Asynchronous parallel search by the parallel genetic algorithm

Heinz Mühlenbein
GMD Schloss Birlinghoven
D-5205 Sankt Augustin1

Abstract

The parallel genetic algorithm (PGA) is a prototype of a new kind of a distributed algorithm. It is based on a parallel search by individuals all of which have the complete problem description. The information exchange between the individuals is done by simulating biological principles of evolution. The PGA is totally asynchronous, running with maximal efficiency on MIMD parallel computers. The search strategy of the PGA is based on a small number of intelligent and active individuals, whereas a GA uses a large population of passive individuals. We will show the power of the PGA with two combinatorial problems - the graph partitioning problem and the autocorrelation problem. In these examples, the PGA has found solutions of very large problems, which are comparable or even better than any other solution found by other heuristics.

1 Introduction

Random search methods based on evolutionary principles have been already proposed in the 60's. They did not have a major influence on mainstream optimization. We believe that this will change. The unique power of evolutionary algorithms shows up with parallel computers. Firstly, our parallel genetic algorithm PGA introduced in 1987 [19] runs especially efficient on parallel computers. Secondly, our research indicates that parallel searches with information exchange between the searches are often better than independent searches. Thus the PGA is a truly parallel algorithm which combines the hardware speed of parallel processors and the software speed of intelligent parallel searching.

We have successfully applied the PGA to a number of problems, including function optimization [21] and combinatorial optimization. In this paper we summarize the results for the graph partitioning problem and the low autocorrelation problem. The quadratic assignment problem has been published elsewhere [17]. The travelling salesman problem is discussed in [18].

We believe that the idea on which the PGA is based can be extended far beyond optimization problems. The PGA is totally asynchronous. It runs with graceful degradation. The algorithm works as long as one search is running. On the other hand, we have shown in [21] that in certain instances a superlinear speedup can be observed if all searches run in parallel.

2 Parallel search and optimization

In this paper we consider the following problem:

OPT 1 *P: Given a function $F : X \mapsto R$, where X is some metric space. Let S be a subspace of X. We seek a point x in S which optimizes F on S or at least yields an acceptable approximation of the suprenum of F on S.*

Many optimization methods have been proposed for the solution of this problem. We will investigate parallel optimization methods. A parallel optimization method of parallelism N is characterized by N different search trajectories, which are performed in parallel. It can be described as follows

$$x_i^{t+1} = G_i(x_1^t, ..x_N^t, F(x_1^t), ..F(x_N^t))i = 1, ..., N \quad (1)$$

The mapping $G = (G_1, ...G_N)$ describes the linkage or information exchange between the parallel searches. If the N searches are independent of each other we just have

$$x_i^{t+1} = G(x_i^t, F(x_i^t)) \quad (2)$$

A parallel search method which combines the information of two searches can be described as follows

$$x_i^{t+1} = G_i(x_{i-1}^t, x_i^t, F(x_{i-1}^t), F(x_i^t))i = 1, ..., N \quad (3)$$

The basic questions of parallel search methods can now be stated

- Are N parallel searches of time complexity t as efficient as a single search of time complexity $N*t$?
- Are N linked searches more efficient than N independent searches?
- How should the linkage be done?

In order to understand these questions intuitively, we leave the abstract mathematical description and turn to a natural search metaphor. The advantage of using a metaphor is that it leads to a qualitative understanding of the problem and the algorithm.

In this paper, we will investigate search algorithms which mimic evolutionary adaptation found in nature. Each individual is identified with an animal, which searches for food and produces offspring. In evolutionary algorithms, $F(x_i)$ is called the fitness of individual i, x_i^{t+1} is an offspring of x_i^t, and G is called the selection schedule. One of the most successful evolutionary algorithms is the genetic algorithm invented by Holland [12].

3 Genetic algorithms

Recent surveys of genetic algorithms can be found in [11] and [?]. The basic genetic algorithm can be described as follows:

Genetic Algorithm

STEP0: Define a genetic representation of the problem

STEP1: Create an initial population $P(0) = x_1^0, ..x_N^0$

STEP2: Compute the average fitness $\overline{F} = \sum_i^N F(x_i)/N$. Assign each individual the normalized fitness value $F(x_i^t)/\overline{F}$

STEP3: Assign each x_i a probability $p(x_i, t)$ proportional to its normalized fitness. Using this distribution, select N vectors from $P(t)$. This gives the set $S(t)$

STEP4: Pair all of the vectors in $S(t)$ at random forming $N/2$ pairs. Apply crossover with probability p_{cross} to each pair and other genetic operators such as mutation, forming a new population P_{t+1}

STEP5: Set $t = t + 1$, return to STEP2

In the simplest case the genetic representation is just a bitstring of length n, the "chromosome". The positions of the strings are called "locus" of the chromosome. The variable at a locus is called "gene", its value "allele". The set of chromosomes is called the "genotype"' which defines a "phenotype" (the individual) with a certain fitness. The crossover operator links two searches. Part of the chromosome of one individual (search point) is inserted into the second chromosome giving a new individual (search point). We will later show with examples why and when crossover guides the search.

A genetic algorithm is a parallel random search with centralized control. The centralized part is the selection schedule. For the selection the average fitness of the population is needed. The result is a highly synchronized algorithm, which is difficult to implement efficiently on parallel computers.
In our parallel genetic algorithm, we use a distributed selection scheme. This is achieved as follows. Each individual does the selection by itself. It looks for a partner in its neighborhood only. The set of neigborhoods defines a spatial population structure.

Our second major change can now easily be understood. Each individual is active and not acted on. It may improve its fitness during its lifetime by performing a local search.
A generic parallel genetic algorithm can be described as follows

Parallel genetic algorithm

STEP0: Define a genetic representation of the problem

STEP1: Create an initial population and its population structure

STEP2: Each individual does local hill-climbing

STEP3: Each individual selects a partner for mating in its neighborhood

STEP4: An offspring is created with genetic crossover of the parents

STEP5: The offspring does local hill-climbing. It replaces the parent, if it is better than some criterion (acceptance)

STEP6: If not finished, return to STEP3.

It has to be noted, that each individual may use a different local hill-climbing method. This feature will be important for problems, where the efficiency of a

particular hill-climbing method depends on the problem instance.

In the terminology of section 2, we can describe the PGA as a parallel search with a linkage of two searches. The linkage is done probabilistically, constrained by the neighborhood. The information exchange within the whole population is a diffusion process because the neighborhoods of the individuals overlap.

There have been several other attempts to implement a parallel genetic algorithm. Most of the algorithms run k identical standard genetic algorithms in parallel, one run per processor. They differ in the linkage of the runs. Tanese [23] introduces two *migration* parameters: the *migration interval*, the number of generations between each migration, and the *migrationrate*, the percentage of individuals selected for migration. The subpopulations are configured as a binary n-cube. A similar approach is done by Pettey et al. [22]. In the implementation of Cohoon et al. [2] it is assumed that each subpopulation is connected to each other. The algorithm from Manderick et al. [15] has been derived from our PGA.

All but Manderick's algorithm use subpopulations that are densely connected. We have shown in [18] why restricted connections like a ring are better for the parallel genetic algorithm. All the above parallel algorithms do not use hill-climbing, which is one of the most important parts of our PGA.

An extension of the PGA, where subpopulations are used instead of single individuals, has been described in [21]. This algorithm outperforms the standard GA by far in the case of function optimization. It is also a better search method than most of the standard mathematical methods.

4 The search strategy of the PGA

The search strategy of the PGA is governed by three components - *the crossover operator,* the *spatial population structure* and the *hill-climbing strategies*. We will briefly discuss the effects of these components.

The crossover operator is the fundamental part of any genetic algorithm. There have been attempts to "prove" that genetic algorithms make a nearly optimal allocation of trials because of crossing-over. This result is called the "Fundamental Theorem of Genetic Algorithms" [11]. We have shown in [18] that the above claim is only valid for very simple optimization problems. The search strategy of a genetic algorithm can be explained in simpler terms. The crossover operator defines a *scatter search* [6] where new points are drawn out of the area which is defined by the old or "parent" points. The more similar the old points are, the smaller will be the sampling area. Thus crossing-over implements an adaptive step-size control. Why and when does this search strategy make sense?

Let us assume that the combinatorial problem has the *building block feature*. We speak of a building block feature if the substrings of the optimal solutions are contained in other good solutions. In this case it seems a good strategy to generate new solutions by patching together substrings of the old solutions. This is simply what the crossover operator does. Unfortunately the crossover operator does not know which substrings are part of the optimal solution. So it combines randomly the two strings ("chromosomes") of the parents.

The major difficulty is to define a crossover operator which creates valid solutions i.e. solutions which fulfill the constraints of the problem. We will explain this problem with the graph partitioning problem in the next section.

The introduction of a spatial population structure with selection done by the individuals themselves changes the PGA to an *asynchronous distributed algorithm*. Each individual of the PGA is a complete problem solver. It has the knowledge about the problem and a complete program for obtaining an approximate solution. The PGA does not use any synchronization. Each processor sends its latest results to the neighboring processors. If an individual on a specific processor looks for a partner for mating, it uses the information which is contained in the local copies. There is no complex handshaking protocol between individuals to get the latest information. Therefore the PGA is extremely fault-tolerant. It works with a few number of individuals, but the results are better if a larger number of individuals is used. The PGA also continues to work, if a processor fails or an individual stops during a run.

The GA community is not yet convinced that using a local hill-climbing method in genetic algorithms pays off. Our research indicates that a PGA with a good hill-climbing strategy performs better than a PGA with a simpler strategy or a PGA without any hill-climbing strategy. This has been shown for function optimization [21], the travelling salesman problem and a difficult 30-bit function [18].

Local hill-climbing of the individuals has an additional benefit. It increases the amount of computation compared to the amount of communication. Therefore the PGA can be tailored to the problem and the communication throughput of the parallel processor.

In summary, the PGA is a distributed algorithm

with new features which make it very attractive for massively parallel systems. The parallelism is introduced by replicating the problem. The parallel searches first explore different search areas, then they concentrate more and more on promising areas.

5 The graph partitioning problem

The graph partitioning problem (GPP) is a fundamental combinatorial problem which arises in many applications. The task is to divide a given graph into a number of partitions in order to optimize some criterion e.g. to minimize the number of edges between partitions. More formally:

Let a graph $G = (V, E, w)$ be given. $V = \{v_1, v_2, ..., v_n\}$ 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 GPP is to divide the graph into m disjunct parts, such that some optimization criteria will be fulfilled. In this paper we will consider the following optimization criteria:

OPT 2 (GPP) *Let $\mathcal{P} = \{P_1, ..., P_m\}$ be a partition. Let $\mathcal{G} = (g_1 g_2 ... g_n)$ denote the partition to which the nodes belong $(1 \leq g_i \leq m)$. Then we look for*

$$\min_{\mathcal{P}} \sum_{\substack{1 \leq i < j \leq n \\ g_i \neq g_j}} w_{ij}$$

such that $\sigma(P)$ is minimal.

$\sigma(P)$ is defined as

$$\sigma^2(P) = \frac{1}{m} \sum_{i=1}^{m} |P_i|^2 - (\frac{1}{m} \sum_{i=1}^{m} |P_i)^2$$

In order to solve the GPP, we have to define the genetic representation and the genetic operators. In the simplest representation, the value (allele) g_i on locus i on the chromosome gives the number of the partition to which node v_i belongs. But this representation is highly degenerate. The number of a partition does not have any meaning for the partitioning problem. An exchange of two partition numbers will still give the same graph partition. In fact, any permutation of the m partition numbers gives the same solution. Alltogether m! chromosomes give the same solution with the same fitness value

$$F(\mathcal{G}) = \sum_{\substack{1 \leq i < j \leq n \\ g_i \neq g_j}} w_{ij}$$

These m! chromosomes code the same partitioning instance, the same phenotype". This genetic representation does not capture the structure of the problem. We did not find a better genetic representation, so we decided that the crossover operator has to be "intelligent". Our crossover operator inserts complete partitions from one chromosome into the other, not individual nodes. It computes which partitions are the most similar to each other and exchanges these partitions. Mathematically spoken, the crossover operator works on equivalence classes of chromosomes.

Figure 1 shows an example. The problem is to partition the 4×4 grid into four partitions.

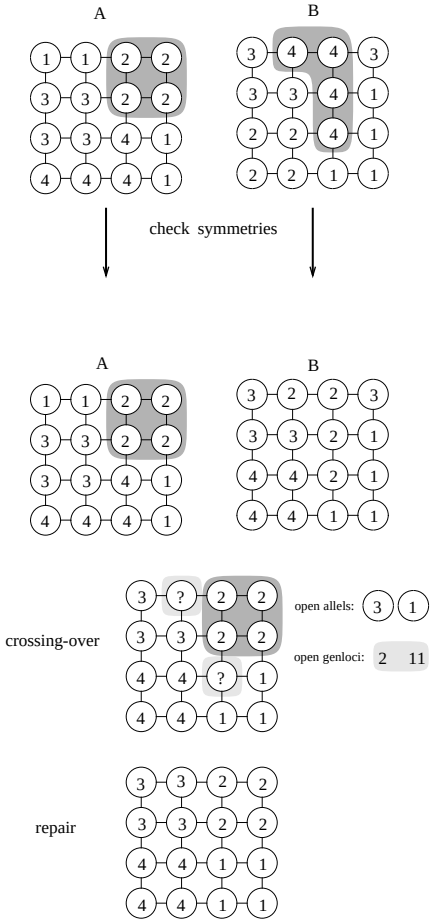

Figure 1: The crossover operator

The crossover operator works as follows. The PGA has randomly decided that partition 2 has to be inserted into $\mathcal{B}$. The crossover operator finds, that partition 4 of $\mathcal{B}$ is the most similar to partition 2 in $\mathcal{A}$. It identifies partition 2 of $\mathcal{A}$ with partition 4 of $\mathcal{B}$. Then it exchanges the alleles 2 and 4 in chromosome $\mathcal{B}$ to avoid the problems arising from symmetrical solutions. In the crossover step it implants partition 2 of

chromosome $\mathcal{A}$ into $\mathcal{B}$.

After identifying all genloci and alleles which lead to a nonvalid partition a repair operator is used to construct a new valid chromosome. Mutation is done after the crossover and depends on the outcome of the crossover. In the last step a local hill-climbing algorithm is applied to the valid chromosome.

For local hill-climbing we can use *any* popular sequential heuristic. It should be fast, so that the PGA can produce many generations. In order to solve very large problems, it should be of order $O(n)$ where n is the problem size. Our hill-climbing algorithm is of order $O(n^2)$, but with a small constant. At the start of the algorithm we reduce the size of the graph by combining up to r nodes into one hypernode. Then 2-opt is applied to the reduced graph. In later generations we apply 2-opt only to nodes which have connections to outside partitions (see [25] for details).

6 Performance evaluation for the GPP

A detailed study of the graph bipartitioning problem can be found in [13]. In that paper random graphs and random geometric graphs up to 1000 nodes are used to compare different heuristics. We decided to make a performance analysis with real life graphs. Furthermore we are more interested in the general partitioning problem, not in the bipartitioning case. Detailed results can be found in [24].

We will give here the computational results for solving two of the largest GPP benchmarking problems. The problems are called *EVER918* and *EVER1005* [4]. *EVER918* is a 3-D graph which consists of 918 nodes and 3233 edges. It has to be partitioned into 18 partitions. *EVER1005* has 1005 nodes and 3808 edges. It has to be partioned into 20 partitions. All results have been obtained on a TRANSPUTER based 64-processor system [20].

Table 1 gives a comparison to other solutions, which have been computed recently. *MultOpt* and *MultLk* are multiple runs of the local search methods 2-opt and the more general Lin-Kernighan exchange [14]. It has to be noticed, that the heuristics of Gilbert et al. [5] and of Moore [16] do not use the constraint of equal partition size. This partitioning problem is simpler, furthermore the solutions should have a smaller cost. Nevertheless, the PGA found in one case the best solution (EVER918). In the second problem, the PGA found the best solution with minimal $\sigma(|P|)$

How can the good results of the PGA be explained? We claim that GPP has the *building block feature* in some space which the crossover operator explores. The proof is analogous to the TSP case, see [18] for details. It is based on a configuration space analysis of local mimima. This analysis is much more difficult for the GPP than for the TSP problem, because of the symmetry or degeneration mentioned earlier. The analysis has to be done in the space of all equivalence classes of local minima. The equivalence classes are defined by the crossover operator.

prob.	alg.	costs	σ ($\|P\|$)
EVER918	MultOpt	1020	0.00
	MultLK	625	0.00
	GZ87	587	0.99
	Moore	453	0.99
	PGA	431	0.00
EVER1005	MultOpt	1023	0.40
	MultLk	725	0.40
	GZ87	696	1.29
	Moore	608	1.29
	PGA	631	0.40

Table 1: Comparison of GPP solutions for problem *beam*

The next combinatorial problem does not have any constraints. Here the genetic representation and the crossover operator are straightforward.

7 Low autocorrelation binary sequences

The investigation of the properties of low autocorrelation binary sequences has a long history and is of great interest for technical as well as theoretical reasons. Autocorrelation sequences play an important role in several communication engineering applications [1]. From the theoretical point of view it seems to be a very difficult optimization problem and the cost function seems to show a *golf course* like configuration space landscape with an irregular structure.

The autocorrelation function is defined on a Boolean hypercube of size n. The elements are binary squences $S = (s_1, ..., s_n)$, where s can have the value +1 or -1. To measure the quality of such a sequence, the following standard criterion, called *merit factor* F was introduced by Golay [8].

$$F = N^2/(2 * \sum_{i=1}^{n-1} R_i^2) \qquad (4)$$

$$R_i = \sum_{k=1}^{n-k} s_k * s_{k+i}; 1 \leq k \leq n-1 \qquad (5)$$

OPT 3 (Auto) *The best autocorrelation sequence of size n is given by* $F^* = \max_S F(S)$

Skew-symmetric sequences of odd length n=2m-1 are defined by

$$s_{m+l} = (-1)^l s_{m-l}, 1 \leq l \leq n-1$$

These sequences seem to be good candidates for high merit factors. In the following we will only consider skew-symmetric sequences. The configuration space can then be reduced from 2^n elements to 2^m elements.

This problem has been investigated recently by Beenker [1] , Golay [9], Wang [26] and de Groot [3]. Beenker used simulated annealing, Golay enumeration techniques, Wang and de Groot evolutionary algorithms.

The genetic representation of this problem seems to be straightforword. Just take the binary string of length m as chromosome. The phenotype is then the skew-symmetric chain of length $n = 2m - 1$. There are no constraints, each sequence is valid. So it seems that a plain PGA can solve this problem easily. But notice, that every configuration is 4-fold degenerated: inversion and sequence reversal give the same merit factor. The following four sequences give the same fitness value for the case $n = 5, m = 3$

s	1	-1	1	1	-1
s'	-1	1	-1	-1	1
s_{inv}	-1	1	1	-1	1
s_{inv}'	1	-1	-1	1	-1

With this representation, no structure of the sequences shows up. We have shown elsewhere [17], that in such cases a simple crossover will not lead to a better maximum. For a genetic algorithm, it is a bad thing to make a crossing-over between e.g. s and s'. These sequences are opposite to each other located on a m-dimensional hypercube. Crossover tries to combine these searches which are phenotypically equal but genetically opposite to each other.

The above situation does only seldom occur with the PGA. In a PGA, crossing-over is only done between nearby individuals. Thus the individuals in a neighborhood get more and more similar. So it becomes unlikely that individuals in a neighborhood are genetically very different. The breeders call this effect inbreeding.

n	Beenker	Golay	Wang	Groot	PGA
81	7.323	8.20		8.04	8.20
101	6.058	8.36	6.911	8.36	8.36
103	5.900	9.56	7.766		9.56
109		8.97			8.97
113		8.49			8.49
141	6.01			6.48	7.45
161	6.02				6.89
181	5.70			6.02	6.77
201				5.92	6.29

Table 2: Comparison of best solutions found

The above representation did not give good results, therefore we changed it slightly. The modification was motivated by an observation of Golay [10] . He showed that good skew-symmetric solutions of order n can be found by an interleaving of good symmetric and antisymmetric solutions of order $n/2$. We show for the case $n = 13, m = 7$ how this is done :

1	1	1	1	1	-1	-1	1	1	-1	1	-1	1
1		1		1		-1		1		1		1
		1		1		-1		1		-1		-1

So we tested the idea, to use two chromosomes as genetic representation. In the above example we have the following two chromosomes $c_{sym} = (1, 1, 1, -1)$ and $c_{asym} = (1, 1, -1)$ The phenotype defined by these chromosomes can be obtained by interleaving c_{sym} and c_{asym} and then expanding the string to a skew-symmetric string. This will give the string shown in the first line above.

Our local hill-climbing method is very simple. One bit is flipped on the skew-symmetric string. This change is accepted, if the fitness value increases. Then the next bit is flipped and so on.

In table 2 the computational results are given. The PGA found all solutions from Golay [10], who used an enumeration technique tailored to the autocorrelation problem. De Groot used an evolutionary algorithm without crossing-over. He kept the sample points far apart by accepting only points which had a minimum Hamming distance to all other points.

Why did the PGA perform so good? Our genetic representation does not lead to a building block feature.

The two chromosomes encode only little information about the problem.

In the autocorrelation problem the secret of success seems to be our local hill-climbing. We will demonstrate this with a small example of size $n = 31$. For $n = 31$ there exist one optimal solution with $F = 6.08228$ and four suboptimal solutions with $F = 5.52299$. After only three generations the PGA always found the best solution and three of the suboptimal solutions. We will show how the best solution was found in a typical example. In the following table the parent chromosomes and the offspring chromosome are shown in various stages of the algorithm.

state	s_{sym}	s_{asym}	F
parent1	1<u>0000</u>011	1100<u>010</u>1	3.599
parent2	0<u>1011</u>011	1101<u>111</u>0	3.599
cross	11010011	11001111	2.235
mutat.	11000011	11011111	1.115
hill-cl.	00100111	11010111	6.082

In the table we use 0 instead of -1. Crossover is done between the underlined substrings. Crossing-over and a small mutation of 1 bit leads to a string with very low fitness value. Nevertheless climbs the local search from 1.115 up to the global optimum. This behavior is different to other combinatorial problems like the GPP, where crossing-over explored the building blocks of the solution.

We are now evaluating, if random insertion of a large string instead of crossingover will give the same or better results. If this is not the case, we have to analyze, what kind of structure crossing-over is exploring in the autocorrelation problem.

8 Conclusion

The parallel genetic algorithm has been very successful applied to benchmark combinatorial optimization problems. The algorithm uses a distributed selection schedule, it self-organizes itself. The crossover operators described in this paper are not the only ones possible. Glover [7] has suggested *adaptive structured combinations* for the TSP and the graph bipartitioning problem. We have used a voting crossover operator for the quadratic assignment problem, which combines five solutions instead of two [17]. A good problem dependend crossover operator is in many applications the key to the success of the PGA.

The *clean* PGA described in this paper is a very simple, but robust distributed algorithm. The major problem is the following. In the last stage of the algorithm the same evaluations are done over and over again. The *clean* PGA does not have any memory. For a computational optimal PGA it seems to be promising to implement some kind of memory, maybe controlled by *tabu search* [7]. The population of individuals could make hypotheses about which areas should not be searched and which areas should be explored.

Other extensions are also worthwhile to explore. We suggest that one or more individuals should use a very sophisticated hill-climbing method. These individuals will be supplied with the best solutions obtained so far and try to improve it. Another extension is to change the size of the population. If a neighborhood gets to similar, then it should shrink.

We will implement these extensions in the course of applying the PGA to more and more challenging applications. We have been very surprised that the *clean* PGA was able to obtain such good results for classical optimization problems.

The success of the PGA suggest exploring other problem solving metaphors also. Why not use the market of economy as a parallel search method? This could be the beginning of another family of distributed algorithms. Moreover a comparison of problem solving by a market framework and by biological evolution on the same set of artificial problems would also give further insight into economy and biology.

References

[1] G.F.M. Beenker, T.A.C.M. Claasen, and P.W.C Hermens. Binary sequences with a maximally flat amplitude spectrum. *Phill. J. of Research*, 40:289, 1985.

[2] J.P. Cohoon, S.U. Hedge, W.N. Martin, and D. Richards. Punctuated equilibria: A parallel genetic algorithm. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154. Lawrence Erlbaum, 1987.

[3] C. de Groot, D. Würtz, and K.H. Hoffmann. Low autocorrelation binary sequences: exact enumeration and optimization by evolutionary strategies. Technical report, ETH Zürich IPS 89-09, 1989.

[4] G.C. Everstine. A comparison of three resquencing algorithms for the reduction of matrix profile and wavefront. *Int. J. Numer. Meth. in Engin.*, 14:837–853, 1979.

[5] J.R. Gilbert and E. Zmijewski. A parallel graph partitioning algorithm for message- passing multiprocessors. In *First Int. Conf. on Supercomputing*, 1987.

[6] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[7] F. Glover. Tabu search for nonlinear and parametric optimization. Technical report, University of Boulder, 1991.

[8] M.J.E. Golay. Sieves for low autocorrelation binary sequences. *IEEE Trans. on Inform Theory*, 23:43, 1977.

[9] M.J.E. Golay. The merit factor of long low autocorrelation binary sequences. *IEEE Trans. on Inform. Syst.*, 28:543, 1982.

[10] M.J.E. Golay and D. Harris. A new search for skewsymmetric binary sequences with optimal merit factors. Technical report, 1989.

[11] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, 1989.

[12] J.H. Holland. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor, 1975.

[13] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning. *Operations Research*, 37:865–892, 1989.

[14] S. Lin and B. W. Kernighan. An efficient heuristic for the traveling salesman problem. *Operations Research*, 21:298–516, 1973.

[15] B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithm. In H. Schaffer, editor, *3rd Int. Conf. on Genetic Algorithms*, pages 428–433. Morgan-Kaufmann, 1989.

[16] D. Moore. A round-robin parallel partitioning algorithm. Technical Report 88-916, Cornell University, 1988.

[17] H. Mühlenbein. Parallel genetic algorithm, population dynamics and combinatorial optimization. In H. Schaffer, editor, *3rd Int. Conf. on Genetic Algorithms*, pages 416–421, San Mateo, 1989. Morgan Kaufmann.

[18] H. Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337, San Mateo, 1991. Morgan-Kaufman.

[19] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. New solutions to the mapping problem of parallel systems - the evolution approach. *Parallel Computing*, 6:269–279, 1987.

[20] H. Mühlenbein, O. Krämer, G. Peise, and R. Rinn. The MEGAFRAME Hypercluster - a Reconfigurable Architecture for Massively Parallel Systems. In G. Wolff, editor, *Parcella'90*, pages 143–156. Akademie-Verlag, 1990.

[21] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17:619–632, 1991.

[22] Ch. C. Pettey and M. R. Leuze. A theoretical investigation of a parallel genetic algorithm. In H. Schaffer, editor, *3rd Int. Conf. on Genetic Algorithms*, pages 398–405. Morgan-Kaufmann, 1989.

[23] R. Tanese. Distributed genetic algorithm. In H. Schaffer, editor, *3rd Int. Conf. on Genetic Algorithms*, pages 434–440. Morgan-Kaufmann, 1989.

[24] G. von Laszewski. Ein paralleler genetischer Algorithmus für das Graph Partitionierungsproblem. Master's thesis, Universität Bonn, 1990.

[25] G. von Laszewski and H. Mühlenbein. A parallel genetic algorithm for the graph partitioning problem. In R. Maenner and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science 496, pages 165–169. Springer-Verlag, 1991.

[26] Q. Wang. Optimization by simulating molecular evolution. *Biol. Cybern.*, 57:95, 1987.