Evolution Strategies for a Parallel Multi-Objective Genetic Algorithm

Ricardo Szmit
Institute of Computer Science
Hebrew University of Jerusalem
Jerusalem 91904, Israel
rszmit@cs.huji.ac.il

Amnon Barak
Institute of Computer Science
Hebrew University of Jerusalem
Jerusalem 91904, Israel
amnon@cs.huji.ac.il

Abstract

This paper compares evolution strategies for a parallel multi-objective genetic algorithm adopting the concept of Pareto optimality. This algorithm was applied to the solution of a set of process scheduling problems that are part of a standard scheduling benchmark. Our main goal was to compare the *efficiency* and the *efficacy* of the evolution strategies, and how they relate to attributes of the problem. In order to quantify the quality of populations produced by the algorithm, we measured the *coverage* of the solution space and the *proximity* to the Pareto-optimal front. Our results show that an evolution strategy using heterogeneous subpopulations with restart is consistently superior to traditional strategies, without being more expensive. We also observe that the performance of the algorithm is directly related to the problem's communication to computation ratio (CCR). Our approach is based on the division of the scheduling problem to two parts that can be solved independently. This division allows a simpler encoding of individuals, so that the crossover and mutation operations can be implemented more efficiently. Thanks to the combination of genetic search with proven heuristics, this gain in efficiency does not imply a loss of efficacy.

Paper Category: Genetic Algorithms

1 Introduction

Most optimization problems are in reality multi-objective problems, in the sense that there is more than one way to measure the quality of a given solution. Early works on the field of multi-objective optimization tried to combine different quality measures using arbitrary functions. This approach, popularly referred to as "*comparing apples to oranges*", never generated satisfactory results. More recent research recognized the advantages of the Pareto-optimal approach [15].

Given an optimization problem with objective functions $f_1 ... f_n$, in which the value of each function must be minimized, the quality of a solution S is represented by a vector $V = (f_1(S), ..., f_n(S))$. A solution S_1 dominates another solution S_2 if, for each objective function f_i, we have $f_i(S_1) <= f_i(S_2)$, and there is at least one objective function f_j such that $f_j(S_1) < f_j(S_2)$. A solution is non-dominated if there is no solution that dominates it. The Pareto-optimal (P-optimal) front is the set of all non-dominated solutions. Clearly, the P-optimal front dominates all other possible solutions.

1.1 The Solution Space

In most real multi-objective problems, the different objective functions are not completely independent. In this case, each pair of objective functions may be either directly related or inversely related. The kind of relation between the diverse objective functions determines important attributes of the solution space. This can be clearly observed when considering a pair of functions.

In the case of functions that are directly related, the solution space looks like the one in Figure 1. This example shows Schaffer's classic function $F2$ [14], defined as $x = t^2$ and $y = (t-2)^2$. In this example, the P-optimal front is the curve between the points $(0, 4)$ and $(4, 0)$.

In the case of functions that are inversely related, the solution space looks like the one in Figure 2. This example shows the function $y = 1/x$, for $x > 0$. In

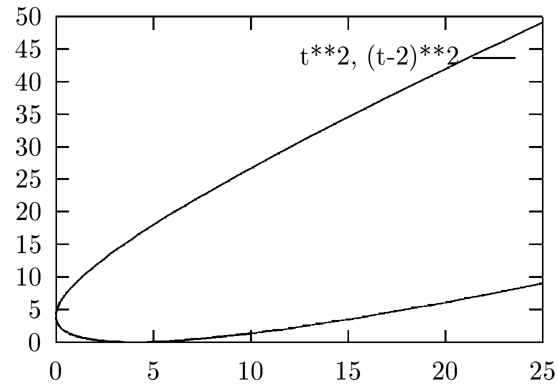

Figure 1: Directly related functions

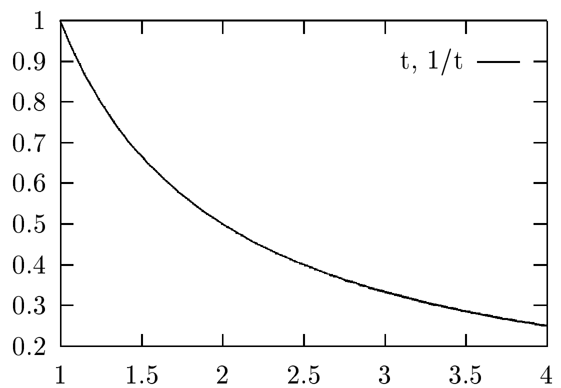

Figure 2: Inversely related functions

this example, all points in the curve belong to the P-optimal front.

In more interesting examples, the solution space would consist of a large set of points contained inside a curve. The format of this curve will be similar to one of the above examples, according to the relation between the objective functions.

Besides the shape of the solution space, another important attribute is the density of points inside the curve. This density may vary greatly, if there is a large number of solutions concentrated on a few small regions.

1.2 A Pareto-oriented Genetic Algorithm

A genetic algorithm (GA) based on the concept of Pareto-optimality uses a non-dominated sorting procedure [7]. Given a population which is a subset of the solution space, this procedure identifies all non-dominated individuals and assigns to them the highest rank. This process is repeated for the remaining individuals until the complete population is sorted. Then, a new population is generated using crossover, through a selection process in which individuals with higher ranks have also a higher probability to generate descendents.

Using this simple GA, the number of generations required to find the Pareto front depends on the population size. In general, the bigger the population, the smaller the number of required generations. We would like to develop evolution strategies that achieve good results with smaller populations. This can be done by driving the simple GA to explore new, potentially good regions of the solution space. Our knowledge about the attributes of the solution space may be useful to identify such regions.

The simple GA may be driven to explore new regions by the insertion of individuals that belong to these regions. The main question in this case is how to find these individuals. Our hypothesis is that these individuals can be obtained by the use of a parallel GA [2], in which each subpopulation evolves according to a different criterion. In the case of a multi-objective problem, there may be a separated subpopulation trying to optimize each objective function. An additional Pareto-oriented population would receive immigrants from all others, and thus explore several interesting regions.

1.3 Efficiency and Efficacy

We try to validate our hypothesis through the comparison of different evolution strategies. In order to compare them, we need the definition of quality attributes. The *efficiency* of an evolution strategy can be measured as the number of generations required to find a P-optimal solution. The *efficacy* of an evolution strategy can be measured as its ability to find a large number of points belonging to the P-optimal front. In general, there is a trade-off between efficiency and efficacy. For example, the use of Niching [8] improves the efficacy but may reduce the efficiency of a GA.

This paper is organized as follows: Section 2 describes the *scheduling problem* we are interested to solve, and how the solutions are represented as individuals in the genetic algorithm. Section 3 presents the *evolution strategies* used by the genetic algorithm, and the criteria for comparing these strategies. Section 4 contains an *analysis of results*, illustrating the performance of the diverse strategies and how this performance is affected by attributes of the problem. In section 5 we suggest some directions for *future research*, and in section 6 we bring our main *conclusions*.

2 The Scheduling Problem

The scheduling problem is characterized by a set of intercommunicating processes that must be executed on a parallel system. For our purposes we assume that this parallel system is composed of a set of homogeneous processors that are completely connected, and that the communication links that connect each pair of processors have equal latency and bandwidth. Each process has a duration time, which is the time required for its execution on any processor. A process is executed serially, without preemption. The inter-process communication is done through messages. Each message has a source and a destination process, and a duration time, which is the time required to transmit the message over a communication link. If the source and destination processes are located in the same processor, the message is delivered instantly. After a process finishes its part of the computation, it may send messages to several dependent processes. A process that depends on messages sent by other processes can start its computation only after it receives all the messages it is waiting for. A process that does not depend on any previous process is called a root process. A process that does not send messages to any other process is called a leaf process.

The scheduling problem can be represented as a weighted Directed Acyclic Graph (DAG), in which the nodes represent processes and the edges represent messages. The weight of nodes and edges represent the duration time of their respective processes and messages. A schedule is composed of an assignment of processes to processors and an ordering of execution among processes. The dependencies in the DAG already define a partial ordering. A complete ordering is necessary to determine which process has priority of execution, if two or more processes are ready for execution at the same processor when it becomes available. The total execution time of a schedule is the time required for the execution of all processes, from the beginning of the first root process to the end of the last leaf process. Other attributes of a schedule are the number of processors used, which can be less than the total number of processors in the system, and the total weight of messages sent. In general these are conflicting attributes, in the sense that one can not be optimized without affecting the others.

In the past, scheduling was approached as a single-objective problem, in which the function to optimize is the total execution time. In this work we are interested in scheduling as a multi-objective problem, trying to optimize also other attributes. When considered a single-objective problem, scheduling in its general form is known to be NP hard [1]. Clearly, treating it as a multi-objective problem does not make it simpler. The single-objective version was extensively studied in the last decades [4], focusing on very particular cases. For several of these special cases there are known polynomial-time heuristics, and we can not expect genetic algorithms to be a competitive technique. On the other hand, for the more general problems, that were proved to be intractable, and for the multi-objective version, genetic algorithms are a powerful tool.

Even for particular problems in which one objective function is clearly more important than others, a multi-objective genetic algorithm may be useful to avoid the trap of local optima. It is known that the general scheduling problem's solution space is characterized by widely dispersed local optima [12]. The maintenance of a broad Pareto front for each generation assures that genetic diversity is preserved, and that there will always be potential paths of evolution until the P-optimal front is found.

2.1 Schedule Representation

As seen before, a schedule of processes on several processors is composed of two parts: an assignment of processes to processors and an ordering of processes. These two components are orthogonal, since the same assignment may have several possible orderings, and a particular process ordering may be applied to any assignment.

Most works that tried to solve scheduling problems through genetic algorithms encoded in each individual both the the assignment and the ordering of processes [13]. This approach has several drawbacks that affect the performance of the genetic algorithm. The schedule can not be represented as a simple sequence of genes, since it must have two dimensions [5] [9] [16]. This causes the crossover and mutation operations to be much more complex and time consuming. Even the generation of the initial population is quite complicated, requiring the validation of each individual. The search space is large, since for every possible assignment we must consider all possible orderings. If B is the number of processors in the system and N is the number of processes in the DAG, the size of the search space is of the order of $O((B^N) * N!)$, where B^N is the number of possible assignments and $N!$ is the number of possible orderings. The exact size of the search space is hard to quantify, because valid orderings must respect the precedence relations between processes.

The traditional research on scheduling has focused much more in heuristics for the definition of good orderings than in finding good assignments [10]. Most of these heuristics actually define tie-braking rules to determine which process has priority of execution. For example, higher priority can be given to processes that have more successors, or to processes that are closer to the roots of the DAG [6]. Some of these heuristics were proven to produce optimal schedules (in relation to execution time) for very special cases, which are often far from real situations. However, these heuristics produce good schedules also for the general case, since the ordering they define is clearly superior than most of the $N!$ possible orderings. Hence, by using a fixed heuristic to produce good orderings, it is possible to concentrate the search efforts in the process assignment space.

A process assignment may be simply encoded as a sequence of N digits in base B. For example, the assignment of 32 processes to 2 processors can be represent as a 32-bit word. Using this encoding, any possible sequence of digits represents a valid assignment. The operations of crossover and mutation are trivial, allowing very efficient implementations without any need for validation. Additionally, the search space is greatly reduced: its size is exactly B^N. The great simplification of the evolution operations and reduction in the search space clearly improve the efficiency of the genetic algorithm. Thanks to the use of proven heuristics to determine the ordering of processes, this efficiency gain does not imply a loss of efficacy. In other words, it is possible to find better solutions in a shorter time.

3 Evolution Strategies

We tested two versions of the algorithm, using heterogeneous and homogeneous populations. In both versions there are two isolated subpopulations that send immigrants to a third main population. In the version with heterogeneous populations (HT), the two subpopulations evolve using a single objective function and the main population is Pareto-oriented, evolving non-dominated solutions and taking in consideration both objective functions. In the version with homogeneous populations (HM), the three populations are Pareto-oriented. The flow of immigrants is unidirectional, from the subpopulation to the main one, to preserve a higher level of genetic diversity in the total population. Thus, the terms homogeneous and heterogeneous are used in relation to the evolution strategy adopted by each population, and not in relation to the diversity of their individuals.

Our hypothesis is that in HT the immigrants can have an important role on the evolution process, driving the main population to explore new regions in the solution space. To verify it, HM is used as a control group, with identical values for all other parameters of the genetic algorithm. It must be noticed that HM represents also the traditional strategy adopted by Pareto-oriented GAs [3].

For scheduling problems, the two objective functions are total *execution time* and total *messages weight*. Both objective functions must be minimized. These functions are computed as follows:

The execution time depends on both the assignment and the ordering of processes. The assignment of processes to processors is determined by each individual's genes. The ordering of processes is determined by a heuristic which gives higher priority to processes that have been waiting longer for execution.

A FIFO execution queue is maintained for each processor. A process enters the execution queue of the processor in which it was assigned at the moment it becomes ready for execution. A process is ready for execution when all the processes that precede it finished their execution and all the messages they sent already arrived. In the beginning, all root processes are inserted in the respective processor's queues, with higher priority to processes with more successors. The total execution time is then computed through an iteration over the processors' queues until all processes are finished. We note that this ordering heuristic is similar to the one that gives higher priority to processes closer to the roots of the DAG.

In order to compute the messages weight it is sufficient to know the assignment of processes to processors. This computation is trivial: A message is sent if the sender and receiver processes were assigned to different processors. In this case, the weight of this message must be added to the total messages weight.

3.1 Proximity and Coverage

For the comparison of the heterogeneous and homogeneous variants of the algorithm, we use two quality attributes: *proximity* to the P-optimal front and *coverage* of the population.

In most situations the P-optimal front is not known in advance, thus it is not possible to measure the absolute proximity. However, it is sufficient to measure the relative proximity, which is determined by the distance between two fronts. For a given region in the solution space, if the points in front A dominate the points in front B, then clearly A is closer to the P-optimal front.

Coverage may be defined as the percentage of the solution space that is dominated by a given front. Again, since it is not possible to know in advance the area

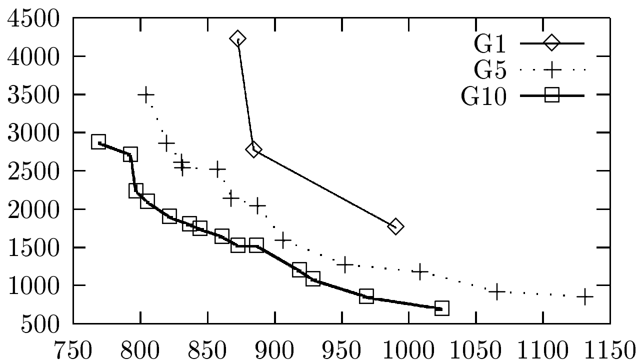

Figure 3: Evolution of Pareto fronts for HT

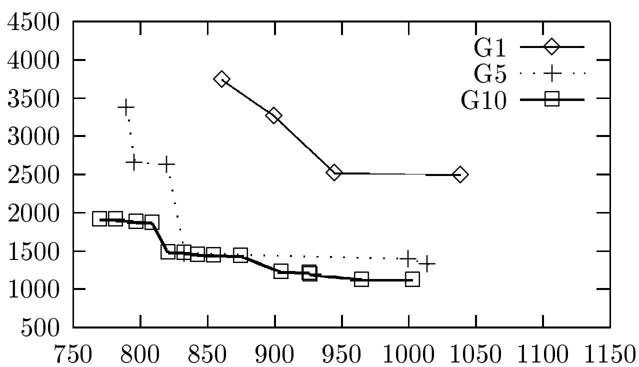

Figure 4: Evolution of Pareto fronts for HM

of the solution space and neither the density of the population in different regions, an exact value for the coverage can not be computed. However, it is possible to obtain an estimate using a Monte Carlo method: Generating a large number of random solutions and verifying the percentage of dominated ones.

4 Analysis of Results

Figures 3 and 4 show the evolution of the Pareto front, during the first 10 generations, for the heterogeneous (HT) and homogeneous (HM) versions of the algorithm. The X axis is the *execution time* and the Y axis is the *messages weight*. In this case, the problem to be solved has 32 processes and 165 messages, and the system has 2 processors. Of course, each execution of the algorithm produces reasonably different results, specially in the first generations, but these examples illustrate some typical characteristics of the evolution process. The HT results show that the immigrants from subpopulations working on a single objective contribute to stretch the Pareto front, increasing rapidly the coverage of the solution space. However, this causes the evolution to become slower, since the search efforts are distributed among a larger number of points in the front. The use of homogeneous populations causes the Pareto front to evolve faster towards the optimal front, but its grow is slower, so that the coverage does not increase so fast. Hence, these results indicate that there is a trade-off between proximity and coverage.

After observing the evolution process for a larger number of generations, it becomes clear that the heterogeneous version suffers from a serious problem: The subpopulations working on a single objective tend to converge much faster than the Pareto-oriented population. After they converge, their immigrants stop contributing to the evolution of the Pareto front. To avoid this problem, we adopted a restart mechanism [11] that creates a completely new subpopulation when appropriate. The good solutions found by the previous subpopulation are still preserved in the main population. A restart is done when the relative difference between the best and the average fitness in the subpopulation is below a given threshold. In our tests, we adopted the value 0.001, or 0.1%, as this threshold.

The next experiments compare the performance of three versions of the algorithm: Homogeneous populations (HM), heterogeneous populations without restart (HT), and heterogeneous populations with restart (HTR).

In these three versions, all parameters are identical: The two subpopulations and the main population have 100 individuals each, so the total population size is 300. For each new generation, the 20% best individuals of each population are preserved and participate in crossover with equal probability. After crossover, the 10% best individuals of each subpopulation are sent as immigrants to the main population. The new immigrants replace the worse individuals of the main population. Crossover is done in a single random point, with mutation probability 0.01, or 1%. These are arbitrary values, and we did not try to optimize them, since our main goal is to compare evolution strategies. However, it must be noticed that the population is small relatively to the problems' size.

4.1 Benchmark Results

We compared the performance of the three different versions using a set of DAGs which are part of a standard benchmark for scheduling problems [10]. This is a set of 36 different DAGs classified according to their communication to computation ratio (CCR). The C-

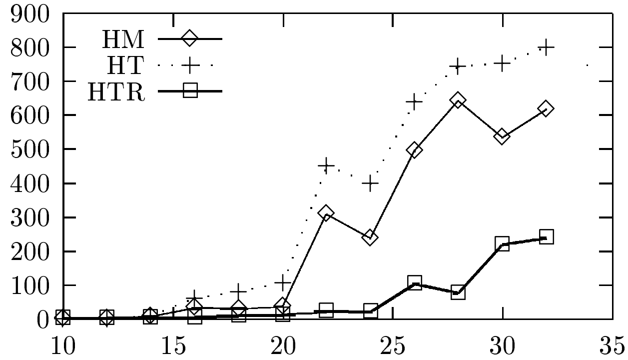

Figure 5: Generations to find optimal, CCR=0.1

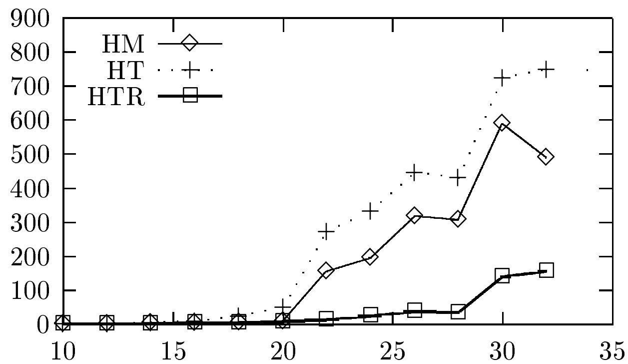

Figure 6: Generations to find optimal, CCR=1.0

CR is computed as the total weight of messages divided by the total weight of processes, and thus is a static problem attribute. There are three groups of 12 DAGs each, with CCR values of 0.1, 1 and 10, and problems varying in size from 10 to 32 processes. The CCR is an important attribute, which affects the characteristics of the solution space. In problems with small CCR, the total execution time and the total messages weight are inversely related, while in problems with large CCR they are directly related.

We know the optimal execution time for these DAGs on a system with 2 processors, so our tests consisted of running the GA until a solution having this time is found. This solution does not necessarily belong to the P-optimal front, since it may be dominated by another solution with the same execution time and lower messages weight. However, for our purposes, finding such a solution is a good termination criterion, because it indicates that the current Pareto front is close enough to the optimal one.

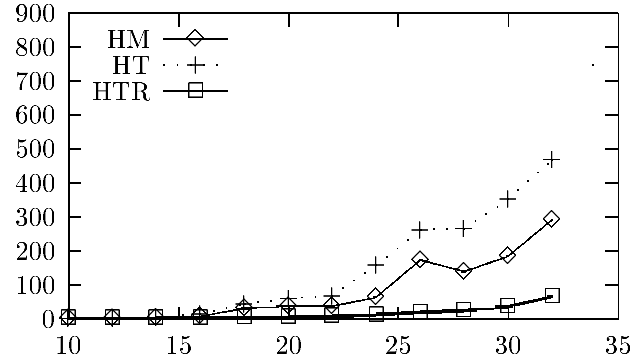

Figure 7: Generations to find optimal, CCR=10.0

Figures 5, 6 and 7 show the number of generations required by the different versions of the algorithm to find the optimal solution. The X axis is the *problem size* (number of processes), and the Y axis is the *number of generations*. The values presented in these figures are the average of a thousand executions of each version of the algorithm for each problem. In each execution, the GA runs at most a thousand generations, and stops as soon as it finds a solution with optimal execution time.

These results show that the version using heterogeneous populations with restart always outperforms the other two, in the sense that it requires less generations to find a solution with optimal execution time. In particular, HTR seems to be more scalable than the other strategies.

It is interesting to note that, in all three versions of the algorithm, when the CCR increases the number of required generations decreases. This indicates that, for these strategies, scheduling problems with low CCR are harder to solve. As observed before, in problems with high CCR the two objective functions are directly related. As a consequence, the size of the Pareto front becomes smaller with each generation, and the evolution tends to be faster. We believe that this is a general characteristic of problems in which the different objective functions are directly related. This is a potential topic for future research.

Figure 8 shows the relation between the number of restarts required by HTR and the CCR of the problem. The X axis is the *problem size* and the Y axis is the *number of restarts*. When comparing this graph to the previous ones, we observe that the ratio between the number of generations and the number of restarts required by HTR is almost constant, i.e., it does not depend on the value of the CCR. This indicates that populations that evolve according to a single-objective

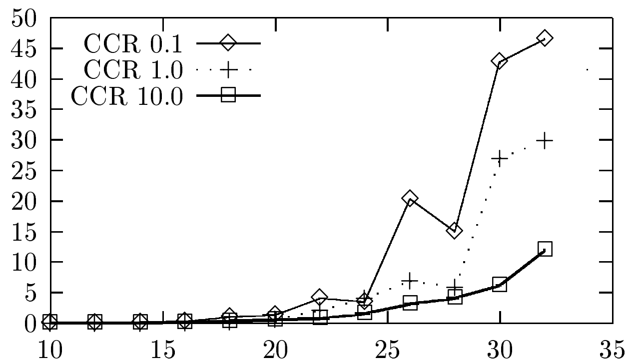

Figure 8: Number of restarts for different CCRs

function are not affected by the CCR in the same way as Pareto-oriented populations. Further research is required for a better understanding of the properties of the search space in both cases.

As a general conclusion, for problems in which there is an inverse relation between the different objective functions the HTR strategy is more *effective*, since the immigrants contribute to increase the *coverage* of the Pareto front. In contrast, for problems in which there is a direct relation between the different objective functions the HTR strategy is more *efficient*, since the immigrants contribute to improve the *proximity* to the P-optimal front.

4.2 Comparison to Heuristics

During our experiments, we observed that in several cases our multi-objective GA was able to find optimal solutions that could not be found by traditional scheduling heuristics. Table 1 is based on values provided by Kwok and Ahmad [10] for the set of 36 problems discussed above. They implemented 11 different heuristic algorithms that are documented in the scheduling literature, and tested them for each problem in the set. The table shows, for each problem, characterized by size and CCR, the number of heuristic algorithms that were able to find the optimal execution time.

Table 1 provides interesting information, specially when compared to our results. It is clear that the performance of the heuristics is related more to the structure of the problem than to its size. Some small problems could not be solved by any heuristic. When the CCR increases, the efficacy of the heuristics decreases, which is the opposite behavior of our GA. For almost half of the problems (16 out of 36), no heuristic

Problem Size	CCR 0.1	CCR 1.0	CCR 10.0
10	7	7	0
12	1	0	3
14	0	4	2
16	7	0	3
18	0	4	0
20	0	0	3
22	3	3	0
24	0	0	1
26	5	1	0
28	4	4	0
30	3	3	0
32	0	0	3

Table 1: Number of heuristics that found the optimal execution time (out of 11)

was able to find the optimal execution time. In average, the heuristics were able to solve less than 20% of the problems. In contrast, our HTR strategy was always able to find the optimal solution for each of these problems. As shown in figures 5, 6 and 7, the optimal solution is found in less than 300, 200 and 100 generations for problems with CCR value of 0.1, 1.0 and 10.0, respectively. It must be noticed also that several of the optimal solutions provided by greedy heuristics actually use more than two processors.

5 Future Research

As observed in the previous section, the performance of our HTR strategy is directly related to the problem's CCR. This may be an important issue for further research: Since the CCR is easy to compute, it can be used to develop adaptable GAs that select evolution parameters, such as the population size, dynamically.

In all our tests, we used a single heuristic to determine the priorities of processes. It would be interesting to have a pool of several good heuristics available. Each individual in the population could choose from this pool the heuristic that provides the best result for its particular process assignment. This technique may increase considerably the probability that the optimal solution is actually found by the GA, and its implementation is simple.

It would be also interesting to apply our HTR strategy to solve other kinds of multi-objective problems. In particular, a topic that needs more research is how the properties of the solution space are affected by the kind of relation (direct or inverse) between the different objective functions.

6 Conclusions

We have presented a new evolution strategy for multi-objective genetic algorithms that exploits the idea of heterogeneous subpopulations. We defined the concepts of *proximity* and *coverage* to compare Pareto fronts, and how they relate to the *efficiency* and *efficacy* of the algorithm. We have analyzed the influence of immigrants from heterogeneous subpopulations on the evolution process, and how this strategy affects the efficiency and efficacy of the algorithm. Our results show that the evolution strategy based on heterogeneous subpopulations with restart is consistently superior to the traditional strategy, based on homogeneous subpopulations. We have also shown the difference in the solution space for directly and inversely related objective functions, and suggested that this knowledge may be used to select evolution parameters dynamically.

Our approach to scheduling as a multi-objective problem represents a new direction with promising results. We have also shown how the implementation of genetic algorithms to solve scheduling problems can be greatly improved, through the use of proved heuristics to determine the process ordering and a simple encoding for the assignment of processes to processors. These implementation improvements may yield substantial performance gains.

We hope that this paper will stimulate further research in the field of multi-objective genetic algorithms, as well as more applications of evolutionary approaches to solve scheduling problems.

Acknowledgments

Thanks to Yu-Kwong Kwok and Ishfaq Ahmad for the benchmark with dozens of nice scheduling problems. Thanks to Claudia Goldman for fruitful discussions. This research was supported in part by grants from the Ministry of Defense and the Ministry of Science.

References

[1] Peter Brucker. *Scheduling Algorithms*. Springer, 1995.

[2] Erick Cantú-Paz. A survey of parallel genetic algorithms. Technical Report 97003, IlliGAL, University of Illinois, May 1997.

[3] C. A. C. Coello. An updated survey of evolutionary multiobjective optimization techniques: State of the art and future trends. In *1999 Congress on Evolutionary Computation*, July 1999.

[4] E. G. Coffman, editor. *Computer and Job-shop Scheduling Theory*. Wiley, 1976.

[5] R. Corrêa, A. Ferreira, and P. Rebreyend. Scheduling multiprocessor tasks with genetic algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 10(8):825–837, August 1999.

[6] Hesham El-Rewini and Ted G. Lewis. *Distributed and Parallel Computing*. Manning, 1998.

[7] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.

[8] J. Horn and N. Nafpliotis. Multiobjective optimization using the niched pareto genetic algorithm. Technical Report 93005, IlliGAL, University of Illinois, July 1993.

[9] E. S. H. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, February 1994.

[10] Y. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59:381–442, 1999.

[11] J. Maresky, Y. Davidor, D. Gitler, G. Aharoni, and A. Barak. Selectively destructive re-start. In *Proceedings of the 6th. International Conference on Genetic Algorithms*, 1995.

[12] D. C. Mattfeld and C. Bierwirth. A search space analysis of the job shop scheduling problem. Technical report, Dept. of Economics, University of Bremen, Germany, April 1996.

[13] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition, 1996.

[14] J. D. Schaffer. *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, Nashville, 1984.

[15] S. Voget and M. Kolonko. Multidimensional optimization with a fuzzy genetic algorithm. *Journal of Heuristics*, 2:221–244, 1998.

[16] A. Y. Zomaya, C. Ward, and B. Macey. Genetic scheduling for parallel processor systems: Comparative studies and performance issues. *IEEE Transactions on Parallel and Distributed Systems*, 10(8):795–812, August 1999.