stochastic character of the procedure. It is important to mention that the other operators, namely crossing and mutation, continue to be used in the way defined in BGA.

In the other iterations, whenever a new global best is obtained, the diversification process is activated.

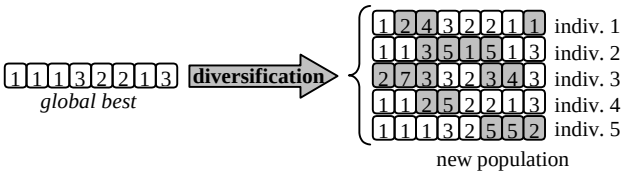

Figure 4: example of the diversification procedure

In GA4, the diversification procedure always uses the fittest individual obtained after the execution of the local search. In other words, even if in a given iteration the global best is updated several times by the used operators in the reproduction stage or during the local search, only at the end of the iteration is the diversification activated (if it is the case).

Version 5 (GA5)

This version has the same characteristics of the GA4, including the diversification module, only differing in the number of times that the diversification module can be executed in each iteration of the algorithm.

As in GA4, the diversification module is activated once after the generation of the initial population and once at each update of the global best. Then, in GA4 the use of the diversification module could obtain a better solution than the global best, what is not considered because the next step is to select the individuals of the obtained diversified population for the next iteration, without updating the global best again. GA5 was implemented in order to consider that possibility, i.e., after the execution of the diversification module in the update of the global best, the obtained population will be evaluated again and the local search will be applied to the best individual of the population.

In case the best individual obtained by the last local search is better than the global best, this will be updated again and the diversification module will be re-activated. This process will be repeated while the diversification module, followed by the application of the local search, update the global best.

This repetitive process will be referred to in this work as *diversification with repetition* and it is worth observing that it should not be considered as being an iteration of the evolution of GA5.

Version 6 (GA6)

This version is based on GA4. However, an alteration was accomplished in its mutation operator, introducing the calibration mechanism in the estimated number of clusters introduced in version GA2.

The aim of this alteration is to allow a gradual adjustment in the maximum number of clusters of the solutions considered in each generation, converging on the optimal number of clusters. At the same time, the re-application of the local search is used to accelerate this convergence and the use of the diversification module investigates an area of the search space.

Version 7 (GA7)

This version is based on the GA5, including the calibration mechanism in the number of clusters introduced in GA2, as performed in GA6.

Summary of the versions

Table 1 summarizes, in a comparative way, the characteristics presented in this section, where a mark (X) in the cell indicates that the respective module is incorporated.

Procedures	GA Versions							
	B	**1**	**2**	**3**	**4**	**5**	**6**	**7**
number of clusters calibration			X	X			X	X
local search		X	X	X	X	X	X	X
local search with reapplication				X	X	X	X	X
diversification					X	X	X	X
diversification with repetition						X		X

Table 1: versions of the implemented GA

4 Computational Results

Each version was run by using the same set of oriented graphs as input and their computational results are described in this section. All versions were implemented in C and run in an Intel Pentium III 800MHz processor.

As instances of the partitioning problem for directed graphs in public libraries were unknown, several classes of directed graphs were artificially created for the evaluation of the proposed algorithms, with different number of nodes: 10, 20, 40, 60, 80, 100, 120, 150, 200 and 500 nodes. Two or more graphs were created for each amount of nodes indicated above, with different formation characteristics.

Due to the presence of random components in an GA, each algorithm has been run three times with each graph, thus reaching the average evaluation in each. Table 2 presents the total number of graphs used for each amount of nodes, as well as the amount of accomplished tests, totalling 66 tests.

number of nodes	**10**	**20**	**40**	**60**	**80**	**100**	**120**	**150**	**200**	**500**
distinct graphs	2	2	2	4	2	2	2	2	2	2
tests in each graph	3	3	3	3	3	3	3	3	3	3
TOTAL	**6**	**6**	**6**	**12**	**6**	**6**	**6**	**6**	**6**	**6**

Table 2: total number of accomplished tests

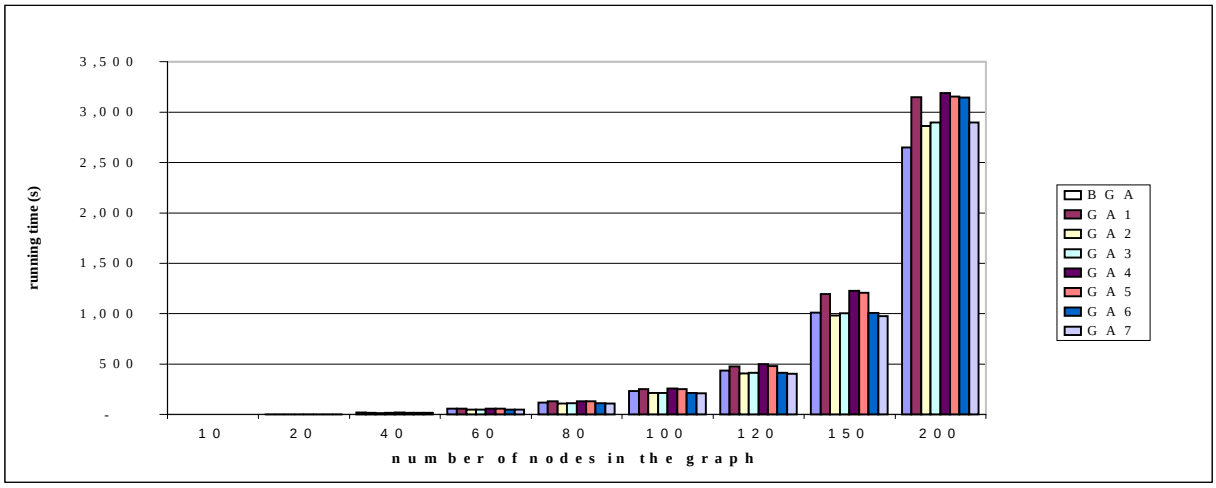

Figure 5: average running time of each version, considering each amount of nodes of the graphs

	10 nodes	20 nodes	40 nodes	60 nodes	80 nodes	100 nodes	120 nodes	150 nodes	200 nodes	500 nodes
BGA	**6**	0	0	0	0	0	0	0	0	0
GA1	**6**	**3**	3	**10**	**3**	2	0	**1**	**3**	0
GA2	**6**	**3**	3	7	2	**4**	**1**	0	1	2
GA3	**6**	0	3	7	2	**4**	**1**	0	2	**3**
GA4	**6**	0	**4**	8	2	2	**1**	0	**3**	0
GA5	**6**	0	3	9	1	2	**1**	**1**	**3**	1
GA6	**6**	0	3	6	2	1	0	0	2	**3**
GA7	**6**	0	3	6	0	0	0	0	1	1

Table 3: number of *best* (*G*) obtained in each version

	10 nodes	20 nodes	40 nodes	60 nodes	80 nodes	100 nodes	120 nodes	150 nodes	200 nodes	500 nodes
BGA	**100.00%**	94.07%	75.22%	71.69%	74.55%	67.73%	69.17%	71.22%	65.59%	61.56%
GA1	**100.00%**	**99.84%**	99.26%	99.74%	**99.14%**	99.00%	98.19%	**99.07%**	**99.36%**	74.92%
GA2	**100.00%**	**99.84%**	99.26%	99.31%	98.06%	99.35%	**99.12%**	98.36%	99.00%	98.58%
GA3	**100.00%**	99.08%	99.26%	99.36%	98.94%	**99.49%**	99.02%	97.17%	98.96%	**98.78%**
GA4	**100.00%**	99.39%	**99.51%**	99.62%	99.05%	98.83%	98.50%	97.51%	**99.36%**	75.07%
GA5	**100.00%**	99.39%	99.26%	**99.86%**	98.41%	98.71%	98.64%	98.28%	99.33%	75.87%
GA6	**100.00%**	99.39%	99.26%	98.60%	98.70%	97.71%	98.51%	97.48%	99.20%	98.41%
GA7	**100.00%**	99.39%	99.26%	99.14%	98.01%	97.44%	97.29%	97.51%	98.78%	98.02%

Table 4: average percentage value from the best solution of all versions for each size of graph

For all the graphs used, except for the graphs with 500 nodes, the versions of GA have executed all the iterations. The total number of iterations for each execution, as exposed in the Section 2, corresponds to $100 \times n$, where n is the number of nodes in the graph. For the graphs with 500 nodes, the execution of each GA was interrupted when completing 4 hours, with an average of 2,200 iterations out of 50,000 iterations. The average execution time for each version, excluding the graphs of 500 nodes, is presented in Figure 5.

In Figure 5 it can be noted that the running time of BGA and the other seven versions proposed here are very similar: the use of additional modules does not necessarily imply a significant increase in the time, since the presence of these modules not only can provide improvement in the quality of the solutions, but also can reduce the space of search of these heuristics. The calibration in the number of clusters significantly reduces the running time of the local search, apart from reducing the total number of conversions of the nodes among clusters.

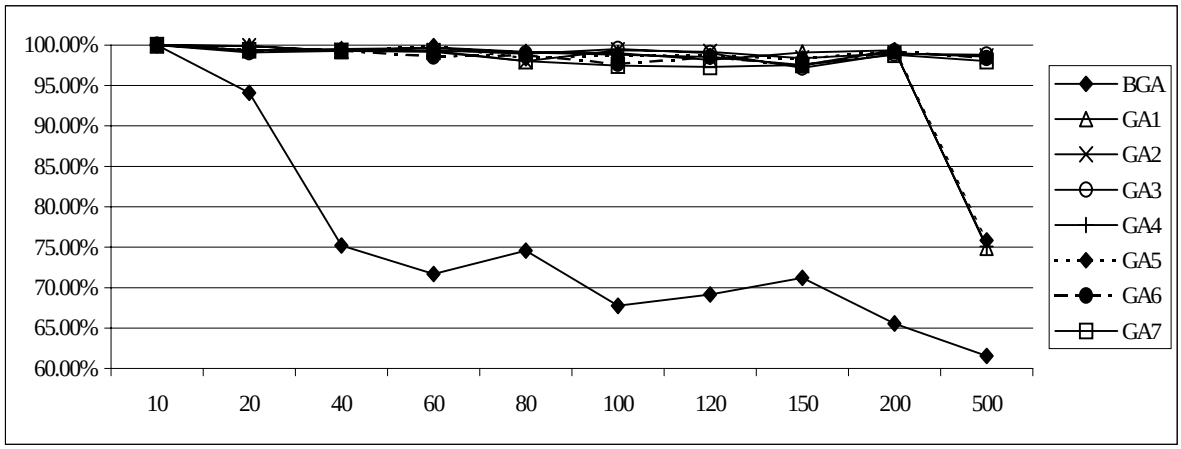

Figure 6: chart regarding Table 4

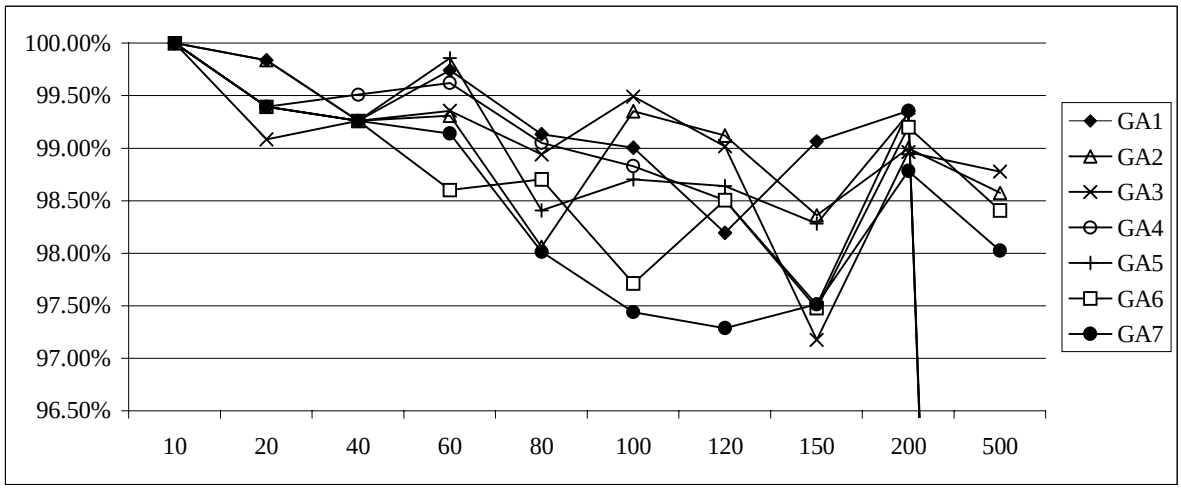

Figure 7: reduction of the scale of the graph in Figure 6, to exhibit details from GA1 to GA7

We will denote as *best*(*G*) the best solution obtained by the execution of all the different versions for each graph *G*. Table 3 indicates the number of times that each version obtained the solution *best*(*G*). Table 4 presents the average percentage value of the *best*(*G*) fitness value for all versions for each size of graph. For example, the value 100% means that the average fitness value of the solution obtained by the respective version using a graph *G* was the same value as that of the *best*(*G*).

It is important to note that even those tests that did not reach the solution *best*(*G*), were close to it almost all the times, as illustrated in Table 4 and Figures 6 and 7.

Based on Figures 6 and 7 and Tables 3 and 4, we can conclude that:

- All the seven versions here proposed, unlike GA in the literature (BGA), produced average fitness value of the final solutions at a maximum of 3% of the fitness value of *best*(*G*) for all the graphs *G* analyzed with up to 200 nodes.
- We observed, however, that some algorithms analyzed here can strongly depend on the type of graph used. This was confirmed through the accomplished tests, in which some versions performed much better than the others for certain graphs used. Therefore, we observed that the versions GA2, GA6 and GA7 were the ones that showed to be more robust with average fitness value at a maximum of 2,6% of the *best*(*G*) values for all the types and sizes of graphs *G* considered (see Table 4).
- Besides this result, the versions GA2, GA3, GA6 and GA7 produced average solutions at a maximum of 2% of the *best*(*G*) for all the graphs *G* with 200 and 500 nodes. For problems of higher loaded graph partitioning we believe that GA6 and GA7 tend to be the best candidates, with a more efficient search, because of their behavior in the accoplished tests.

5 Conclusions

In this work the main objective was to analyse the behavior of different versions of evolutionary algorithms based on populations of solutions. In order to achieve this, different versions of genetic algorithms were developed,

from a basic one to versions including control mechanisms in the clusters number, local search modules and mechanisms of diversification for a population of solutions.

The computational tests showed that there is an *evolution of the algorithms* in so far as these proposals are included in the BGA. Another important result was to verify that the versions with the combination of the proposals procedures had good performance in instances of great load (over 200 nodes).

As regards suggestions for future work, we are beginning to carry out research whose objective is to develop cooperative and adaptative evolutionary algorithms and whose main characteristic is to take advantage of the conclusions of this work so as to obtain more consistent algorithms with stable average performance in all types of input graphs. Another work to be initiated will be the accomplishment of parallel versions of the presented algorithms, not only to reduce the computational time demanded by the sequencial versions, but also to take advantage of the communication among processors in order to attempt to improve the quality of the generated solutions.

Bibliography

Berkhin, P. 2002. *Survey of Clustering Data Mining Techniques*. Accrue Software.

Chiun, Y. and Lan, L.W. 2001. "Genetic Clustering Algorithms". *European Journal Of Operational Research* (135) 2, pp. 413-427.

Cole, R.M. 1998. "Clustering with Genetic Algorithms". *Master's thesis*, Dept. of Computer Science, Univ. of Western Australia.

Doval, D., Mancoridis, S. and Mitchell, B.S. 1999. "Automatic Clustering of Software Systems using a Genetic Algorithm". In *Proceedings of the 1999 Int. Conf. on Software Tools and Engineering Practice* (STEP '99).

Drummond, L.M.A., Ochi, L.S. and Figueiredo, R.M.V. 1997. "Design and Implementation of a Parallel Genetic Algorithm for the Travelling Purchaser Problem". In *Applied Computing/ACM*, pp. 257-263.

Drummond, L.M.A., Vianna, D.S. and Ochi, L.S. 1998. "Genetic Algorithm for the Vehicle Routing Problem". In *Future Generations on Computer Systems*, Elsevier, vol. 14(5-6), pp. 285-292.

Drummond, L.M.A., Vianna, D.S. and Ochi, L.S. 1998. "An evolutionary hybrid metaheuristic for solving the vehicle routing problem with heterogeneous fleet". In *Lecture Notes in Computer Science*, Springer Verlag, vol. 1391, pp. 187-195.

Glover, F. 1997. "Heuristics for Integer Programming Using Surrogate Constraints". *Decision Sciences*, vol. 8, no. 7, pp. 156-166.

Glover, F. and Laguna, M. 2000. "Fundamentals of Scatter Search and Path Relinking". *Control and Cybernetics*, vol. 29, no. 3, pp. 653-684.

Hartuv, E., Schmitt, A., Lang, J. et al. 1999. "An Algorithm for Clustering for Gene Expression Analysis". In *Proceedings of Third Annual International Conference on Computational Molecular Biology* (RECOMB '99).

Hartuv, E. and Shamir, R. 1999. "A Clustering Algorithm based on Graph Connectivity". *Technical Report*, Tel Aviv University, Dept. of Computer Science.

Laguna, M. 2002. "Scatter Search". In *Handbook of Applied Optimization*, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford University Press, pp. 183-193.

Lorena, L.A.N. and Furtado, J.C. 2001. "Constructive Genetic Algorithm for Clustering Problems". *Evolutionary Computation*, vol. 9, no. 3, pp. 309-327.

Maini, H.S., Mehrotra, K.G., Mohan, C.K. and Ranka, S. 1994. "Genetic Algorithms for Graph Partitioning and Incremental Graph Partitioning". In *Proceedings of the 1994 Conference on Supercomputing*, pp. 449-457.

Martí, R., Laguna, M. and Campos, V. 2002. "Scatter Search vs. Genetic Algorithms". *Technical Report*, University of Colorado at Boulder.

Moscato, P. 1989. "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms". *Technical Report*, Caltech Concurrent Computation Program, California Institute of Technology.

Ochi, L.S. and Rocha, M.L. 2000. "A new hybrid evolutionary algorithm for the vehicle routing and scheduling problems". In *Proceedings of the Ninth International Conference on Intelligence Systems: Artificial Intelligence Applications for the New Millennium*, pp. 135-140.

Ochi, L.S., Vianna, D.S. and Drummond, L.M.A. 2001. "An asynchronous parallel metaheuristic for the period vehicle routing problem". In *Future Generations Computer Systems*, Elsevier, vol. 17, pp. 379-386.

Radcliffe, N.J. and Surry, P.D. 1994. "Formal Memetic Algorithms". In T. Fogarty, editor, Evolutionary Computing AISB Workshop, volume 865 of Lecture Notes in Computer Science, pp. 1-16, Springer-Verlag.