Using a Set of Elite Individuals in a Genetic Algorithm

Marko Musnjak
Alstom Croatia
Mala Svarca 155
HR-47000 Karlovac, Croatia
Phone: (+385 47) 665 202
marko.musnjak@power.alstom.com

Marin Golub
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3, HR-10000 Zagreb, Croatia
Phone: (+385-1)6129 967
marin.golub@fer.hr

Abstract. *In this paper, a new method of selection of individuals for crossover is presented and tested. Its main idea is that one of the selected individuals is always a member of an "elite" group, consisting of n best individuals currently in the population. This method does not slow down the execution of the algorithm and shows a marked improvement in search results.*

Keywords. Genetic algorithm, selection, scheduling, elitism

1. Introduction

The genetic algorithm is a heuristic optimization method that does not guarantee finding the optimum of the optimization function [7]. It was first described by John H. Holland in the early seventies of the 20th century and it has since been applied to many areas [8] [9]. The genetic algorithm is a useful tool in optimization of many kinds of problems, but its flaws are that it demands a lot of processing time and does not guarantee finding the optimum. The main motivation for modifying the selection method is improving the search quality, and reaching the optimum more quickly.

Genetic algorithm is based on a simulation of natural evolution while searching a solution for a problem. In nature individuals in a population compete for limited resources. Those that are better adapted to conditions in their environment survive and carry their genes to the next generation, and those poorly adapted perish. New individuals are created by crossover of their parents, and can be modified by mutation. Mutation is a random modification of one or more chromosome elements.

Implementing a genetic algorithm on a computer is simple and, once implemented, can be reused for solving many different problems. An individual, or its chromosome represents one solution to the problem, and most of the time spent executing the genetic algorithm is used for operations with the individuals. Because of that it is important to efficiently model the chromosome. In order to solve a new class of problems usually only the operators of crossover, mutation and calculation of fitness function need to be redefined, and the rest of the implementation can remain unchanged.

The problem used for testing this method is multiprocessor scheduling, and the implementation that is used as a basis for our modification is the same as in [4].

Many methods of selection for crossover have been described, and the method called 3-tournament selection has proven itself to be very robust. In this method three individuals are randomly selected from the population. One with the lowest value of the fitness function is removed from the population. The remaining two individuals are used to generate a new individual that takes the place of the one that was removed. In this article a modification of that method is presented, in which at least one of the individuals selected is in a set of *n* best individuals in the population. This modification is simple to implement, and does not increase demands for processing time or memory space. A similar modification is described in [5].

The implementation of 3-tournament selection is described in section 2, modified 3-tournament selection is described in section 3, and section 4 shows experimental results using the problem of multiprocessor scheduling, and compares with previous work in this area.

2. Implementation of the genetic algorithm

The implementation of the genetic algorithm that was used for the test consists of two main classes, describing the GA procedure and the individual. The first class contains details about the selection method and the conditions for termination of the procedure, while the other contains details about the chromosome, describing the individual and all associated genetic operators.

For testing we used the multiprocessor scheduling problem, described by an acyclic directed graph [4]. Value of the fitness function is defined as the time of the end of the last executing task, with a negative sign, so that the algorithm decreases the time by maximizing this function.

The genetic algorithm is shown in Fig. 1. Initialization of the algorithm is simple: a certain number of individuals is randomly generated. After that three individuals are randomly selected from the population, and one that has the lowest value of the fitness function is removed from the population. The vacant spot is filled by a new individual, created from the two remaining individuals. Mutation is then applied to the new individual. Probability of mutation is a parameter of the algorithm.

```
Begin
    Initialize population
    Repeat
        Choose three individuals
        Remove weakest ind. from pop.
        Crossover + Mutation
        Evaluate new ind.
    until finished
End
```

Figure 1. Genetic algorithm with 3-tournament selection without elite set

After that the value of the fitness function is evaluated and stored for later use. For more complex problems calculation of the fitness function can be expensive, so it is useful to have it cached after the first evaluation. This can be done because the value of the fitness function for an individual does not change after it is created. This procedure is repeated until the termination condition is reached. That condition can be for example the number of generations or high similarity between all individuals.

Result of this algorithm is the individual that had the highest value of the fitness function or, in our experiment, the shortest time to complete all tasks.

3. Implementation of the elite set

Implementation of the genetic algorithm with 3-tournament selection and elite set is not very different from the original algorithm. The modified algorithm is shown in Fig. 2.

```
Begin
    Initialize population
    Find members of elite set
    Find candidate for removal from elite
    Repeat
        Choose one ind. from elite set
        Choose two more individuals
        Remove weakest ind. from pop.
        Crossover + Mutation
        Evaluate new ind.
        If all three are elite then
            reconstruct elite
        else if new > candidate
            place new into elite
            find new candidate
    until finished
End
```

Figure 2. Genetic algorithm with 3-tournament selection and elite set

The beginning is the same: a set of solutions is initialized. After initialization, the elite set is constructed by finding the best *n* individuals from the population. The elite set is implemented as a field of indices in the population. This allows us to handle elite individuals like all others, since they are not removed from the population, but we still have quick access to all elite individuals. Construction of the elite set is implemented as a simple linear search of the population. After that the candidate for removal from elite set is selected. It is the individual in the elite set that has the lowest value of the fitness function. Every time a new individual is created it is compared to the worst individual in the elite set. To speed up the comparison the index of the candidate is kept also.

After the initialization of the population and all necessary arrays, the crossovers are started. First an individual from the elite set is chosen randomly. Two other individuals are also chosen randomly, regardless of whether they are members of the elite set or not. The worst individual is removed from the population, and crossover and mutation proceed as in the original algorithm. After the new individual is created, it needs to be checked if it is good enough to become a member of the elite set.

Worst case of selection is when all three individuals are members of the elite set. It is then possible for the new individual to have lower value of the fitness function than the individual that was removed, and should not be a member of the elite set. If that occurs, the elite set is reconstructed by searching the entire population. If the worst case did not occur, a simpler procedure is executed: the new individual is compared to the candidate for removal from the elite set. If it is better, it takes the candidate's place in the elite set. A new candidate is found by a linear search of the elite set. It is a very fast procedure since the elite set contains only a few elements, significantly less than the entire population. This shortened procedure saves time because the entire population does not have to be searched after creating a new individual.

As can be seen from the description, this modification only marginally increases the demands for processing time and storage space. The most demanding operation that can occur is a linear search of the population, which is not a problem since the population usually contains a relatively small number of individuals, and it happens only in the case if all three individuals are members of the elite set. Probability of this happening in a population of 120, with an elite set of 4, is 0.0014%, which means that it will happen once in about 70000 algorithm cycles. There was no noticable increase in execution time, with all execution times at 65 minutes.

In the more likely case that at least one if the individuals is not a member of the elite set only a comparison of two cached values is necessary, and maybe a linear search of the elite set. The search also takes very little time because the elite set has a very small number of individuals. The increase in use of storage space is also negligible All that is needed is a small array of integers to maintain the contents of the elite set.

4. Experimental results

Experiments were performed on a multiprocessor scheduling problem, scheduling 1104 tasks on 7 processors [1] [2] [10], with population sizes 120 and 240, with mutation probability 1% and 40000 algorithm cycles. The number of elements in the elite set has varied between 1 and 5, and a comparison with the classic algorithm has been performed. For every combination of parameters 15 experiments were performed, and the best, worst and average results have been recorded. Execution times were equal for all combinations, and on the computer we used (Intel Pentium 4, 2 GHz, 1 GB RAM) it took a little over an hour, which shows that there was no increase in processing time due to the modifications.

The problem description was taken from [6], where an optimal value of 1504 was found using a heuristic method. The best solution found using this algorithm was only 6% more than optimal value. The problem is described by a directed acyclic graph and a partial precedence relation is given, along with the time necessary to perform each task and number of processors. Every task is present in the graph, and appears only once. One processor can execute only one task at a time. The number of processors is fixed during the execution of all tasks.

The best result (see Table 1) was found with 240 individuals, with 4 members of the elite set. That result was 1596, and the second best result was 1598, found with 120 individuals, 5 of which were in the elite set. The best result found using the unmodified algorithm was 1664 time units, with a population of 120 individuals.

Table 1. Best results

Elite	-	1	2	3	4	5
120	1664	1606	1606	1615	1600	1598
240	1707	1599	1611	1600	1596	1605

The best average result (see Table 2) was found with two combinations of parameters: population of 120, with 4 elite individuals, and also with population of 120, with 3 elite individuals. That result is 1617 time units. The unmodified algorithm found average solutions of 1680 for 120 individuals and 1716 for 240 individuals.

Table 2. Average results

Elite	-	1	2	3	4	5
120	1680	1627	1624	1626	1617	1621
240	1716	1627	1626	1617	1622	1624

Two worst results (see Table 3) were found with the unmodified algorithm; for population of 120 the result was 1698, and for population of 240 it was 1729. The worst result found using the elite set was for 240 individuals and 1 in the elite set, and that result was 1670 time units.

Table 3. Worst results

Elite	-	1	2	3	4	5
120	1698	1653	1642	1642	1635	1647
240	1729	1670	1654	1637	1642	1644

From tables 1-3 it can be seen that the best combination was population of 120 and 4 elite members. That combination found the best average result, and it was very close to the best solution found in all experiments (only 4 time units longer). It can also be seen that all combinations using the modified algorithm show significantly better results than the unmodified algorithm.

In figure 3 we show a comparison of average results from two selection methods, as well as the best solution found, and the solution from [6]. The top most line is the unmodified selection algorithm, and the line below is our modified algorithm. Next two lines are best found solution and solution from [6]. It can be seen from figure 3 that the modified algorithm makes much better progress in the early stages of the genetic algorithm. Later it approaches the solution at about the same speed as the unmodified algorithm.

In [4] similar experiments were performed and we repeated them using the modified algorithm. The first problem was scheduling 452 tasks on 20 processors, with the optimal solution of 537 time units. In [4] the parameters were: mutation probability 1%, population size 20 and 10000 iterations, and we used the same parameters, modifying the size of the elite set from 1 to 4. In [4] the best result was 540 time units, while with the modified algorithm results were much better (see Table 4). With 4 elite members over 85% of experiments found the global optimum. The worst results were found with only one member of the elite set, only 25% found the best solution, and average solution was 544 time units.

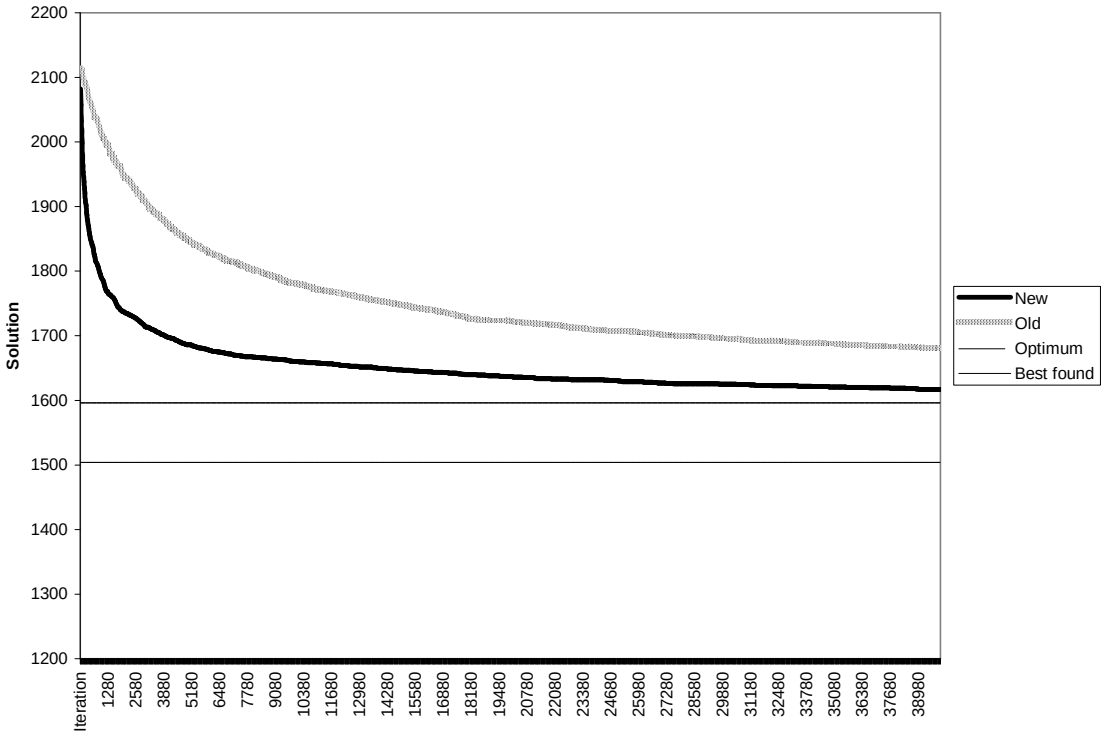

Figure 3. Evolution process for classic selection and new selection compared with best solution found and global optimum

Table 4. Results for problem from [4]

Elite	1	2	3	4
Average	543	537.87	537.27	537.13

Second problem was scheduling 473 tasks to 4 processors, with 30000 iterations, population of 50 individuals and mutation probability 1%, with the optimal solution of 1178 time units. In [4] the best solution was 1182, and average 1184 time units. Our results for this problem were approximately same, with the best solution 1182 found for all combinations and average solution of 1185 for 4 members in the elite set. These results are compared in table 5.

Table 5. Problems and best results

	No. of tasks	No. of proc.	New method	Old method	Solutin from [6]
1.	1104	7	1596	1680	1504
2.	452	20	537 (538*)	540	537
3.	473	4	1182	1182	1178

* 538 is the average solution from 60 runs

5. Conclusion

The method described here is an efficient upgrade to the standard 3-tournament selection method. Adding it to an existing implementation was simple, and did not increase demands for memory or processing time. Results achieved by this method are better than results using the unmodified 3-tournament selection algorithm. Future direction of this research is adapting this method to parallel genetic algorithm [3], where special attention is needed to maintain the elite set.

6. References

[1] Ahmad I, Kwok Y-K. On parallelizing the multiprocessor scheduling problem. IEEE Transactions on Parallel and Distributed Systems, 1999; 10(4).

[2] Correa R.C., Ferreira A., Rebreyend P, Scheduling multiprocessor task with genetic algorithm. IEEE Transactions on Parallel and Distributed Systems, 1999; 10(8).

[3] Golub M., Jakobovic D., Budin L. Parallelization of elimination tournament selection without synchronization. In: Proceedings of the 5th IEEE International Conference on Intelligent Engineering Systems; 2001. p. 85-89, available from: http://www.zemris.fer.hr/~golub/clanci.html

[4] Golub M., Kasapovic S. Scheduling multiprocessor tasks with genetic algorithms. In: Proceedings of the IASTED International Conference on Applied Informatics; 2002 Feb 18-21; Insbruck, Austria. 2002. p. 273-278, available from: http://www.zemris.fer.hr/~golub/clanci.html

[5] Jung, S.H., Queen-bee evolution for genetic algorithms. IEE Electronic Letters, 2003.

[6] Kasahara Laboratory. Advanced Computing Systems, available from: http://www.kasahara.elec.waseda.ac.jp [01/29/2004]

[7] Michalewicz Z. Genetic Algorithms + Data Structures = Evolutionary Programs. Berlin: Springer-Verlag; 1992.

[8] Munetomo M, Takai Y, Sato Y. An efficient migration scheme for subpopulation-based asynchronous PGA. Technical Report HIER-IS-9301, Hokkaido University Information Engineering, 1993.

[9] Talbi E.-G., Bessiere P., A parallel genetic algorithm for the graph partitioning problem. Supercomputing, 1991. p. 312-320.

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