1991

Intelligent Structural Operators for the k-way Graph Partitioning Problem

Gregor von Laszewski
Syracuse University, Northeast Parallel Architectures Center; Gesellschaft fur Mathematik und Datenverarbeitung mbH, Schloss Birlinghoven, gregor@npac.syr.edu

Intelligent Structural Operators for the k-way Graph Partitioning Problem

Gregor von Laszewski
Gesellschaft für Mathematik und Datenverarbeitung mbH
Schloß Birlinghoven, D – 5205 St. Augustin
gregor@npac.syr.edu
NorthEast Parallel Architectures Center
Syracuse University
111 College Place
Syracuse, NY 13244-4100

Contents

Intelligent Structural Operators for the k-way Graph Partitioning Problem

Gregor von Laszewski
Gesellschaft für Mathematik und Datenverarbeitung mbH
Schloß Birlinghoven, D – 5205 St. Augustin
gregor@npac.syr.edu
NorthEast Parallel Architectures Center
Syracuse University
111 College Place
Syracuse, NY 13244-4100

Abstract

A parallel genetic algorithm for the graph partitioning problem is presented, which combines general heuristic algorithms with techniques that are described in evolution theory. In the parallel genetic algorithm the selection of a mate is restricted to a *local neighborhood*. In addition, the parallel genetic algorithm executes an *adaptation* step after an individual is generated, with the genetic operators crossover and mutation. During the adaptation step the solution is improved by a common algorithm. Another selection step decides if the adapted descendant should *replace* the parent individual. Instead of using a uniform crossover operator a more *intelligent* crossover operator, which copies subsets of nodes, is used. Basic parameters of the parallel genetic algorithm are determined for different graphs. The algorithm found for a large sample instance a new unknown solution.

1 GENETIC ALGORITHMS

Genetic Algorithms are stochastic search algorithms introduced by J.Holland in the 70's [8]. These algorithms are based on ideas and techniques from genetic and evolutionary theory. Genetic algorithms simulate an evolutionary process with N individuals which represent points in a search space. Every individual is encoded as a string called a *genotype*. The value of the cost function which is defined for such a string is called a *phenotype*.

In each step of the genetic algorithm, called a generation, every individual is evaluated with regard to the entire population. This value is called the *relative fitness* of an individual. According to "natural evolution" offspring are produced using genetic operators. The *selection* operator chooses individuals with a probability that corresponds to the relative fitness. Two chosen individuals produce a descendant using the genetic operator *crossover*. The crossover operator exchanges substrings of the codes of the two chosen individuals. The descendant replaces an individual in the population after the generation step is complete. Another genetic operator, called *mutation*, changes the genotype of the descendant, with a small probability. Mutation and crossover cause variation in the search process. The mutation operator allows a search close to a point in the search space, because only a small number of changes occur. Crossover causes longer jumps in the search space.

However, only selection leads the search in a specific direction. Substrings of individuals that are more fit than others are kept for the next generation. The search is successful if the search space has the property that a combination of two high valued points of the search space leads to a higher valued point with high probability [12].

Further information about genetic algorithms and their applications is provided in [3] and [5].

2 PARALLEL GENETIC ALGORITHMS

In Holland's genetic algorithm, selection occurs in the entire population, whereas in the parallel genetic algorithms the selection of a mate is restricted to a *local neighborhood*. In addition, the parallel genetic algo-

rithm executes an *adaptation* step after an individual is generated, with the genetic operators crossover and mutation. During the adaptation step the solution is improved by a common algorithm. Another selection step decides if the adapted descendant should *replace* the parent individual.

The parallel genetic algorithm (PGA) can be described as follows: An environment consists of a set of locations $X = \{x_1, ..., x_N\}$ which are divided geographically. Connections between locations are described by a relation R on X. At each location x_k there exists an individual I_k^t at time t. At the beginning of the evolutionary process the initial individuals I_k^0 are randomly initialized. For each individual I_k^t, a set of neighbor individuals $\mathcal{N}^t(x_k)$ are determined by the relation R. Figure 1 shows the evolution process that runs on each location.

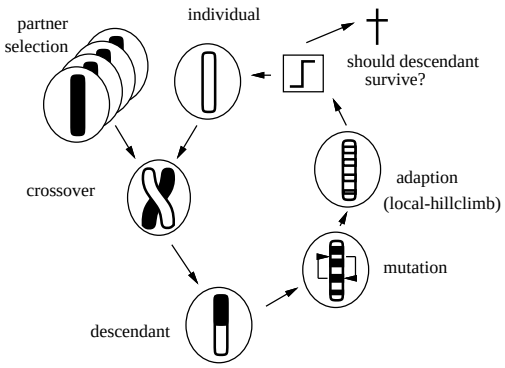

Figure 1: Evolution process

PGA $= (C, N, I^0, K, c, GO)$

C	is the set of genetic codings for the solutions.
N	is the number of locations. The locations are $X = \{x_1, ..., x_N\}$. At each time t there is a individual $I_k^t \in C$ on location x_k.
I^0	$= \{I_1^0, ..., I_N^0\}$ is the initial population at time $t = 0$.
K	$\subset X \times X$ is the communication relation.
c	is a cost function which determines the phenotype of the individual. A coding is evaluated.
GO	= {mutation, crossing-over, selection, parent replacement strategy} is the set of genetic operators.

Figure 2: Parameters of the parallel genetic algorithm

First, an individual chooses a partner for mating in its neighborhood and creates a descendant using the crossover operator. After the mutation operator is applied, the descendant is improved in the adaptation step. If the descendant is well adapted to the environment,[1] it replaces the parent individual. The algorithm is terminated when a termination constraint is fulfilled.[2]

Since the evolution process runs simultaneously on each location, this model can be mapped onto a multiprocessor system. Each processor must know the codings of the individuals living on its neighbor processors. The parallel genetic algorithm has been successfully applied to the traveling salesman problem [14, 6]. In this paper it is demonstrated that the parallel genetic algorithm can also be applied to the complex k-way graph partitioning problem. A formal description of the k-way graph partitioning problem is given in the next section.

3 THE k-WAY GRAPH PARTITIONING PROBLEM

The k-way graph partitioning problem (k-GPP) is a fundamental combinatorial problem which has applications in many areas of computer science (e.g., design of electrical circuits, mapping) [10]. Mathematically we can formulate the k-way graph partitioning problem as follows:

Let $G = (V, E, w)$ be an undirected graph, where $V = \{v_1, v_2, ..., v_n\}$ is the set of nodes, $E \subseteq V \times V$ is the set of edges and $w : E \mapsto I\!N$ defines the weights of the edges. The k-way graph partitioning problem is to divide the graph into k disjoint subsets of nodes $P_1 ... P_k$, such that the sum of the weights of edges between nodes in different subsets is minimal, and the

[1] For example, if the descendant is better than the parents, or if it is better than the worst individual in the neighborhood, it is considered as well adapted.

[2] For example, a time limit may be used as termination constraint.

sizes of the subsets are nearly equal. The subsets are called *partitions*, and the set of edges between the partitions is called a *cut*.

Let $P = \{P_1, ..., P_k\}$ be the partitions. Then the string $(g_1 g_2 ... g_n)$ describes a solution:

$$g_i = a \iff v_i \in P_a \quad \forall i \in \{1, ..., n\}$$

With $a \in \{1, ..., k\}$. Node v_i is assigned to the partition specified by g_i. Instead of minimizing the cost of the cut we maximize the sum of the weights of all the edges between nodes in the same partitions. This is an equivalent problem because the total cost of edges is constant. This leads to a cost function of:

$$c_(g_1 g_2 ... g_n) = \sum_{\substack{1 \le i < j \le n \\ g_i = g_j}} w(v_i, v_j) \quad .$$

The advantage of this cost function is that a selection operator for a genetic algorithm can be easily formulated. Furthermore, the parallel genetic algorithm described in this paper does not change the sizes of the partitions during the computation. The equal size of the partitions is controlled by the variance

$$\sigma^2(P) \stackrel{\text{def}}{=} \frac{1}{m} \sum_{i=1}^{m} |P_i|^2 - \left(\frac{1}{m} \sum_{i=1}^{m} |P_i| \right)^2 .$$

4 PARALLEL GENETIC ALGORITHM APPLIED TO THE k-GPP

To apply the parallel genetic algorithm to the k-way graph partitioning problem, a representation of problem solutions has to be defined. Genetic operators which control the composition of two solutions or the modification of one solution have also to be defined. In addition, the values of the parameters used by the parallel genetic algorithm have to be determined (e.g. population size, relation between the locations, mutations, etc.).

4.1 Representation, Communication Relation, and Selection

Rather than a simple binary representation, the discrete string representation defined in section 3 is used to code solutions of the k-way graph partitioning problem. Therefore, a larger alphabet $\Sigma = \{1, .., k\}$ is used. To guarantee the constraint of the equal partition size, only a subset of all k^n possible strings is allowed. This straightforward representation implies that the phenotype of a string $g_1 ... g_n$ is given by the value $c(g_1 ... g_n)$.

For the experiments, the communication relation between the locations is determined by a ring:

$$x_k \text{ is neighbor of } x_l \iff 0 < (l - k + N) \; mod \; N \le A,$$

where A denotes the number of neighbors and N denotes the population size. Let $\mathcal{N}^t(x_k)$ be the set of individuals located in the neighborhood of the individual I_k^t. These individuals are called the *selection neighbors*. For example, let three be the size of the the selection neighborhood. Then those three individuals are in the selection neighborhood that lie in the ring directly before the individual itself (Figure 5).

The individual which is currently the best[3] can be added to the selection neighbors. This individual is called the *currently best* individual.

With the selection neighbors, the relative fitness of an individual in an environment is defined as follows:

$$f(I_l^t) = \frac{c(I_l^t)}{\sum\limits_{I_r^t \in \mathcal{N}^t(x_k)} c(I_r^t)} \qquad \forall I_l^t \in \mathcal{N}^t(x_k)$$

The relative fitness determines the probability of selecting an individual from the *selection neighbors* for mating. With the help of this fitness function, very good solutions can be found. Other selection strategies are described elsewhere [2, 5].

[3] The currently best individual is the representation of the best solution found since the parallel genetic algorithm is started.

4.2 The Structural Crossover Operator

The crossover operator is very important for the success of the genetic algorithm. If a crossover operator destroys too much information already gained in the past, the genetic algorithm degenerates to a simple random search algorithm. To avoid losing too much information, an intelligent structural crossover operator is defined. It copies whole partitions from one solution into another.

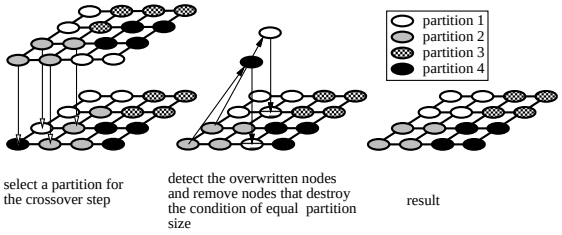

Figure 3: Recombination of two solutions

Figure 3 depicts the recombination of two solutions. A grid with 4×4 nodes is to be divided into 4 partitions. To show the recombination step more clearly, colors are used in the figure instead of numbers to represent the different partitions.

First, a partition is randomly chosen in a parent solution (the light gray partition). Then this partition is copied into the other parent solution. Because this copying process may destroy the constraint of equal partition sizes, a repairing operator is applied. In the repairing step, all nodes in the temporary solution which are not elements of the copied partition, but have the same color as this partition, are detected. These nodes are marked in the second part of Figure 3 with horizontal lines.

To assign these nodes to a partition, they have to be marked (e.g. randomly) with the colors of those nodes which have been overwritten by the copied partition. In the example the white and the black partitions have one node too few. So the nodes marked with horizontal lines are relabeled with the colors white and black. A new code is generated which represents a valid solution for the problem instance.

Executing the crossover operator on arbitrary genotypes creates descendants which temporarily have a lot of open positions during the crossover process. In the extreme case, these positions could correspond to a whole partition. If the number of nodes in a partition is large in comparison to the number of nodes in the graph, a great disturbance of the old solutions will arise. In order to avoid losing too much information computed in the past, the codings are adapted before the crossover process starts. They are changed, in such a way that the difference between the two parent solutions is as small as possible. Let $(a_1 ... a_n), (b_1 ... b_n)$ denote the parent individuals. Then the difference of the two parent individuals is defined as follows:

$$\text{difference } (a_1 ... a_n, b_1 ... b_n) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \begin{cases} 1 & \text{if } a_i \neq b_i \\ 0 & \text{otherwise} \end{cases}$$

4.3 The Structural Mutation Operator

A common mutation operator that replaces values in the string with an element randomly chosen out of Σ will destroy the condition of equal partition size. To avoid leaving the search space, a mutation is defined as the exchange of two numbers of the coding.

Because, at the beginning of the evolution process, the solutions generated with the crossover operator are very different from each other, there is no need to disturb them with a mutation operator. Mutations are only executed if the difference between a parent and the solution created by the crossover operator is smaller than a parameter called *mutations*. Let Δ denote the minimum of the difference between the two parents and its descendent generated by the crossover step. If this difference is smaller than the parameter *mutations*, then Δ - *mutations* swap operations are executed on the coding of the descendant.

4.4 The Adaptation Step

For large problem instances, it is important to restrict the solution space. This can be achieved by using a

hill climbing algorithm to improve the solutions represented by the coding. Therefore, a variant of the 2-opt algorithm introduced by Kernighan and Lin is implemented [10]. For all pairs of nodes, the 2-opt algorithm exchanges these nodes if the solution can be improved by the exchange. This step is repeated until no further improvement can be made. Since one iteration step is done in $O(n^2)$ time, it is necessary to reduce the number of nodes on which this heuristic is used. Instead of trying the exchange over all pairs of nodes, the 2-opt algorithm is only executed on the nodes located at the border of the partitions.

5 RESULTS

The parallel genetic algorithm is implemented on a 64 node transputer system. Each evolution process is executed on one transputer. The maximal population size is 64.

This paper concentrates on two different problem instances. First, a graph whose edges are connected like a *grid* is used to demonstrate some basic effects of the parallel genetic algorithm. This graph is to be divided into four partitions. Therefore, the globally optimal solution for a grid with 100 nodes has a cost function value of 20. Without equivalent solutions[4], there exists only one globally optimal solution. The problem *grid* provides a test instance for determining the basic properties of the implemented algorithm.

Second, a graph called *beam* is used [4]. This graph has 918 nodes and 3233 edges, and is to be divided into 18 partitions.

There are only a few algorithms which can be compared with the PGA, because other algorithms are usually restricted to the 2-way graph partitioning problem. Two of these comparable algorithms are the round robin algorithm of Moore and the divide-and-conquer Kernighan-Lin algorithm of Zmijewski[13, 7]. These algorithms do not use the constraint of equal partition size, so that the partitioning problem is simpler.

For the grid graph, parameters were found that allow the globally optimal solution to be generated in every case. Also, the PGA found the best known solution for the instance *beam*. Figure 4 shows the progress of this solution. The table 1 also shows the best known results found with the different algorithms.

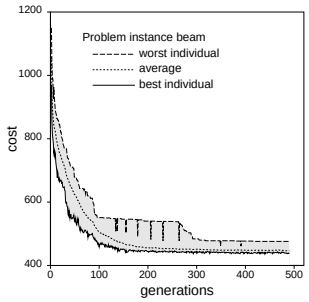

Figure 4: Problem beam, 64 individuals

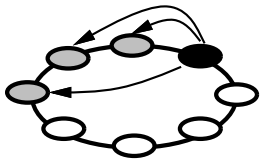

Figure 5: Communication structure, Ring with 8 individuals

algorithm	minimal cost	$\sigma(P)$	running time
GZ87	587	0.99	78 rounds
Moore	453	0.99	78 rounds
PGA	430	0.00	500 generations, 28 min

Table 1: Comparison of the best solutions for the instance beam

6 PARENT REPLACEMENT STRATEGY

In the implementation of the parallel genetic algorithm for the k-way graph partitioning problem the convergence speed, is an important factor. To increase the convergence speed two special concepts are introduced:

1. The parent solution is only replaced if a specific condition is fulfilled.
2. The currently best individual is included in the selection neighborhood.

[4]Solutions are equivalent to each other only if they are different in the names of the partitions

In this section, different strategies for deciding whether a descendant should survive and replace a parent individual are compared. The following replacement strategies are considered:

Strategy "each": Each parent individual is replaced by its descendant. The replacement is done regardless of the quality of the parent or the descendant solution.

Strategy "better": A parent individual is only replaced by its descendant if the descendant is better than the parent individual.

Strategy "locally better": This strategy is a combination of the previous strategies. The replacement of a parent individual is dependent on the cost of the neighbor individuals, the descendant, and the parent itself. A parent is replaced if the descendant is better than the parent solution, or if the descendant is better than the worst individual in the *local* selection neighborhood.

The same experiments are done with and without the currently best individual in the selection neighborhood. The problem instance *grid* is used. For the experiments with the problem instance grid, the population size is 16 and the size of the selection neighborhood is 4. Figure 6 and 7 display the range of the cost values of the population over generations. For each generation, the cost of the worst and best individuals are shown. Furthermore, the average cost of the individuals in the population is shown. The graphs shown in Figure 6 do not include the currently best individual in the selection neighborhood. Whereas, the graphs shown in figure 7 include the currently best individual in the selection neighborhood.

The experiments show that for the strategy "each," the cost range of the individuals in the population fluctuate heavily among the generations. A relative long time period is needed to find the minimal solution. If the currently best individual is included in the selection neighborhood, the convergence speed can be improved.

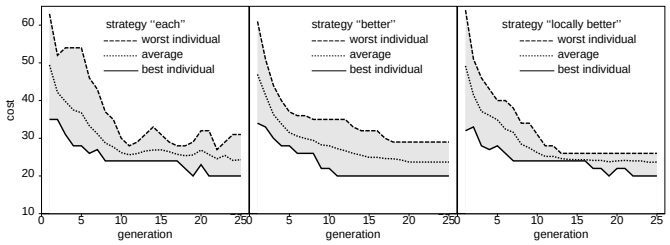

Figure 6: Replacement strategy for a parent without the currently best individual in the selection neighborhood

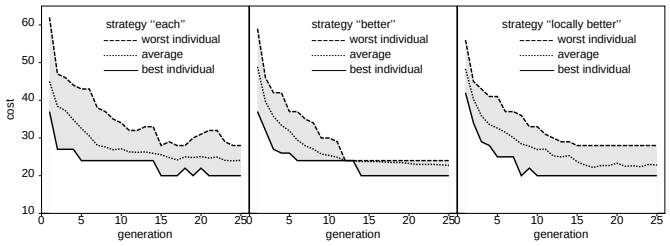

Figure 7: Replacement strategy for a parent with the currently best individual in the selection neighborhood

Using the strategy "better," the convergence speed is greater than with the strategy "each". If the currently best individual is also included in the selection neighborhood, then the algorithm often gets stuck in a locally optimal solution. The algorithm converges too quickly without discovering other possible solutions in the solution space. The second graph in Figure 7 shows that at a specific time period, all individuals have the same cost value. No new information is gained by the genetic algorithm.

Now the question arises, how convergence in low quality solutions can be avoided. One simple way to overcome this problem is to combine the two strategies with each other as described above. The replacement strategy "locally better", with the inclusion of the currently best individual in the selection neighborhood, enables the parallel genetic algorithm to find high quality solutions with a high convergence speed.

Another important factor for jumping out of stable but suboptimal solutions is a perturbation of the solution

with the mutation operator. Using a sufficiently large mutation rate enables the parallel genetic algorithm to introduce new variation into the search process as shown in the second graph of Figure 7. Here, the mutation rate is $\frac{1}{5}$.

This result also holds for the larger problem instance *beam*. Figure 8 shows the range of the solutions generated with the strategy "locally better" and the inclusion of the currently best individual in the selection neighborhood. This approach produces the best results.

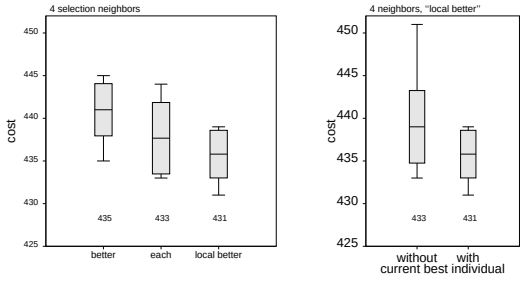

Figure 8: Replacement strategy. The problem instance *beam* is used.

7 MUTATION

In the last section, it was shown that the mutation operator is important for varying solutions when the genetic algorithm get stuck in locally optimal solutions. With a large experiment, the optimal number of mutations are determined for the problem instance *grid*. The optimal number of mutations achieves

1. that the average number of generations needed to find a very good solution[5] is minimal.

2. that the frequency of finding the globally optimal solution is maximal.

Figures 9 and 10 depict the result of the experiments used to find the optimal number of *mutations*. 100 experiments were done for each mutation in the interval from 0 to 50. Each experiment was terminated after the globally optimal solution is found, or the evolution cycle (Figure 1) is repeated more than 100 times. The strategy "locally better" is used for replacing the parent individual.

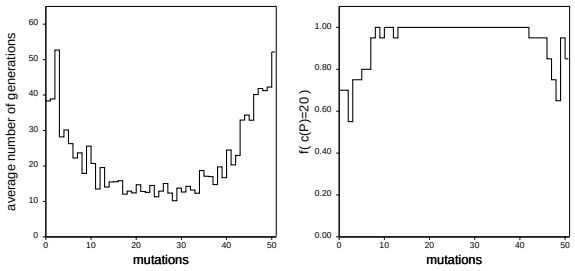

Figure 9: Analysis of the number of mutations. The currently best individual is not included in the selection neighborhood

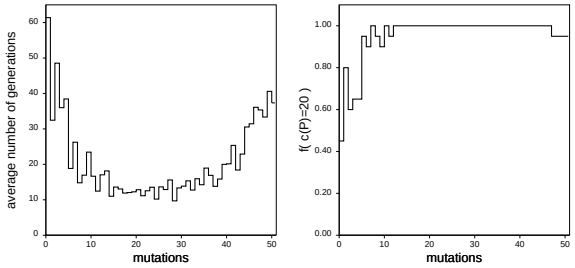

Figure 10: Analysis of the number of mutations. The currently best individual is included in the selection neighborhood

If the currently best individual is not included in the selection neighborhood, the optimal number of mutations is between 17 and 34. If fewer mutations are done, the algorithm often get stuck in suboptimal solutions. This can be avoided by increasing the number of mutations. Finding the globally optimal solution is prevented if too many mutations are executed. In this case the algorithm degenerates to a simple random search algorithm.

If the currently best individual is included in the selection neighborhood, the same effects appear. However, the average number of generations which are needed to find the globally optimal solution is smaller.

[5] For the problem instance *grid*, "very good" means "globally optimal"

7.1 The Correlation Between Mutation and the Adaptation Step

Common genetic algorithms use very small mutation rates. For the problem instance *grid* an optimal solution rate of about $\frac{1}{4}$ was observed. This section explains, why a high mutation rate is needed to find very good solutions quickly. To see the correlation between the mutation rate and the adaptation step, one has to remember that:

1. the adaptation step is executed after the recombination of a new descendant.
2. the mutation operator is applied when the crossover operator generates a descendant that is very similar to one of its parents.

The PGA is applied to the problem instance *grid*. At the end of the evolution process, a descendant is only slightly different from one of its parents. If the 2-opt algorithm were executed next, no new variation would introduced into the search process.
Furthermore, with advancing generations nearly optimal solutions are generated. Applying the 2-opt algorithm on a slightly disturbed solution near the optimum leads with high probability to the same old solution. This fact is displayed in Figure 11. Let a be a solution and b be the solution which is created by applying some mutations and the 2-opt algorithm on a. Figure 11 shows how often the solutions a and b are equivalent for different good solutions.
To prevent the parallel genetic algorithm from getting stuck, the mutation operator is used to disturb the descendants. In addition the number of mutations has to be sufficiently large.
The mutation operator is defined in such a way that it is only applied if the difference between the parent solutions and the descendant is smaller than the number of mutations. Because the solutions are so different at the beginning of the search process the mutation operator is only applied later. Therefore, the crossover and repairing operator are responsible for introducing variety early on in the search process.

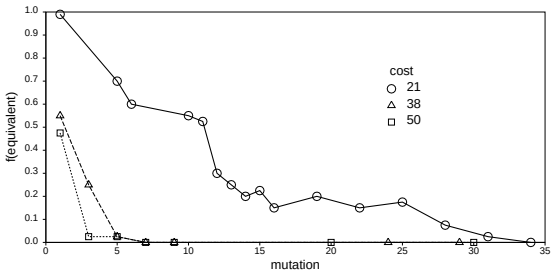

Figure 11: Relative frequency of generating an equivalent 2-optimal solution after applying a number of mutations and the 2-optimal algorithm for different good solutions

Another factor for a large mutation rate is that also mutations are done regardless of the position of the nodes in a partition. If the nodes are in the same partition then the mutation operator generates obviously not a new solution. This can in future implementations avoided if only mutations between different partitions are allowed. Furthermore, the nodes which are allowed to mutate could be restricted to the borders of the partitions.

7.2 Population Size and Neighborhood Size

Using the problem instance *grid*, only slight differences occur if other parameters like the population size are modified. Therefore, the problem instance *beam* is focus of this section.

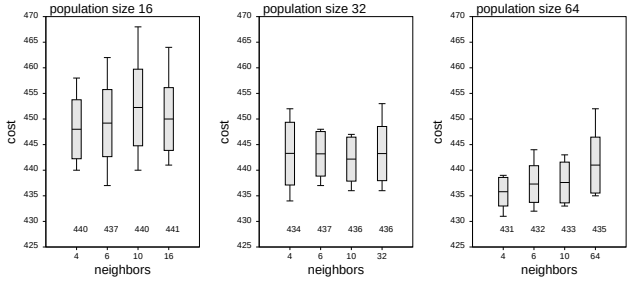

Figure 12: Analysis of the population size and the size of the selection neighborhood

Figure 12 compares the cost of solutions found with

different population and neighborhood sizes. The selection neighborhood of each individual includes the currently best individual to increase the convergence speed of the algorithm. To get good results, the population size is more important than the size of the selection neighborhood. The best results are found using the largest population – i.e. 64 individuals. In addition, the size of the neighborhood should be small. One important result is that, for large populations, the PGA algorithm produces better solutions with a restricted neighborhood than with a panmictic population.

Crossover and Mutation

In [1], a genetic algorithm for the graph partitioning problem can be found. This algorithm only generates solutions for the 2-way partitioning problem. Experimental results are only presented for graphs of up to 64 nodes. A uniform crossover operator is used to produce offspring. Each position of the offspring is randomly labeled by one of the two corresponding numbers in the parent genotypes.

In this paper the uniform crossover operator has been extended for the k-way graph partitioning problem. With this crossover operator, however, no solutions were found which were as good as those generated with subset crossover for different mutations (Figure 13).

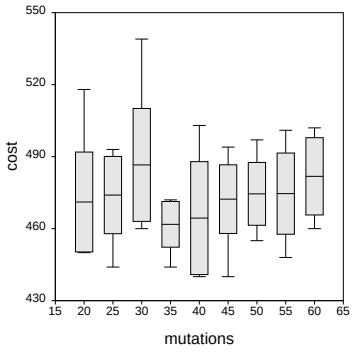

Figure 13: Uniform crossover operator with different mutations

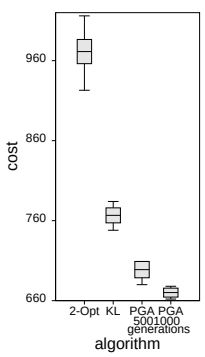

Figure 14: Comparison of the cost of the cut for different heuristics on a random graph with 900 nodes

Random Graphs

Random graphs are defined so that the average degree of each node is $(n-1)p$, where n is the number of nodes, and p is the probability that a pair of nodes is connected by an edge. For a constant p and a large n, random graphs are dense. The graph partitioning problem is easier to solve for dense graphs, because the solutions have nearly the same cost of the cut. The GPP is more difficult for instances of restricted random graphs whose degree is bounded, e.g. by 4 [9]. To compare the PGA with other heuristics, the 2-opt and the KL-algorithm for the m-partitioning were also implemented. The KL-algorithm tries to exchange sequences of nodes instead of exchanging only two nodes in one step. A detailed description of these algorithms is provided in [10] and [11].

Figure 14 shows the comparison of the algorithms 2-opt, KL, and PGA terminated after 500 and 1000 generations. They are tested on a random graph with 900 nodes and maximum node degree of 4. Experiments with the problem instances with 900 and 918 nodes show that the PGA is much faster for regular graphs than for restricted random graphs.

8 CONCLUSIONS

The parallel genetic algorithm computes very good results for the graph partitioning problem. For a large problem instance the algorithm found a new unknown minimal solution. The search space has the property that a combination of two high valued points often leads to a higher valued point. Implanting a small, maximized subset of nodes from one solution into another and applying a local hill-climbing heuristic to this solution, often leads to a better partition.

Furthermore, this paper shows:

- that a parent replacement strategy improves the quality of the solutions.
- that mutation is needed only if the crossover operator produces a solution which is nearly equal

to one of its parents.

- that the population should be chosen to be as large as possible.
- that better solutions are generated with the restricted neighborhood structure than with the panmictic population structure.
- that for the implementation presented in this paper, the selection neighborhood should have a size of 4, and should include the currently best individual to achieve the best results with a high convergence rate.
- that to restrict the solution space, a discrete problem representation and structural genetic operators are important.
- that the adaptation step is very important for restricting the solution space and improving the convergence rate of the algorithm.

This implementation of a parallel genetic algorithm shows that there exist two strategies for defining genetic algorithms. The first strategy uses a sophisticated representation and simple genetic operators onto the codings to generate good solutions. Sometimes it is difficult to find a sophisticated representation. Then it is easier to chose a simple straightforward representation and introduce intelligence into the algorithm by defining genetic operators which use the structure of the problem to generate offspring.
There are many opportunities for further research in this area. The most interesting would be to choose a larger population size to display more clearly the difference between the panmictic population and the neighborhood model. A more sophisticated mutation and selection operator may also be defined. The implementation environment makes it possible to run different adaptation strategies on different locations in order to inspect the solution space with different strategies. Also, different communication relations may be compared.

Acknowledgment

The author thanks Heinz Mühlenbein and Martina Gorges-Schleuter for providing the initial stimulation for the study and helpful discussions along the way, and Jim Buuck and Steve Edwards for suggestions that helped improving the final presentation.

References

[1] D. H. Ackley. A Connectionist Machine for Genetic Hill-climbing. The Kluwer international series in engineering and computer science SECS 28. Kluwer Academic Publisher, Norwell, Massachussets, 87.

[2] J.E. Baker. Adaptive Selection Methods for Genetic Algorithms. In 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 85.

[3] L. Davis. Genetic Algorithms and Simulated Annealing. Morgan Kaufmann, Los Altos, 87.

[4] G.C. Everstine. A comparison of three resequencing algorithms for the reduction of matrix profile and wavefront. Int. J. Numer. Methods in Eng., Vol. 14, 837-853, 79.

[5] D. E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 89.

[6] M. Gorges-Schleuter. ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy. In 3rd Int. Conf. on Genetic Algorithms, San Mateo, Morgan Kaufmann, 89.

[7] J. R. Gilbert and E. Zmijewski. A Parallel Graph Partitioning Algorithm for a Message-Passing Multiprocessor. Techn. Report 87-803, Cornell University, 87.

[8] J. H. Holland. Adaptation in natural and artificial systems. Ann Arbor, University of Michigan Press, 75.

[9] D.S. Johnson, C.R. Aragon, L.A.McGeoch, C.Schevon. Optimization by Simulated Annealing: An Experimental Evaluation. Part I, Graph

Partitioning. Operations Research, 37(6):865-892, Dec 89.

[10] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. Technical report, Bell Syst. Techn. J., February 70.

[11] G. von Laszewski. Ein paralleler genetischer Algorithmus für das GPP. Master's thesis, Univerisität Bonn, 90.

[12] G. von Laszewski, H. Mühlenbein. Partitioning a Graph with a Parallel Genetic Algorithm. First Int. Workshop on Parallel Problem Solving from Nature, University Dortmund, West Germany, 90.

[13] D. Moore. A Round-Robin Parallel Partitioning Algorithm. Technical Report 88-916, Cornell University, Ithaca, NY, 88.

[14] H. Mühlenbein. Parallel Genetic Algorithm, Population Dynamics and Combinatorial Optimization. In 3rd Int. Conf. on Genetic Algorithms, San Mateo, Morgan Kaufmann, 89.