# Genetic Algorithms as Multi-Coordinators in Large-Scale Optimization *

Ioannis T. Christou† Wayne Martin† Robert R. Meyer†

**Abstract**

We present high-level, decomposition-based algorithms for large-scale block-angular optimization problems containing integer variables, and demonstrate their effectiveness in the solution of large-scale graph partitioning problems. These algorithms combine the subproblem-coordination paradigm (and lower bounds) of price-directive decomposition methods with knapsack and genetic approaches to the utilization of "building blocks" of partial solutions. Even for graph partitioning problems requiring billions of variables in a standard 0-1 formulation, this approach produces high-quality solutions (as measured by deviations from an easily computed lower bound), and substantially outperforms widely-used graph partitioning techniques based on heuristics and spectral methods.

## 1 Introduction

Most large-scale mathematical programming problems are constructed by concatenating and linking together blocks of data corresponding to major components of the problem. In the case of network optimization, for example, these components may correspond to different commodities or goods with different priorities or origins. There may also be multiple time periods or scenarios, and even combinations of all of the preceding factors. In many important cases it is thus possible to formulate such problems as block-angular problems of the following form:

$$
\begin{array}{lllllllll}
\min\limits_{x} & f_1(x_1) & + & f_2(x_2) & + & \dots & + & f_P(x_P) & \\
s.t. & A_1x_1 & & & & & & & = b_1 \\
 & & & A_2x_2 & & & & & = b_2 \\
 & & & & & \ddots & & & \\
 & & & & & & & A_Px_P & = b_P \\
 & D_1x_1 & + & D_2x_2 & + & \dots & + & D_Px_P & \leq d \\
 & & & \multicolumn{5}{l}{0 \leq x_p \leq u_p, \;\; p = 1, \dots, P} &
\end{array}
\tag{1}
$$

In addition, we assume that some or all of the variables are constrained to be integer. Note that $x_p$ represents a block of variables and the matrices $A_p$ and $D_p$, $p = 1, \dots, P$ correspond to blocks of constraints. For large-scale problems, the number of variables in each block may be in the thousands and there may be thousands of constraints. (In the largest problem that we have solved by our approach, $P = 100,000$ and each $x_p$ is a block of $100,000,000$ variables.)

*This research was partially supported by the Air Force Office of Scientific Research under grant F49620-94-1-0036, and by the NSF under grants CDA-9024618 and CCR-9306807.

†Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706.

The constraints $\sum D_p x_p \leq d$ are called *coupling constraints* since the deletion of these constraints removes the coupling between the blocks of variables $x_p$. Thus, the resulting "relaxed" problem may be solved by decomposing it into $P$ smaller optimization problems (called subproblems) of the form:

$$\begin{array}{lll} \min\limits_{x} & f_p(x_p) & \\ s.t. & A_p x_p = & b_p \\ & 0 \leq x_p \leq u_p & \end{array} \tag{2}$$

Because of the relationship between their feasible sets, the optimal value of the full relaxed problem (which is the sum of the optimal values of the subproblems) provides a *lower bound* for the objective of the original problem, but an optimal solution of the relaxed problem (the concatenation of the subproblem solutions) will generally violate the coupling constraints. In order to satisfy these constraints, a major iteration of a traditional price-directive decomposition approach for convex optimization utilizes a subproblem phase in which the problems (2) are modified by altering their objective functions (usually by adding infeasibility penalties via Lagrangian-related terms) and then employs a coordination phase in which the solutions of these modified subproblems are combined (usually with linear weights) subject to the coupling constraints to obtain feasible solutions of the original problem. For continuous optimization problems, these weights are continuous variables and the coordination problem is generally a tractable continuous optimization problem. However, for discrete optimization problems, continuous weights would generally destroy the integrality of the subproblem solutions, so the coordination problem in this case is a more difficult discrete optimization problem. The approach that we consider here can also be viewed within this subproblem-coordination framework, but the techniques used for both subproblem generation and for coordination differ significantly from traditional decomposition methods. One major difference in our approach is that we employ the techniques of genetic algorithms (GA's) in the coordination phase. The concept of combining parts of solutions to obtain better solutions is central to GA's and hence meshes well with the coordination paradigm. However, while traditional GA's focus on the original problem variables and encode solutions as binary strings, the "genes" in our high-level GA correspond to encodings of subproblem solutions or collections of subproblem solutions.

## 2 Minimum-perimeter equi-partition

We will first outline our approach in the case of a particular class of graph partitioning problems, and then consider generalizations to other block-angular problems. In order to establish a simple geometric viewpoint, consider a rectangular grid $\mathcal{G}$ of unit-area cells and define the *minimum perimeter problem* MP($\mathcal{G}$, $P$) to be the equi-partition of the grid into $P$ components (whose areas differ by at most one unit) of minimum total perimeter. Graph partitioning of large 5-point uniform grids arises in the context of the parallel solution of PDEs using finite difference schemes [Str89] in a parallel computing environment, in computer vision [Sch89] and in the simulation of molecular behavior in chemical engineering. It also is needed in the parallel solution of large-scale maximum-flow problems arising from the study of magnetic flux lines in super-conductors. The size of the graph required for the study of the asymptotic behavior of these flux lines renders large versions of this problem impossible to solve in a single workstation mainly because of memory requirements (up to 127MB of memory is required even for relatively small problems of this category). Therefore, a decomposition of the graph into smaller pieces that fit in the memory of a single node of a Network of Workstations or a CM-5 or an SP2 is required. Since preflow push maintains a list of active nodes and performs "flow push" on the immediate neighbors of the selected node, it performs local computations on a graph that is logically a 5-point rectangular grid. To minimize interprocessor communication we need to minimize the borders of the region each processor occupies. Under the assumption that all processors are equally powerful, load balancing becomes equivalent to equi-partitioning, i.e., the area of the domain assigned to each processor differs by no more than one from the area of the domain assigned to any other processor.

Letting the binary variable $x_i^p$ correspond to the possible assignment of processor $p$ to cell $i$, $\mathcal{I}$ denote the set of pairs of adjacent cells, and $A_p$ the area (i.e., total number of cells to be assigned) for processor $p$, the simplest algebraic expression of the minimum perimeter problem is as a Quadratic Assignment Problem (QAP) [PRW93]:

$$\min \sum_{i,j \in \mathcal{G}} \sum_{\substack{p,p'=1 \\ p \neq p'}}^{P} c_{ij} x_i^p x_j^{p'} \tag{3}$$

$$s.t. \begin{cases} \sum_{i \in \mathcal{G}} x_i^p = A_p & p = 1 \dots P \\ \sum_{p=1}^{P} x_i^p = 1 & i \in \mathcal{G} \\ x_i^p \in \mathbf{B} = \{\mathbf{0}, \mathbf{1}\} \end{cases}$$

$$\text{where } c_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \mathcal{I} \\ 0 & \text{else} \end{cases}$$

and the partition sizes $A_p$ are chosen such that their cardinalities differ by at most 1. Note that this objective function does not count the edges of the boundary of the original domain $\mathcal{G}$, which are part of the total perimeter of any partition, but since this contribution is simply a fixed constant, it may be ignored in setting up an optimization problem, and then added to the optimal value of that problem to obtain the value of the total perimeter of the solution. While this algebraic formulation of the objective function is compact and is useful for comparisons with methods for the QAP, it does not match the block separable format of the block angular problem (1) that we wish to use for algorithmic purposes below, so we observe that the objective can also expressed as $\sum_{p=1}^{P} \mathcal{P}(x_p)$, where $\mathcal{P}(x_p)$ is the perimeter of component $p$ and $x_p$ is the block of variables $x_i^p$ where $p$ is fixed and $i$ ranges over $\mathcal{G}$.

Deletion of the coupling constraints $\sum_{p=1}^{P} x_i^p = 1,\ i \in \mathcal{G}$ yields a relaxed problem that decomposes into $P$ independent subproblems of the form:

$$\min \mathcal{P}(x_p) \tag{4}$$

$$s.t. \begin{cases} \sum_{i \in \mathcal{G}} x_i^p = A_p \\ x_i^p \in \mathbf{B} = \{\mathbf{0}, \mathbf{1}\}. \end{cases}$$

Each such subproblem requires the determination of the minimum perimeter of a collection of $A_p$ cells in the domain. In most cases of interest, the domain $\mathcal{G}$ will contain at least one embedded square of area at least $A_p$, in which case it can be shown that the optimal value of the subproblem is precisely $2\left\lceil 2\, A_p^{1/2} \right\rceil$. (It is possible to establish optimal subproblem values [CM96a] even if $\mathcal{G}$ is so elongated that it does not contain even one such square, but we will not consider such unusual cases here.) In addition, as shown in [YM92] it is possible to construct all possible optimal solutions to the subproblems, essentially by determining those rectangular "frames" with perimeter is $2\left\lceil 2\, A_p^{1/2} \right\rceil$ and area at least $A_p$ (there are $O(A_p^{1/4})$ such frames). If $h$ is the height of such an optimal frame, we will refer to $h$ as an optimal height in the discussion below. As we will see, the subproblem solutions, which we will refer to as "optimal shapes" and the corresponding optimal heights are related to the building blocks of the GA's that we develop below.

If one can tile together $P$ such optimal shapes to completely cover the grid, it follows that one has an optimal solution to the minimum perimeter problem, since the value of the solution will match the lower bound (obtained by summing the optimal values of the subproblems):

$$LB = \sum_p 2 \left\lceil 2\, A_p^{1/2} \right\rceil . \tag{5}$$

As we will see below, this lower bound tends to be quite good in cases in which the original domain is rectangular and $P$ is relatively large. On the other hand, it is easy to construct examples in which a tiling that uses only optimal shapes does not exist and hence this lower bound cannot be attained (consider, for example, the minimum perimeter equi-partition of a $3 \times 3$ grid into two components of areas 4 and 5). Hence, the general approach that we will consider involves the generation of optimal solutions to the subproblems above (or variants of these subproblems with suitably modified objective functions), accompanied by coordination procedures that combine (and sometimes modify) subproblem solutions to produce good feasible solutions to the original problem. We first consider rectangular domains, and then show how a GA for that problem class may generalized and applied to non-rectangular domains.

# 3 Stripe Decomposition for Rectangular Domains

In this section we consider the case in which the domain $\mathcal{G}$ is an $M \times N$ rectangular grid and we will initially assume that $P$ divides $MN$ (so that each component has equal area $A = MN/P$). For this class of problems, we will first describe an algorithm based on using a knapsack algorithm for coordination, and then consider a GA that represents an alternative coordination approach and applies to more general problem classes.

Under these assumptions, the rectangular domain may be partitioned into a collection of horizontal "stripes" with heights $h_i \leq A$ with the property that the total area of each stripe is a multiple of $A$ and that the sum of the stripe heights is $M$ [Mar96]. Heights $h \leq M$ with the property that $hN$ (the area of the corresponding stripe) is a multiple of $A$ will be termed "knapsack-valid"; observe that $min\{A, M\}$ is always a knapsack-valid height. Both the knapsack and GA approaches that we now discuss utilize stripe heights as their primary variables.

Before detailing the knapsack approach, we observe that even the single parameter family corresponding to the case of $M = N = P$ yields graph partition problems that are extremely difficult for commonly used techniques. To provide some insight into why this is true, note that the symmetry of the problem implies that in any optimal solution each component may be assigned an arbitrary distinct index $p$. This implies that there are at least $N!$ algebraically distinct optimal solutions. While this property is advantageous for GA's and other local search procedures, in branch-and-bound procedures it requires the consideration of enormous numbers of intermediate nodes in the tree in order to prove optimality. An additional disadvantage in the case of branch-and-bound applied to a standard equivalent linear integer programming formulation of (3) is that an optimal solution of the initial linear programming relaxation is obtained by setting all $x_i^p = 1/N$. To see this, note that for $N \geq 2$, the objective value of this feasible solution will match the linear program's lower bound of 0 (this is true for any term-wise equivalent linear formulation of the QAP, such as that obtained by replacing each quadratic objective term that has a non-zero coefficient by a non-negative linear variable $y(i,p,j,p')$ subject to the added constraint $y(i,p,j,p') \geq x_i^p + x_j^{p'} - 1$). Branch-and-bound can be expected to behave poorly when there is a large gap between the optimal value of the initial linear programming relaxation and the true optimal value (which in this case is $O(N^{3/2})$) . Test runs with a variety of strategies using the well-known CPLEX commercial mixed-integer-programming branch-and-bound code verified that even with a good branching strategy on the very small problem corresponding to N=5, the branch-and-bound method was unable to prove optimality after running for 6000 seconds and generating more than 45,000 nodes in the tree. (In contrast, the knapsack approach solved to optimality problems with N as large as 1000 in less than 1 second.)

The knapsack approach was motivated by error bound results [CM96b] for the special case $P = M \geq N$. These error bound results demonstrated the asymptotic optimality of solutions corresponding to a particular stripe decomposition. Specifically, it was shown that stripes of optimal or near-optimal height could be employed in a partition of the rows of the domain, and that these stripes could be filled with optimal or near-optimal shapes. The result of coordinating subproblem solutions in this way was to produce a feasible solution with objective value $v$. Letting $\delta$ denote the relative distance of this solution from the lower bound, i.e., $\delta = (v - LB)/LB$, where $LB$ is the lower bound, we showed

$$\delta < \frac{1}{\lceil 2\sqrt{N} \rceil}. \tag{6}$$

Thus, as $N \to \infty$, this relative gap tends to 0, and we say that this family of solutions is asymptotically optimal. (See [CM96a] for analogous results for more general domains.) While this result establishes the quality of a particular stripe decomposition under certain assumptions on the parameters of the problem, the knapsack approach assumes only that $P$ divides $MN$ and determines a stripe decomposition that is optimal with respect to a particular technique for assigning cells within stripes. This is done by computing for each knapsack-valid stripe height the total perimeter of the shapes obtained by filling the stripe in a column-wise fashion from left to right. The corresponding shapes can be considered as optimal solutions of modified subproblems in which Lagrangian penalty terms are added to the objective to force assignments to the leftmost unassigned cells within the stripe. The corresponding total perimeter for the filled stripe is used as the objective coefficient associated with this stripe height, and coordination of stripe heights for this class of problems may be accomplished via a knapsack problem that minimizes the corresponding objective function subject to the single constraint that the the stripe heights add to $M$.

Algebraically, assuming $n$ valid heights and letting $h_i$ denote a valid stripe height and $c_i$ its associated perimeter contribution, this knapsack problem is:

$$\min \sum_{i=1}^{n} c_i x_i \tag{7}$$

$$s.t. \begin{cases} \sum_{i=1}^{n} h_i x_i = M & \\ x_i \in \mathbb{N} & i = 1 \ldots n \end{cases}$$

Figure 1 shows the relative distance from the lower bound of the best stripe form partition computed by the knapsack method (denoted as MSP for minimum stripe partition ) for a set of problems of the form MP($10N \times 10N$, $N$); here the area assigned to each processor is $100N$ and since the number of processors is 10 times smaller than each dimension of the grid, the error bound (6) does not apply. However, one can clearly see that the resulting partitions are of excellent quality. For example, the solution obtained for the $10,000 \times 10,000$ grid partitioned among 1000 processors was within 0.042% of the lower bound.

The MSP algorithm is very fast; this is due to the fact that stripe filling is linear in the size of the grid, and that the knapsack problem is of relatively small dimension. Table 1 gives the results of comparisons between our approach and two other popular (and well established) graph partitioning methods: recursive spectral bisection (RSB) and the Geometric Mesh Partitioner (GEOMETRIC). For RSB, we used the Chaco package [HL95] that was provided to us by B. Hendrickson and R. Leland. Chaco is written entirely in ANSI C. We obtained a version of the Geometric Mesh Partitioner from the ftp site indicated in [GMT95]. It is written in MATLAB which helps explain the

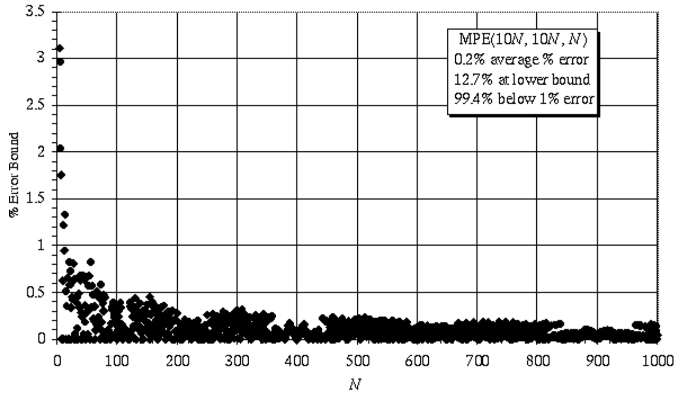


Figure 1: Distance from Lower Bound of Knapsack Solutions

long response times for some of the larger problems in our test-suite. MSP is written in FORTRAN 77. All programs run on a Sun SPARC-Station 20 workstation. The column labeled Gap gives the relative distance (in percentage terms) from lower bound. (Note that since the lower bound is not necessarily the optimal value, the distance from optimality may be much less.)

| PROBLEM | | RSB | | GEOMETRIC | | MSP | |
|---|---|---|---|---|---|---|---|
| $M \times N$ | P | Time | Gap% | Time | Gap% | Time | Gap% |
| 7 x 7 | 7 | - | - | - | - | 0.01 | 0.00 |
| 32 x 31 | 8 | 1.8 | 6.52 | 43.6 | 5.43 | 0.01 | 1.09 |
| 32 x 30 | 64 | 3.0 | 6.25 | 90.4 | 6.25 | 0.01 | 0.00 |
| 100 x 100 | 8 | 9.0 | 9.33 | 111.0 | 7.39 | 0.04 | 5.63 |
| 128 x 128 | 128 | 85.5 | 14.13 | 539.9 | 7.13 | 0.04 | 1.63 |
| 256 x 256 | 256 | 227.8 | 13.25 | 3304.2 | 4.15 | 0.09 | 0.00 |
| 512 x 512 | 512 | - | - | - | - | 0.25 | 0.14 |

Table 1: Comparison of Spectral Bisection, Geometric and MSP

Note that neither RSB nor GEOMETRIC could solve the first problem because these recursive methods require the number of partitions to be a power of two. For the last problem, RSB and GEOMETRIC ran out of memory while trying to construct the adjacency matrix of the graph. In those cases in which all of the algorithms produced solutions, the solution quality (as measured by the relative gap) of MSP was markedly superior. (We observe here that these test problems were also run on the widely-used METIS graph partitioner, which employs multi-level heuristics, and the GA to be described below. The quality of the METIS solutions was comparable to that of RSB, while the GA produced solutions nearly identical to those of MSP, but required run times comparable to RSB.) These results show that by using well-chosen building blocks corresponding to groups of subproblem solutions and by appropriately coordinating these groups either by the knapsack or GA method, it is possible to substantially outperform state-of-the-art graph partitioning codes, which do not take advantage of problem structure. (Details of additional computational tests and comparisons are given in [Mar96].)

However, it is difficult to generalize this knapsack coordination approach to handle cases in which $P$ does not divide $MN$ or cases of non-rectangular domains. In the next section, we consider a more general approach based on utilizing GA's for the coordination of stripe heights. This GA approach produces essentially the same solutions as MSP for the class of rectangular domains to which MSP applies, but in addition has produced excellent results over a broad range of large-scale problems.

# 4 Snake Decomposition for General Domains

## 4.1 Snake Decomposition

We now describe how stripe decomposition can be modified in order to handle general domains. Figure 2 shows the solution produced by applying a GA based on this generalized approach to an discretized ellipsoidal domain. We need not assume that $P$ divides the area of the grid and we

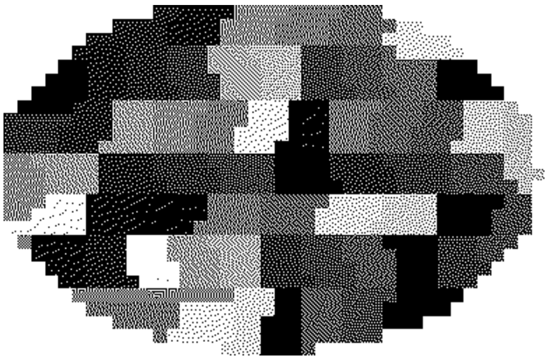

Figure 2: Partition of an ellipsoidal domain among 64 procs

will relax the requirement of dealing with "knapsack-valid" stripe heights. Observe that a general 5-point grid domain can be enclosed within its "rectangular hull", i.e., the smallest rectangle enclosing all of the grid. *Snake decomposition* extends stripe decomposition by accepting as input a partition of the rows of the rectangular hull of the grid, and then "snaking" through the domain, filling the columns of these stripes with processor indices top to bottom, column after column (within each stripe) first going left to right, then right to left for the next stripe. (As we will discuss below, this is essentially a procedure for transforming a genotype (comprised of an array of stripe heights) into a phenotype (a set of assignments of cells to processors). Of course, alternative such transformation procedures are possible. For example, the assignments could be done row-wise (with bounds on row lengths related to optimal widths), allowing each component to "spill" into the next stripe.) Snake decomposition differs from stripe decomposition in the sense that stripes that cannot be filled with an integer number of components are permitted and are handled by allowing the fill procedure to "overflow" to the next stripe so that stripes are interdependent. A component corresponding to such an overflow can be thought of as an optimal solution to a subproblem in which there is a hierarchy of penalties added to the original objective: very large penalties for cells already assigned or not in the region spanned by the two stripes under consideration and smaller penalties for assignments to appropriate cells of the second stripe. Once a partition of the rows of the rectangular hull of the grid is given, the `snake` process assigns the cells of the grid to the available processors in linear time ($O(|\mathcal{G}|)$). However, because the overflow process blurs the boundaries between stripes, the determination of good partitions becomes more difficult because it is no longer possible to associate a unique perimeter with a stripe height as was the case with the knapsack approach. Since an objective function analogous to the one in the knapsack problem that provides total perimeter as a function of stripe heights cannot be constructed for these more general cases, we consider more general mechanisms for coordinating stripes. In the next section, we describe multi-coordination techniques, i.e., procedures in which multiple coordination steps are performed in parallel, based on the use of genetic algorithms applied to high-level encodings.

## 4.2 Multi-Coordination via a Genetic Algorithm

The initial GA that we considered for this problem class used an encoding in which each individual was represented by a vector of optimal shape indices [YM92]. That is, an allele corresponded to a particular optimal shape chosen from a library of optimal shapes, and the procedure for cell assignment was designed to construct an assignment similar to the one corresponding to that optimal shape. This approach was similar to the Grouping GA approach of [Fal94]. However, we have now developed a higher-level GA in which the alleles correspond to stripe heights, which can be regarded as collections of subproblem solutions. This higher-level approach is faster and has provided better results for our collection of test problems.

To evaluate the "fitness" of an individual produced by crossover and mutation we first "repair" the height values so that they sum to the height of the rectangular hull (at present this is done by changing each allele as little as possible, but other alternatives, such as directing changes toward the set of optimal heights, are also under consideration). We then employ a snake process to generate a feasible solution (optionally followed by a local search starting at that feasible solution). We use a distributed and asynchronous GA, termed DGA, that employs the *island model* of computations [MSB91, Lev95] where the algorithm spawns a number of processes, each of which is a GA running independently and asynchronously of the others

Each island GA process is equipped with a communication mechanism that is essentially nothing more than simple send and receive calls and a broadcast feature available in most message passing systems that allows it to send or receive incoming individuals according to certain migration criteria that are based on island load (population density of each island). The population in each island can vary (as the DNA length of each individual can, since an individual is an array of integers adding up to the number of rows in the rectangular hull of the grid) but cannot exceed a population limit. Depopulated islands attract well-fit immigrants from other islands.

To help prevent premature convergence phenomena due to the appearance of "super-individuals" early on in the evolutionary process, "age" is utilized to eliminate from the population any individual that has reached its life-limit (which is a random variable following the normal distribution with a mean that is directly proportional to its fitness value).

Each island process uses 1-point crossover with a crossover rate of 0.8%, it has a mutation rate of 0.1% and a maximum population of 16 individuals. A Roullete-wheel strategy was used for selecting the mating individuals. Since DGA follows the *steady-state* approach, approximately 70% of the population is considered for mating at each iteration. Finally, when an individual is born, if there is an empty slot in the island, it is inserted there, else it replaces the worst individual in the population. Migration occurs when an island's population is less than half the population of another island, and the populated island's best individuals have stayed in their home island for at least 3 years (iterations). DGA was allowed to run for 20 generations. It was implemented in C and uses the PVM 3.3.10 message passing interface for all communication required [GBD+94]. DGA runs on a Cluster of Workstations (COW) available at the Computer Sciences Dept. at the University of Wisconsin - Madison. Each workstation is a Sun SPARC-Server 20 running at 66MHz, and comes equipped with 64MB of RAM. We used an 8 node partition of the cluster for our experiments. We tested DGA against RSB and RSQ (Recursive Spectral Quadrisection) as implemented in the Chaco package from Sandia National Laboratories, as described earlier. The results for some non-rectangular domains (see table 2, in which SIZE denotes the number of cells of the domain) are shown in table 3 and Gap denotes the relative distance from lower bound. (It should be noted that the quality of the lower bound from the theory of optimal shapes is not as good for non-rectangular domains as for rectangular grids, since the theory does not take into account that shapes containing portions of the irregular boundary will generally have non-minimal perimeter. Hence, the gaps for DGA do not necessarily mean that its solutions are far from optimal.)

It is clear that DGA can find significantly better partitions than RSB or RSQ for uniform 5-point (regular or irregular boundary) grids, and that it compares well with them in response time as well.

| PROBLEM | SIZE |
|---|---|
| circle | 7800 |
| torus | 7696 |
| diamond | 4019 |
| ellipse | 823 |

Table 2: Sizes of Non-rectangular Test Problems

| PROBLEM | | RSB | | RSQ | | DGA (8 procs) | |
|---|---|---|---|---|---|---|---|
| Shape | P | Time | Gap% | Time | Gap% | Time | Gap% |
| circle | 16 | 23.3 | 24.44 | 9.1 | 21.80 | 20.29 | 8.47 |
| circle | 64 | 34.7 | 16.87 | 14.5 | 28.34 | 17.5 | 5.87 |
| ellipse | 16 | 2.3 | 10.83 | 1.4 | 13.33 | 6.4 | 8.33 |
| ellipse | 64 | 3.5 | 5.16 | 2.2 | 15.10 | 7.5 | 5.56 |
| torus | 16 | 27.3 | 28.97 | 12.5 | 32.67 | 16.7 | 11.50 |
| torus | 64 | 36.5 | 22.86 | 18.5 | 34.3 | 13.8 | 11.00 |
| diamond | 16 | 14.0 | 38.67 | 6.5 | 35.74 | 10.4 | 16.40 |
| diamond | 64 | 18.7 | 29.78 | 9.0 | 28.80 | 14.7 | 13.37 |

Table 3: Spectral Methods vs. DGA on non-rectangular grids

DGA was also compared against the widely-used METIS mesh partitioner, which uses a variety of multilevel heuristics. The quality of solutions produced by METIS was comparable to that RSB and RSQ and therefore had an optimality gap typically 2 to 3 times poorer than that of DGA on the non-rectangular grids. (For rectangular grids, the difference in performance was much more dramatic, as may be seen from Table 1, noting that the quality of the DGA solutions was essentially the same as that of MSB.)

# 5 Directions for Further Research

We plan to investigate the augmentation of the `snake` procedure for fitness evaluation by a post-processing local search mechanism. Promising candidates include variants of Kenighan-Lin [KL70] tailored to focus on boundary cells [KK95], tabu search, and simulated annealing. In addition, we will consider methods involving more general swap cycles (as opposed to pairwise swaps) based on linear network approximations to the QAP and focused on boundary cells of components with poor perimeter and neighboring components. Finally, the `snake` process will be generalized to allow the user to specify a range of acceptable processor areas as opposed to strict balancing constraints. The user will thus be allowed to assess the corresponding tradeoff between balance and communications cost in the original application.

An interesting aspect of the encoding in terms of stripe heights is that the fitness computation requires the application of `snake` (optionally followed by a local search), which can be time-consuming for large problems. Under these circumstances the investigation of GA variants by means of virtual GA's [Gre95] (in which fitness evaluations are replaced by statistical models of fitness) offers the possibility of streamlining the GA construction process. Our statistical models could extend the approach of [Gre95] by taking advantage of data on quality indices for slices or slice "overflow" at breakpoints.

The approach described above for two-dimensional graph partitioning can be readily extended to three dimensions. The slices in that case are determined by planes orthogonal to one of the axes. In fact, the approach of fitting together, via multi-coordination techniques such as GA's, good or optimal solutions to subproblems, applies to a broad range of areas. In [YM92] we reported the successful application of a similar decomposition approach to data partitioning problems arising from parallel database design. This problem class has the same constraints as (3) but an objective function that represents the minimization of communication with a centralized query server as opposed to the decentralized communication represented by the minimum perimeter problem. Note that while obtaining true optimal solutions to subproblems such as minimal perimeter and surface area problems is of theoretic interest, computationally it is sufficient to simply generate good subproblem solutions, since these objects only play the role of starting points in a multi-stage coordination process that includes the capability of repairing or modifying subproblem solutions. Thus, heuristics such as GA's may be utilized for the subproblems (see [PJ94]) as well as for the coordination step. Extensions of the results to non-uniform grids, triangulations, other data partitioning problems arising from parallel database design, and other types of fixed-charge networks provide promising areas for further research for this approach. In multi-commodity fixed-charge networks, for example, the subproblems would be single-commodity subproblems corresponding to various approximations of the the fixed-charges. Recent results in telecommunications network design [Dav97] and pharmaceutical design [Har94] demonstrate that this paradigm of utilizing subproblem solutions as building blocks within the context of genetic algorithms is very effective in those problem domains as well. By focusing on the "island" GA model with appropriately designed high-level problem representations to reduce communication costs and well-designed procedures for subproblem multi-coordination, we anticipate that the evolutionary paradigm of genetic algorithms will prove to be a very effective technique for solving structured large-scale combinatorial optimization problems in distributed computing environments.

# References


[CM96a] I. T. Christou and R. R. Meyer. Optimal and asymptotically optimal equi-partition of rectangular domains via stripe decomposition. In H. Fischer, B. Riedmuller, and S. Schaffler, editors, *Applied Mathematics and Parallel Computing - Festschrift for Klaus Ritter*, pages 77–96. Physica-Verlag, 1996.

[CM96b] I. T. Christou and R. R. Meyer. Optimal equi-partition of rectangular domains for parallel computation. *Journal of Global Optimization*, 8:15–34, January 1996.

[Dav97] L. D. Davis. Hybrid genetic algorithms for optimization. In *Proceedings of IMA Evolutionary Algorithms Workshop*. Springer-Verlag, 1997. To appear.

[Fal94] Emanuel Falkenauer. A new representation and operations for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 User's Guide and Reference Manual*. Oak Ridge National Laboratory, 1994.

[GMT95] J. R. Gilbert, G. L. Miller, and S. H. Teng. Geometric mesh partitioning: Implementation and experiments. In *Proceedings of the 9th International Symposium on Parallel Processing*, pages 418–427, 1995.

[Gre95] J. J. Grefenstette. Virtual genetic algorithms: First results. Technical Report AIC-95-013, Navy Center for Applied Research in AI, 1995.

[Har94] W. E. Hart. *Adaptive global optimization with local search*. PhD thesis, University of.California, San Diego, 1994.

[HL95] B. Hendrickson and R. Leland. *The Chaco User's Guide Version 2.0*. Sandia National Laboratories, July 1995.

[KK95] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report 95-064, Department of Computer Science, University of Minnesota, 1995.

[KL70] B. W. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *Bell Systems Tech. Journal*, pages 291–308, February 1970.

[Lev95] D. Levine. *User's Guide to the PGAPack Parallel Genetic Algorithm Library Version 0.2*. Argonne National Laboratory, June 1995.

[Mar96] W. Martin. Fast equi-partitioning of rectangular domains using stripe decomposition. Technical Report MP-TR-96-2, University of Wisconsin - Madison, February 1996.

[MSB91] H. Muhlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. In R. Belew and L. Booker, editors, *Proceedings of the Fourth Intl. Conference on Genetic Algorithms*, pages 45–52. Morgan Kaufmann Publishers, Los Altos, CA, 1991.

[PJ94] Mitchell A. Potter and Kenneth De Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings of the Third International Conference on Parallel Problem Solving from Nature*. Springer-Verlag, 1994.

[PRW93] P. M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem: A survey and recent developments. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*. American Mathematical Society, 1993.

[Sch89] R. J. Schalkoff. *Digital Image Processing and Computer Vision*. John Wiley & Sons, 1989.

[Str89] J. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Wadsworth & Brooks, 1989.

[YM92] J. Yackel and R. R. Meyer. Optimal tilings for parallel database design. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 293–309. North - Holland, 1992.