December 15, 1996

Experiences with Fine-Grained Parallel Genetic Algorithms

Udo Kohlmorgen[1], Hartmut Schmeck[1], and Knut Haase[2]

[1] *Institut für Angewandte Informatik und Formale Beschreibungsverfahren*
Universität Karlsruhe, D-76128 Karlsruhe
E-mail: {kohlmorgen, schmeck}@aifb.uni-karlsruhe.de

[2] *Institut für Betriebswirtschaftslehre*
Christian-Albrechts-Universität zu Kiel, D-24098 Kiel
E-mail: haase@bwl.uni-kiel.de

In this paper we present some results of our systematic studies of fine-grained parallel versions of the island model of genetic algorithms and of variants of the neighborhood model (also called diffusion model) on the massively parallel computer MasPar MP1 with 16k processing elements. These parallel genetic algorithms have been applied to a range of different problems (e.g. traveling salesperson, capacitated lot sizing, ressource constrained project scheduling, flow shop, and warehouse location problems) in order to obtain an empirical basis for statements on their optimization quality.

Keywords: fine-grained parallel genetic algorithm, island model, neighborhood model, combinatorial optimization

1 Introduction

Genetic and – more general – evolutionary algorithms are heuristic optimization methods based on the principle of natural evolution. Their universal applicability and their good performance on a variety of different optimization problems have led to a strong interest in this type of algorithm (see e.g. [13] and [21]). If the evolution of a large population of potential solutions is to be investigated for many generations, the computational requirements can be extremely large. Therefore, it is reasonable to use parallel computers to obtain satisfying results in a reasonable amount of time.

Fortunately, genetic algorithms are known to be inherently parallel, i.e. it should be easy to achieve a significant speedup by parallel execution. Nevertheless, there are many different ways of parallelizing genetic algorithms (cf. [9], [12], [14], [23], [26], and [31]). While usually, the parallel execution should preserve the functional behaviour of the sequential algorithm, the standard approaches to the parallelization of genetic algorithms lead to new algorithmic structures:

In the *island model* genetic algorithms are executed concurrently on several independent (sub-) populations with the added possibility of exchanging regularly good individuals between neighboring islands (cf. [19], [27] and [31]).

Besides this coarse-grained parallel approach the *neighborhood model* provides a fine-grained parallel variant: The population is distributed over the processors of a large mesh connected array. This spatial arrangement of the population allows the natural use of local neighborhoods in the selection of parents for producing new individuals for the next generation (cf. [6], [20] and [24]).

Thus, these parallel variants of genetic algorithms differ mainly with respect to their way of selecting parents for producing new individuals: While in the sequential version selection is done with respect to the whole population, the island model restricts selection to subpopulations which are disjoint except for the exchange of some good individuals, and the neighborhood model performs selection in extensively overlapping neighborhoods. In several applications these new algorithmic structures have shown good optimization behaviour.

In this paper we present some results of our intensive studies of several variants of these parallel genetic algorithms. In particular, we implemented the island model on a massively parallel MasPar MP1 machine having 16k processing elements. The islands and their neighborhoods are obtained naturally by distributing the population over the array (as in the neighborhood model) and by dividing the array into subarrays. In every island, the processing elements of the associated subarray can then be used for the efficient parallel execution of the genetic operators ([2], [11]). The same machine is used to implement the neighborhood model and to test the suitability of different neighborhoods obtained by considering neighbors at different distances in the eight possible directions of the MasPar's X-net.

The performance of these algorithms has been investigated with respect to several different optimization problems like traveling salesperson, capacitated lot sizing, ressource constrained project scheduling, flow shop, and warehouse location problems. We were especially interested, how the number and size of sub populations and the migration rate, intervall, and strategy (for the island model) and the selection strategy and neighborhoods (for the neighborhood model) influence the course of evolution and the quality of the generated solutions.

2 The Classical Genetic Algorithm

The classical genetic algorithm (cf. [13], [21]) operates on a set of N *individuals*, also called *population*. Every individial is a fixed length binary sequence (also called *chromosome*) representing a potential solution of the optimization problem. The *quality* of the individual is determined by the quality of this solution with respect to the problem's objective function. Quite often this quality is also called *fitness*, although – genetically – the fitness of an individual is a measure of its reproductive strength which can only be determined relative to a population. Based on their relative quality individuals are selected as potential parents to produce offspring for the next generation. The classical selection method corresponds to the use of a roulette wheel where the size of a sector depends on the relative quality of the corresponding individual. There is a variety of other selection methods e.g. based on rank instead of quality or selecting only the k best individuals for some k (also called *elitist* strategy).

Selected parents are mated randomly and produce one or two children by *crossover*,

i.e. by recombining genetic subsequences produced by cutting the chromosomes at a position chosen randomly. The newly formed individuals undergo *mutation*, usually changing the genetic information with some very low probability. A new population is formed by either replacing all the parents with their offspring or by replacing only those having a lower quality.

This process is repeated until there is no more improvement or until some other termination criterion is satisfied.

So, the classical genetic algorithm has the following structure:

```
GENERATE initial population
EVALUATE individuals
REPEAT
    SELECT parents
    RECOMBINE parents (by CROSSOVER)
    MUTATE offspring
    EVALUATE offspring and form new population
UNTIL termination condition satisfied
```

Genetic algorithms have been generalized in many ways, e.g. by allowing nonbinary genetic representations (e.g. using integral or real values) and – correspondingly – more complex recombination and mutation operators. Although these more general variants are sometimes called evolutionary algorithms or evolution programs (see e.g. [21]) we shall call them genetic algorithms, too, since their algorithmic structure is essentially the same.

3 Parallelizing Genetic Algorithms

Obviously, major components of genetic algorithms are inherently parallel: Evaluation, recombination, and mutation may be executed independently on the individuals of a population. If enough processors are available, these operations can be done in constant time, i.e. independent of the size of the population. But this is not true with respect to selection, since a global exchange of information is necessary to determine the relative quality or the rank of an individual. Nevertheless, there are several approaches to parallelizing genetic algorithms by localizing selection. These algorithms differ principally from the classical sequential genetic algorithm, but they seem to have even better optimization quality.

In this paper we shall not give a survey of these different approaches, but present only the variants that we implemented and investigated on the massively parallel MasPar MP1 machine, having 16k processors arranged as a 2-dimensional 128×128 torus with additional diagonal interconnections, i.e. every processor has 8 direct neighbors. This interconnection structure is also called X-net.

In all our variants of parallel genetic algorithms we consider the comparatively large population of 16,384 individuals, i.e. every individual is associated with a different processor.

3.1 Island Model

The standard computer architecture for implementing the island model would be a coarse-grained parallel machine, using one processor per island. Instead, we chose the massively parallel MasPar MP1. In our implementation , the population is divided into 1, 4, 16, 64, 256, or 1024 subpopulations by appropriately dividing the grid into subarrays. Therefore, we can easily measure the effect of the number of islands, while the total population size remains constant. In Fig. 1 four islands are sketched having 4096 individuals each. The division into subarrays naturally defines a neighborhood between islands, i.e. every island has four direct neighbors (diagonal interconnections are not considered for migration). The migration strategy varies between sending in one, two, three, or all four directions and between sending every 15, 30, or 50 generations. The migration rate is adjusted to the size of the islands by activating at migration time only 10% of the processors on the border to the neighboring islands.

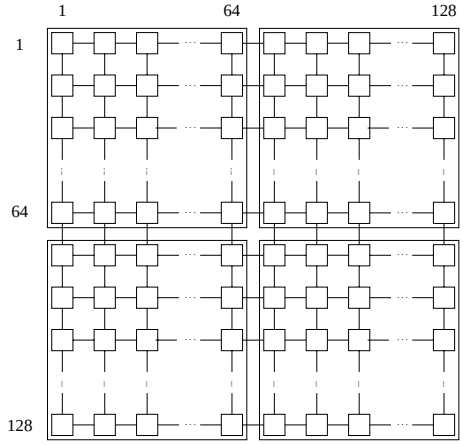

Figure 1: Four islands with 4096 individuals each

The subarrays can be used to execute all the genetic operators in parallel. Efficient algorithms for executing different types of selection on a 2-dimensional array can be found in [2]. In particular, selection can be performed on an $n \times n$ subarray in time $O(n)$ which is asymptotially optimal for this interconnection structure.

Two different mating schemes are used for generating a new population on each $n \times n$ subarray. In the classical scheme, 2 n^2 parents are selected and mated randomly to produce 2 n^2 children. Motivated by the array structure we also use a diallel mating scheme: In each $n \times n$ subarray only n parents are selected, either by quality based roulette wheel selection (with an appropriate scaling factor) or just the n best individuals. Afterwards, all possible pairwise crossings are used to produce 2 n^2 children. In both schemes, the children compete with their parents to be selected for the new population. The diallel mating scheme is well known in plant breeding but has rarely been used before in genetic algorithms. It has the advantage of exploiting the selected genetic material

more intensively than the classical scheme. A disadvantage could be the higher selection pressure.

Obviously, this fine-grained parallel implementation of the island model has several advantages over the standard coarse-grained implementations. Specifically, it is more flexible in adjusting the island structure and it leads to a much larger speedup by parallel execution. In every generation, only constant time is needed for crossover, mutation, and evaluation of individuals, i.e. the highest possible speedup has been achieved. The only step which can not be executed in constant time is the selection process, but still, optimal algorithms are used (cf. [2]). Moreover, the selection process is only global within the islands. Therefore, this problem gets less severe the more islands (and hence the smaller subpopulations) we use. Some more details of our investigation of this variant of the island model can be found in [10].

3.2 Neighborhood Model

In our implementation of the neighborhood model of genetic algorithms, every processor selects a partner for recombination from some local neighborhood by considering neighbors at different distances in the eight possible directions of the MasPar's X-net. Specifically, the following neighborhoods have been used and tested:

4-n	:	Horizontal and vertical neighbors.
5-n	:	Horizontal and vertical neighbors plus center.
8-n	:	All neighbors at distance 1.
9-n	:	All neighbors at distance 1 plus center.
16-n	:	In all 8 directions neighbors within distance 2.
17-n	:	In all 8 directions neighbors within distance 2 plus center.
24-n	:	In all 8 directions neighbors within distance 3.
25-n	:	In all 8 directions neighbors within distance 3 plus center.
(m)16-n	:	All neighbors at distance 1 plus horizontal and vertical neighbors at distance 3 plus diagonal neighbors at distance 8 (the latter are called "missionaries").
(m)17-n	:	All neighbors at distance 1 plus center plus horizontal and vertical neighbors at distance 3 plus diagonal neighbors at distance 8 (the latter are called "missionaries").

Figures 2 and 3 show the potential mating partners for neighborhoods 8-n and 16-n, respectively. For reproduction, every processor selects one individual from its neighborhood using one of the standard selection strategies (random, quality or rank based roulette wheel, or the best). In addition, a mixed strategy allows an arbitrary choice of the selection strategy at every processor. A detailed description of our investigation of this variant of the neighborhood model is given in [28].

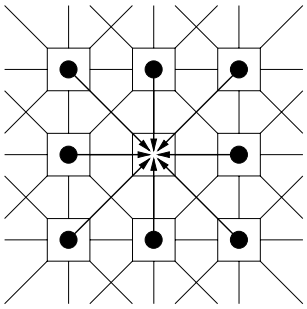

Figure 2: Mating partner selected out of 8 neighbors

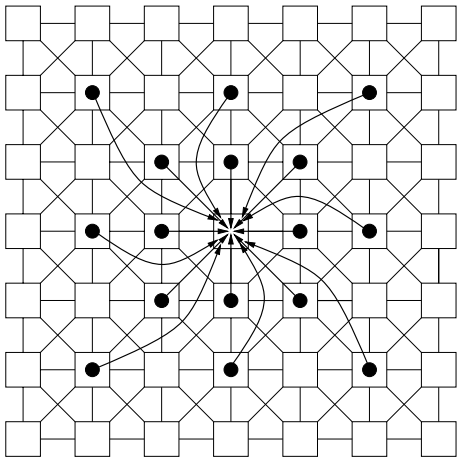

Figure 3: Mating partner selected out of 16 neighbors

4 Test Problems

Our implementations of parallel genetic algorithms have been applied to the optimization problems described below. We chose those problems, since for most of them benchmark instances are available. So we were able to check the results of our implementation of the genetic algorithm against the results computed by other approaches.

- *The Traveling Salesperson Problem (TSP)*: This is probably one of the most thoroughly investigated optimization problems and therefore suggests itself as a benchmark problem for testing the optimization quality of genetic algorithms. As genetic representation we used permutations of cities, i.e. a sequence of numbers.

Michalewicz [21] describes different crossover operators especially designed for permutation problems. We chose the edge recombination operator introduced by Whitley, Starkweather, and Fuquay (cf. [30]) which is especially well suited for the traveling salesperson problem.

From the database of benchmarks compiled by Reinelt [22] we selected problem instances ranging from 51 to 439 cities. Most of our test were conducted by using the 51 cities instance. This gave us the chance to run a large number of tests using a wide range of parameters.

- *The Resource Constrained Project Scheduling Problem (RCPSP)*: This is another complex problem having many different practical applications (see e.g. [8]): A project consisting of n tasks with precedence constraints has to be executed on k machines each with capacity constraints. Every task needs concurrently some capacity and some time on every machine. The goal is to find a minimum time schedule observing both types of constraints.

  Instead of a direct genetic representation as a sequence of tasks we used a representation consisting of a set of parameters (real values between 0 and 1) for a heuristic scheduling method. These parameters (one for each task to be scheduled) alter the values generated by a priority rule (like "latest starting time" or "latest finishing time"). This new approach introduces problem specific knowledge (the heuristic method) into the construction of the phenotype (the schedule). Since every string of real values between 0 and 1 is a valid genotype, we can use the standard genetic operators without having to deal with unfeasible solutions or with repair functions. Furthermore, we get a different fitness landscape which seems to be advantageous for the evolutionary optimization process.

  Recombination is done by standard two-point crossover and mutation changes values within a predefined small neighborhood of the alleles. We selected problem instances generated by the problem generator ProGen developed by Kolisch et al. [18] having in between 10 and 60 tasks on 4 machines. The problem instances with 60 tasks on 4 machines are the largest benchmark instances available. Optimal solutions are known only for the smaller problem instances.

- *The Uncapacitated Warehouse Location Problem*: The problem consists of selecting an optimal subset from a given list of warehouses such that the costs of building and maintaining the warehouses and the transport costs between selected warehouses and customers are minimized. We used a binary representation to indicate the selected locations. The recombination operator was a standard two-point crossover. From a set of benchmark problems given by Beasley [1] we chose the following instances: 50 locations / 50 customers, 100 locations / 500 customers, and 200 locations / 200 customers.

- *The Flow Shop Problem*: This is another classical optimization problem considered in operations research: n jobs have to be scheduled on a fixed sequence of m machines. The execution of job i on machine j takes time p_{ij} and the goal is to minimize the maximal total execution time of any job. Potential solutions are represented as sequences of job numbers. As for the TSP several recombination operators were tested. In the presented results we use the order crossover (cf. [21]).

The mutation operator exchanges arbitrary genes of the chromosome. For our test runs we chose a fairly complex problem instance of 50 jobs on 10 machines from a set of benchmark problems generated by Taillard [25].

- *The Capacitated Lot-Sizing Problem (CLSP)*: A number of J different items is to be manufactured on one machine (i.e. restricted by a single capacity constraint). The planning horizon is segmented into a finite number of T periods. In period $t \in \{1, ..., T\}$ the machine is available with C_t capacity units. Producing one unit of item j requires $p_j > 0$ capacity units. The demand for item j in period t, $d_{jt} \geq 0$, has to be satisfied without delay. Setting up the machine for item j causes setup cost $s_j > 0$. Setup costs occur for each lot produced in a period *(basic assumption)*. Holding cost $h_j \geq 0$ is incurred for the inventory of item j at the end of a period. The objective is to minimize the costs for setups and holding. As for the Resource Constrained Project Scheduling Problem potential solutions are genetically represented by a string of real valued parameters controlling a heuristic method for generating a feasible production schedule. The heuristic is an extension of an approach used by Haase [15]. For recombination we use standard two-point crossover. Mutation slightly changes the genetic values. We employed the well known 120 benchmark instances described in [4], where T as well as J range from 8 to 50.

5 Results

In this section we present some of our results of testing the different variants of parallel genetic algorithms on the problems described above. Specifically, so far, the island model has been tested systematically on the TSP and the RCPSP and the neighborhood model on the Warehouse Location Problem and the Flow Shop Problem.

Furthermore, different variants of the island model have been applied to a large number of problem instances of the resource-constrained project scheduling problem and one variant of the neighborhood model has been applied to a large number of problem instances of the CLSP.

5.1 Island Model

Figures 4 to 7 show some typical results of our investigation of the island model. In all of the test runs corresponding to these figures we used the diallel mating scheme. Fig. 4 and 5 refer to a TSP with 51 cities and Fig. 6 and 7 to an RCPSP with 60 tasks on 4 machines. The values are averaged over 4 independent runs of the genetic algorithm. Best results have been obtained whenever all 4 directions have been used for migration (see Fig. 4). Furthermore, Fig. 5 shows that a small migration intervall had positive influence on the optimization behaviour.

Both figures indicate that it seems to be advantageous to use 64 or 256 subpopulations. The use of only one large population always led to inferior results. The decrease in optimization quality for the largest numbers of islands is due to the fact that for 256 and 1024 subpopulations the genetic algorithm was stopped after 600 generations, whereas in the other cases the algorithm terminated before due to convergence.

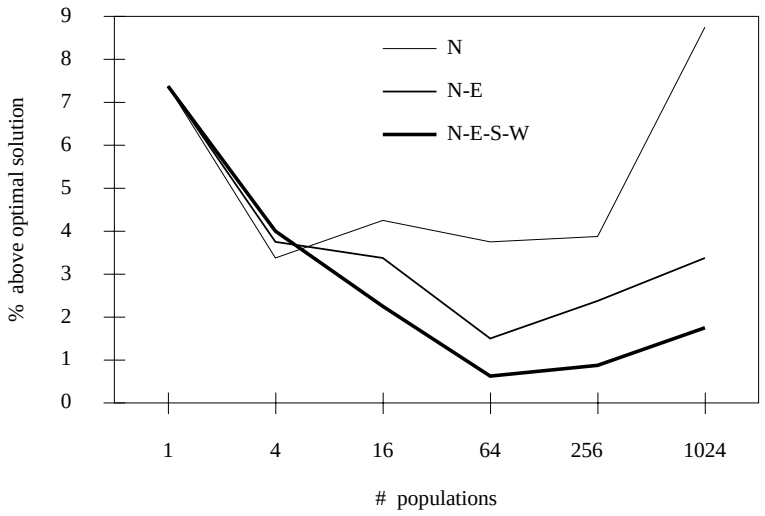

Figure 4: Influence of the direction of migration on optimization quality (for a TSP with 51 cities)

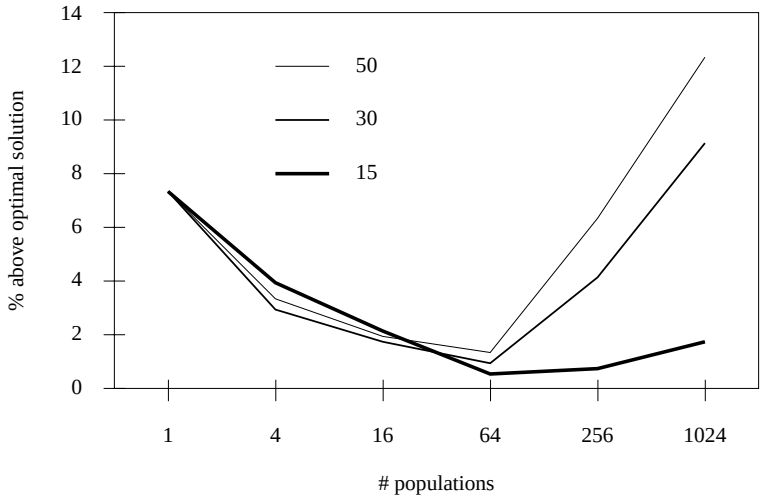

Figure 5: Influence of the migration intervall on optimization quality (for a TSP with 51 cities)

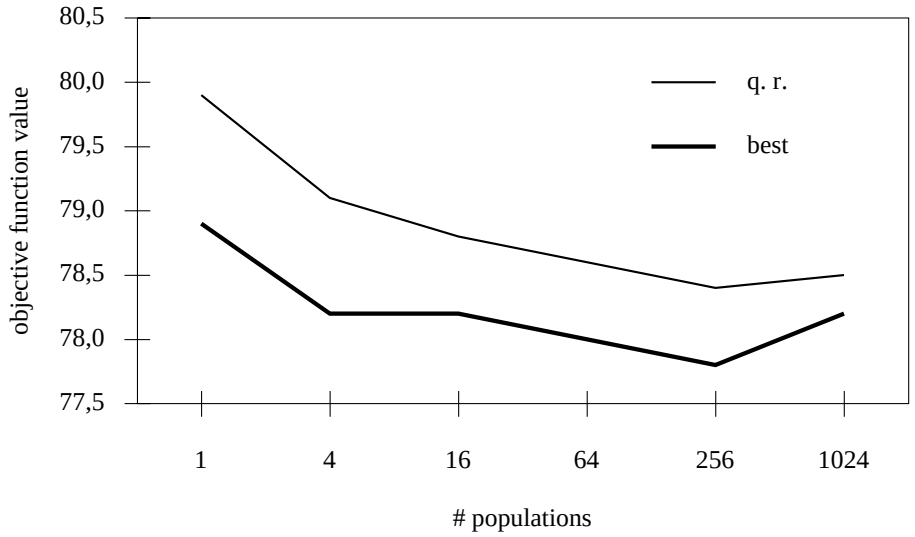

Figure 6: Comparison of selection strategies: Best versus quality based roulette wheel (for an RCPSP with 60 tasks on 4 machines)

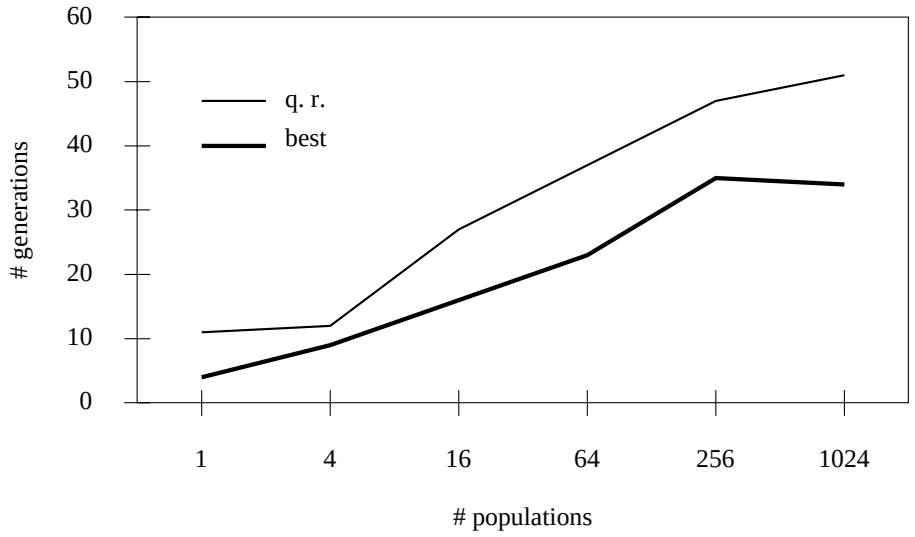

Figure 7: Influence of selection strategies on the number of generations (for an RCPSP with 60 tasks on 4 machines)

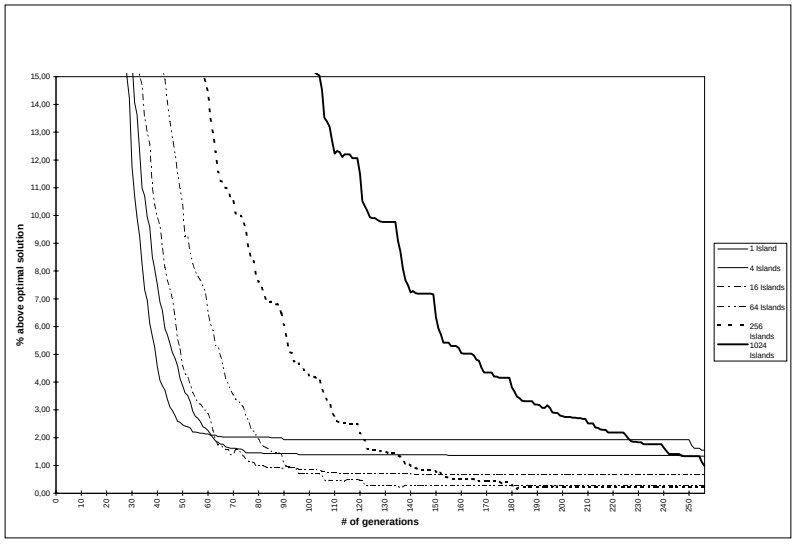

Figure 8: Evolution of fitness of the best individuals for variants of the island model (for a TSP with 51 cities)

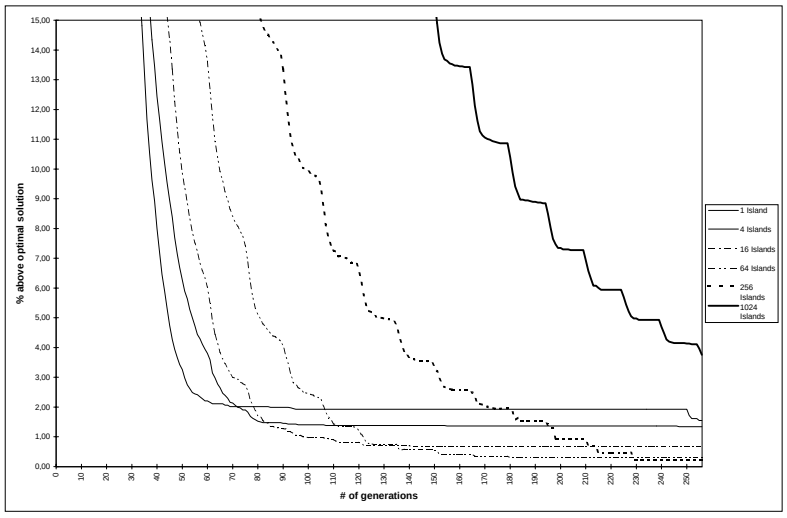

Figure 9: Evolution of average fitness for variants of the island model (for a TSP with 51 cities)

The comparison of the two selection strategies (Fig. 6) showed a clear advantage of the elitist (best) over the quality based roulette wheel selection. Again, the division of the population into 64 or 256 islands was favorable. However, as can be seen in Fig. 7, the division into many small subpopulations leads to a much larger number of generations before good results are obtained.

The influence of the number of islands on the optimization behaviour is illustrated in more detail in Fig. 8 and 9, referring again to the TSP with 51 cities, but this time using the classical mating scheme. Obviously, the rate of convergence decreases with the number of islands. The use of a medium number of islands seems to combine a rapid detection of relatively good solutions with a sufficiently broad exploration of the search space leading to high quality final solutions. For 1024 islands the rate of convergence is much slower than in all other cases. A comparison of the two figures shows for this case a large difference between the best and the average fitness in the population. This indicates a high remaining potential for further quality improvement. The figures also show very clearly the effect of migration: Every 15 generations there is a significant quality improvement.

Overall, one may conclude that the island model is well suited to observe the role of diversification and intensification in the evolutionary optimization process: For instance, each island might be seen as an intensification in a particular region and, hence, a great number of islands provides some diversification in the global process. That's probably why 64 and 256 islands tend to give the best results; here, intensification and diversification seem to be well balanced. For 1024 islands the population in each island might be too small. Furthermore, the elitist selection is a form of intensification while the roulette wheel selection allows more diversity. So, since the diversification is already provided by a large number of islands, the roulette wheel selection looses its main advantage and the elitist strategy performs better.

Since the main purpose of our experiments was a comparative evaluation of different parallel variants of genetic algorithms, we did not put too much effort in optimizing our algorithms for the particular test problems. Nevertheless, we got remarkably good results. In particular, for the 480 tested instances of the RCPSP with 60 tasks on 4 machines our best solution matched the previously best known result in 342 cases, and for 136 instances the previous upper bound was even improved.

5.2 Neighborhood Model

For the neighborhood model we tested the influence of the different neighborhoods and of the strategy for selecting the partner for reproduction on the optimization performance and on the number of generations. For the Warehouse Location Problem all test runs produced the same optimal solution. Therefore, in Fig. 11 we only give the number of generations needed to find this solution whereas for the Flow Shop Problem, the best objective function value is given (cf. Fig. 10).

Obviously, at least for our variants of genetic algorithms, the Flow Shop Problem turned out to be much harder than the Warehouse Location Problem. For the Flow Shop Problem the genetic algorithm improved slightly on the upper bound given by Taillard. For this complex problem the neighborhood had only moderate influence on the quality of the best solution, whereas the elitist selection strategy (called "best" in the figure) was clearly the best, especially for small neighborhoods.

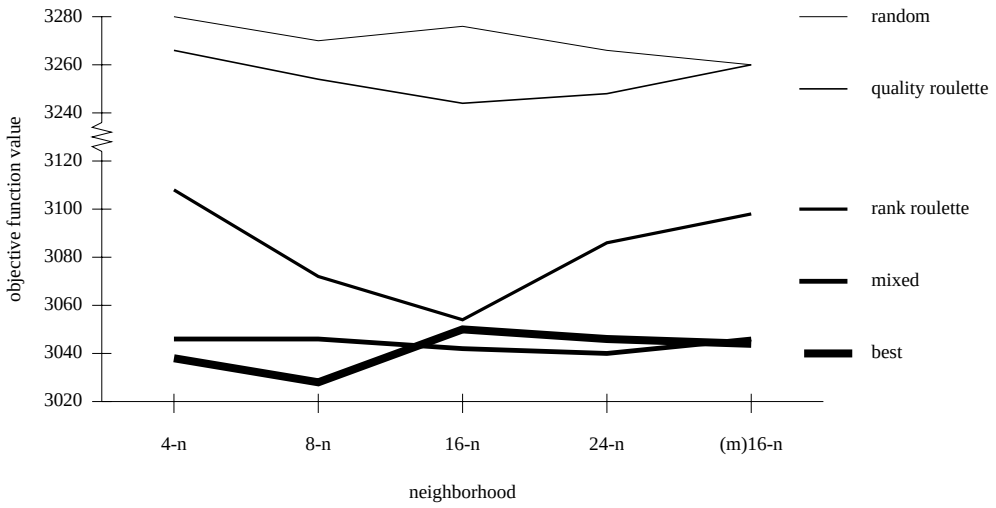

Figure 10: Influence of selection strategies and neighborhoods on optimization performance (for the Flow Shop Problem with 50 jobs on 10 machines)

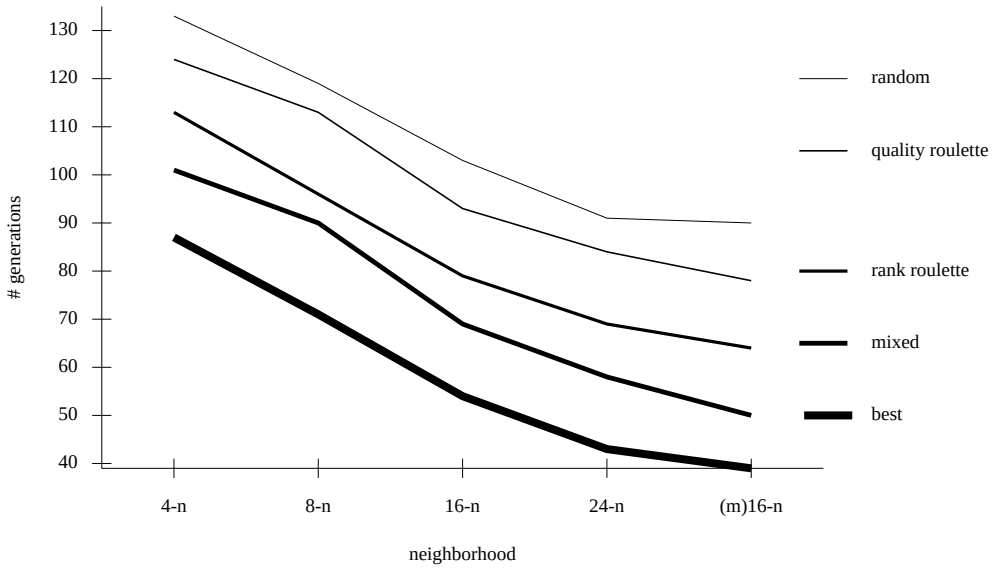

Figure 11: Influence of selection strategies and neighborhoods on the number of generations (for the Warehouse Location Problem with 200 locations and 200 customers)

For the Warehouse Location Problem, the elitist strategy was again the winner, but here, the larger neighborhoods had a clear advantage. Especially the neighborhoods with distant missionaries performed best under all selection strategies.

To further investigate the optimization potential of the neighborhood model on hard optimization problems we used it on the Capacitated Lot-Sizing Problem. Our empirical study [16] shows that the results obtained by the parallel genetic algorithm has the same solution quality as the state of the art algorithm from Kirca and Kökten [17], which outperforms the heuristics of Dixon and Silver [7] and Thizy and Van Wassenhove [29]. Some of these results are shown in Table 1. Z^* denotes the best result obtained by the three algorithms. The results indicate that our parallel genetic algorithm is superior for problems with 50 items, 8 periods and slightly better for problems with 8 items, 50 periods. For problems with 20 items, 20 periods Kirca and Kökten get better results. But on the average, our results in this category are only 1.44% behind.

Table 2 shows the number of problem instances for which each algorithm found the best result of all three algorithms.

As shown in [3], our massively parallel genetic algorithm is easily adapted to a slightly different CLSP with linked lot-sizes of adjacent periods where it outperformed all other known optimization methods by 5 to 20 percent.

Table 1: Computational results for the CLSP
(% deviation from the best solution Z^* found by DS, KK or PGA,
averaged over 40 instances per problem type)

	Dixon-Silver	Kirca-Kökten	parallel GA
50 items, 8 periods	1.29	0.65	0.17
20 items, 20 periods	7.55	0.06	1.50
8 items, 50 periods	9.57	0.99	0.76
total average	6.14	0.57	0.81

Table 2: Number of best results for the CLSP

	Dixon-Silver	Kirca-Kökten	parallel GA
50 items, 8 periods	4	12	24
20 items, 20 periods	0	37	3
8 items, 50 periods	0	19	21
total number	4	68	48

6 Conclusion

Our comparative evaluation of the performance of several fine-grained parallel genetic algorithms shows that they have clear advantages over the sequential version: In addition to achieving a large speedup by parallel execution, the fine-grained parallel genetic algorithms also show better optimization performance due to the larger genetic diversity obtained by dividing the population into a number of subpopulations.

The results presented in this paper do not allow a comparative evaluation of the island model and the neighborhood model. Nevertheless, the results of test runs of the island model with 64 populations on the Flow Shop Problem with 50 jobs on 10 machines indicate that the island model converges much earlier than the neighborhood model but it does not produce as good solutions. This is consistent with our statements on the optimization potential of the variant having 1024 islands which is relatively close to the neighborhood model.

In order to get more general statements on the quality of parallel genetic algorithms one should apply the different parallel models to other problem sizes and other types of problems. It is especially interesting to further compare the performance the fine-grained implementations of the island and the neighborhood model.

We believe that the good optimization performance of our genetic algorithms for the RCPSP and the CLSP is partially due to the genetic representation of potential solutions by a set of parameters controlling a heuristic method to produce feasible solutions. In this way the search process seems to be directed towards more promising regions of the search space. This effect will be subject of further studies.

Although all our experiments have been run on the MasPar MP1, our conclusions are not restricted to this type of architecture. In particular, the island model could as well be implemented on a coarse-grained parallel machine, but with the disadvantage of a much smaller speedup. The types of neighborhood we used are clearly influenced by the X-net interconnection structure of the MasPar MP1, but they could be implemented as well on other types of architectures. Our results clearly show that a large number of islands is advantageous as long as the size of the subpopulations is not too small. Therefore, in a coarse-grained implementation with only a few processors the number of islands should be higher than the number of processors, i.e. several islands should be simulated on one processor.

It might be advantageous to design new variants of the island model by either dividing the population into islands of different sizes or by adjusting the number and size of the islands dynamically in order to combine the advantages of high genetic diversity and rapid quality improvement.

Acknowledgements

We gratefully acknowledge valuable remarks and suggestions of anonymous referees.

References

[1] J.E. Beasley, An algorithm for solving large capacitated warehouse location problems, European Journal of Operational Research, 33 (1988) 314–325.

[2] J. Branke, H. C. Andersen, and H. Schmeck, Global selection methods for massively parallel computers, in *Proceedings of the AISB Workshop on Evolutionary Computing*, T. C. Fogarty ed., volume 1143 of Lecture Notes in Computer Science, Springer-Verlag, 1996, pp.175–188.

[3] J. Branke, U. Kohlmorgen, H. Schmeck, and H. Veith, Steuerung einer Heuristik zur Losgrößenplanung unter Kapazitätsbeschränkungen mit Hilfe eines parallelen genetischen Algorithmus, in *Proceedings Workshop Evolutionäre Algorithmen in Management-Anwendungen*, J. Kuhl, V. Nissen eds., Göttingen, 1995, pp.21–31.

[4] D. Cattrysse, J. Maes, and L.N. van Wassenhove, Set partitioning and column generation heuristics for capacitated lotsizing, European Journal of Operational Research, 46 (1990) 38–47.

[5] J. P. Cohoon, S. U. Hedge, W. N. Martin, and D. Richards, Punctuated equilibria: a parallel genetic algorithm, in *Proceedings of the Second International Conference on Genetic Algorithms*, J. J. Grefenstette ed., Lawrence Erlbaum Associates, 1987, pp.148–154.

[6] R. J. Collins and D. R. Jefferson, Selection in massively parallel genetic algorithms, in *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. K. Belew and L. B. Booker eds., Morgan Kaufmann, San Diego CA, 1991, pp.244–248.

[7] P.S. Dixon and E.A. Silver, A heuristic solution procedure for multi-item single-level, limited capacity, lot-sizing problem, Journal of Operations Management, (1981) 23–39.

[8] W. Domschke and A. Drexl, *Einführung in Operations Research*, Springer-Verlag, Berlin, Heidelberg, 1991.

[9] D. Duvivier and P. Preux and E. G. Talbi, Parallel genetic algorithms for optimization and application to NP-complete problem solving in *Int. Workshop on Combinatorics and Computer Science*, Brest, France, 1995

[10] D. Eichberg, *Untersuchung des Insel-Modells Genetischer Algorithmen auf einem massiv parallelen Rechner*, Diplomarbeit, Institut AIFB, Universität Karlsruhe, 1996.

[11] D. Eichberg, U. Kohlmorgen, and H. Schmeck, Feinkörnig parallele Varianten des Insel-Modells Genetischer Algorithmen, in *Mitteilungen - Gesellschaft für Informatik e.V., Parallel-Algorithmen und Rechnerstrukturen*, PARS-Workshop, Stuttgart, Oct. 9-11, 1995, pp.74–80.

[12] T. Fogarty, Implementing the genetic algorithm on transputer based parallel processing systems, in H.P. Schwefel and R. Männer eds., *Parallel Problem Solving from Nature - PPSN I*, volume 496 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1991, pp. 145–149.

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

[14] M. Gorges-Schleuter, Explicit parallelism of genetic algorithms through population structures, in *Parallel Problem Solving from Nature* Schwefel and Männer eds., Springer-Verlag, Berlin, 1991, pp.150–159.

[15] K. Haase, Capacitated lot-sizing with linked production quantities of adjacent peroids, Technical Report No. 334, Institut für Betriebswirtschaftslehre, Universität Kiel, 1994.

[16] K. Haase and U. Kohlmorgen, Parallel genetic algorithm for the capacitated lot-sizing problem, in *Operations Research Proceedings 1995*, P. Kleinschmidt et al. eds., Springer-Verlag, Berlin, 1996, pp.370–375.

[17] Ö. Kirca and M. Kökten, A new heuristic approach for the mult-item dynamic lot sizing problem, European Journal of Operational Research, 75 (1994) 332–341.

[18] R. Kolisch, A. Sprecher, and A. Drexl, Characterization and generation of a general class of ressource constrained project scheduling problems, Management Science, Vol. 41, No. 11

(1995).

[19] B. Kröger, P. Schwenderling, and O. Vornberger, Parallel genetic packing on transputers, in *Parallel Genetic Algorithms: Theory & Applications*, J. Stender ed., IOS Press, Amsterdam, 1993, pp.151–185.

[20] B. Manderick and P. Spiessens, Fine-grained parallel genetic algorithms, in *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer ed., Morgan Kaufmann, San Mateo CA, 1989, pp.428–433.

[21] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin, 1996.

[22] G. Reinelt, TSPLIB - a traveling salesman problem library, ORSA Journal on Computing, Vol. 3, No. 4(1991) 376–384.

[23] M. Schwehm, Implementation of genetic algorithms on various interconnection networks, in *Proceedings of the International Conference PACTA*, M. Valero et al. eds., IOS Press/CIMNE, 1992, pp.195–203.

[24] M. Schwehm, Th. Oparterny, and K.-H. Kirsch, Plazierung von Makrozellen durch genetische Algorithmen auf verteilten und massiv parallelen Rechnern, in *Mitteilungen - Gesellschaft für Informatik e.V., Parallel-Algorithmen und Rechnerstrukturen*, Workshop 1994, 1995, pp.69–74.

[25] E. Taillard, Benchmarks for the basic scheduling problems, European Journal of Operational Research, 64 (1993) 278–285.

[26] E. G. Talbi and P. Bessire and J. M. Ahuactzin and E. Mazer, Parallel cooperating genetic algorithms in *Practical Handbook of Genetic Algorithms: New Frontiers* L. Chambers ed., CRC Press, 1995, pp.93-109.

[27] R. Tanese, Distributed genetic algorithms, in *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer ed., Morgan Kaufmann, San Mateo CA, (1989), pp.434–439.

[28] U. Tempel, *Vergleich lokaler Selektionsstratgien für feinkörnig parallele genetische Algorithmen zur Lösung von schweren Optimierungsproblemen*, Diplomarbeit, Institut AIFB, Universität Karlsruhe, 1995.

[29] J.M. Thizy and L.N. Van Wassenhove, Lagrangean relaxation for the multi-item capacitated lot-sizing problem: a heuristic implementation, IIE Transactions, 17 (1985) 308–313.

[30] L.D. Whitley and T. Starkweather and D'Ann Fuquay, Scheduling problems and traveling salesman: the genetic edge recombination, in *Proceedings of the Third International Conference on Genetic Algorithms*, J.D. Schaffer ed., Morgan Kaufmann, San Mateo CA, 1989, pp.133–140.

[31] L. D. Whitley and T. Starkweather, Genitor II: a distributed genetic algorithm, Expt. Theor. Artif. Intell., 2 (1990) 189–214.