Parallel genetic algorithm with parameter adaptation

Shisanu Tongchim*[a] and Prabhas Chongstitvatana†[a]

[a]Department of Computer Engineering, Chulalongkorn University,
Bangkok 10330, Thailand

This paper presents an adaptive algorithm that can adjust parameters of genetic algorithm according to the observed performance. The parameter adaptation occurs in parallel to the running of genetic algorithm. The proposed method is compared with the algorithms that use random parameter sets and a standard parameter set. The experimental results show that the proposed method offers two advantages over the other competing methods: the reliability in finding the optimal solution and the time required for finding the optimal solution.

Keywords: Parallel Genetic Algorithm; Parameter Adaptation

1. INTRODUCTION

Genetic Algorithm (GA) [1,2] is a general search algorithm that can be applied to a wide range of problem domains. The process of GA is controlled by several parameters, e.g. population size, mutation rate. These parameters largely determine the success and efficiency of GA in solving a specific problem. Unfortunately, these parameters interact with each other in a complicated way. Many practitioners find a promising parameter set for a particular problem by trying various combinations of the control parameters. This approach of parameter selection obviously requires a lot of computation which sometimes is larger than the time used for solving a particular problem by GA itself. A prominent example of exhaustively testing several combinations of parameters was shown in the study done by Schaffer *et al.* [3]. That study carefully examined the performance of GA using various combinations of the control parameters. The experiments that involved several test functions and parameter combinations took approximately 1.5 CPU years.

In this paper, an adaptive mechanism for dynamically adjusting the parameters of GA is proposed. This mechanism performs parameter adjustments during the run of GA. The proposed method adjusts both the operator selection and the parameter values. Four parameters are adjusted by the algorithm: crossover operator, mutation operator, crossover rate, and mutation rate. Two performance measurements are used to compare the proposed method with the algorithms using a standard parameter set and random parameter sets. The first measurement illustrates the reliability of the algorithm using the number of runs yielding the optimal solution, whereas the second measurement shows the resource used by the algorithm to find the solution to the given problem. The results show that the proposed method outperforms the other algorithms for both performance measurements under a set of test problems.

2. BACKGROUND

Over the past decade, many variations of GA have been investigated. Many representations and operators have been proposed in order to tackle with some problems which cannot be solved properly with the standard bit string representation and operators. However, the advent of these various implementations also increases the difficulty of parameter selection. Harik and Lobo [4] pointed out that the user decision for the parameter setting can be divided into two categories. The first category was the operator selection and the coding. The second category was the values of these parameters. The article by Eiben *et al.* [5] provided a comprehensive re-

*43718072@student.chula.ac.th
†prabhas@chula.ac.th

view and a classification of the various techniques of parameter control in evolutionary algorithms. The dissertation by Lobo [6] also gave a review of the research in this area.

Early systematic research of the GA performance using various combinations of parameters was undertaken by De Jong [7]. The performance of GA was measured on five test functions which were later used as a standard test suite by many researchers. The findings from his empirical study showed that the following parameters yielded a good performance: population size 50-100, one-point crossover probability of 0.6 and bit mutation probability of 0.001. These parameter values have been widely used by many researchers and accepted as a *standard* parameter setting.

Another study trying to understand interactions among the parameters of GA was carried out by Deb and Agrawal [8]. They investigated the effect of three parameters: population size, crossover rate, and mutation rate. The results showed that mutation-based approaches and crossover-based approaches perform quite differently between simple problems and complex problems. In either approach, a correct population size is needed to achieve good performance.

Grefenstette [9] investigated the use of meta-level GA to select feasible parameter values. The method was designed as two levels of GA. The higher level GA maintained a population of parameter sets. The lower level GA used the parameter sets from the higher level GA to solve the problems. The observed performance of the lower level GA was assigned as the fitness of the parameter set. The results showed that the obtained parameter set did only slightly better than the parameter set found by De Jong.

Pham [10] proposed a technique for parameter selection by establishing a competition among several subpopulations that use various parameter sets. Several populations independently evolved by using their own parameter sets. These populations were maintained by a single processor. The populations with good parameter sets received additional processing time to evolve further.

Lis [11] introduced a technique to adapt the mutation rate in a model of parallel GA. Several subpopulations evolved separately on different processors by using various mutation rates. After a predetermined interval, these populations were compared. If the best result was acquired from the processor with the highest mutation rate, the mutation rates of other processors were shifted by one level. The mutation rates were also reduced by one level if the best result was obtained from the processor with the lowest mutation rate.

Schlierkamp-Voosen and Mühlenbein [12] used competition between subpopulations with different parameter sets for selecting the proper parameter set. The size of each group varied, while the total population size was fixed. Each group competed against other groups for gaining its size. The size of the best group was increased, while all other groups were decreased. A similar method was used in [13] for choosing the best crossover operator.

3. PROPOSED METHOD

The intention of the proposed method is to provide a technique for adjusting parameters while the search is ongoing. By using the concept of coarse-grained parallelization, the population is divided into a few large subpopulations. These subpopulations evolve independently and concurrently on different processors. After a predefined period of time, some selected individuals are exchanged via a migration process. The subpopulations operate by using different parameter values.

GA is applied to the evolution of the parameter sets. Pseudo-code of the algorithm is shown in Fig. 1. The subroutine *Parameter_adaptation* is an extension to a general coarse-grained model. The chromosome representation of each parameter set is a vector of integer numbers. Let us assume that $\vec{C} = (c_1, \ldots, c_n)$ $(c_i \in [a_i, b_i] \subset I^+, i = 1, \ldots, n)$ is an integer chromosome. The value of a particular gene (c_i) denotes the value of the i_{th} parameter. Two parameter sets are employed in each subpopulation. The average fitness of individuals that are processed by a particular parameter set is used for the assessment of the fitness value of the parameter set. The best parameter set selected from two parameter sets in each

node is allowed to mate with another parameter set from other nodes. Each processor decides whether the local parameter set mates with the neighboring parameter set. In particular, each node sends its best parameter set and the fitness value of this parameter set to other nodes. The topology used in this study is a loosely connected topology, the one-way ring topology, which the communication is limited to occur between the adjacent nodes. If the fitness of the parameter set from the adjacent node is better than the best local parameter set, two new parameter sets are produced by applying a sequence of genetic operators, uniform crossover and mutation, to both parameter sets. Each parameter set is used to produce a half of the new population. For mutation, each field in both parameter sets may be replaced by a random value according to the predetermined probability. The mutation rate is set to 0.25.

The proposed method is motivated in part by the method proposed by Pham [10]. Pham maintains several populations using different parameter sets in order to avoid the unsuccessful run from a poor initial parameter set. That method is penalized by the increase in the computational cost since several populations evolve concurrently in a single processor. Our proposed method overcomes this disadvantage by using a parallel model of GA. In particular, a coarse-grained model is used in order to evaluate several parameter sets simultaneously. Another difference is that Pham uses static parameter sets, whereas our method dynamically adjusts parameter sets according to the observed performance. The proposed method can be viewed as meta-level GA. However, this method mainly differs from the work done by Grefenstette [9] that the parameters are adjusted during the run of the algorithm, whereas the method of Grefenstette finds the parameters before the run of the algorithm.

Another important distinction between our method and other subpopulation-level adaptive methods [10,12,13] is that our method is more suitable for parallelization. The techniques in these studies reward the successful subpopulation by increasing its size [12,13], or by giving additional processing time [10]. These techniques may not be effective when implemented in parallel forms since they are likely to cause uneven work loads among processors.

4. EXPERIMENT AND DISCUSSION

4.1. Experimental Design

The experiments are carried out on a dedicated cluster of PC workstations. The number of processing nodes used in the experiments is 8. The program is based on a modified version of LibGA software package [14]. MPICH, a portable implementation of MPI standard, is used for providing communication functions in parallel computing environment.

The following algorithms using the proposed method and other methods are examined and compared.

1. *Adaptive algorithm*: This is the proposed method.
2. *Uniform random algorithm*: A parameter set is randomly generated at the beginning of the algorithm. All subpopulations use this parameter set.
3. *Diverse random algorithm*: At the beginning, each subpopulation randomly creates its own parameter set. This algorithm is comparable to the adaptive algorithm without the parameter adaptation. The similar techniques of using different parameter sets on the multiple subpopulations were also presented in [15,16]. However, their parameters in each subpopulation were not random.
4. *Static algorithm*: This algorithm uses a static parameter set from the study by De Jong [7]

The following four parameters are involved in the experiments.

1. *Crossover operator*: The five crossover types used in this study are listed as follows: (i) one point crossover (ii) uniform crossover with a probability of 0.5 (iii) two point crossover (iv) uniform crossover with

```
0:    initialize the population, P
1:    while generation < max_generation
2:        evaluate P
3:        apply genetic operators determined by the first parameter set to create
              the first half of the new population, P1'
4:        apply genetic operators determined by the second parameter set to create
              the second half of the new population, P2'
5:        merge P1' and P2' to P'
6:        replace P with P'
7:        if an interval of K generations is reached
8:            Migration
9:            Parameter_adaptation
10:       end
11:       generation = generation + 1
12:   end
13:
14:   subroutine Migration
15:       send and receive migrants
16:       add migrants to P
17:   end
18:
19:   subroutine Parameter_adaptation
20:       send the best parameter set with its fitness
21:       receive the parameter set with its fitness
22:       if the received parameter set is better
23:           produce two new parameter sets by uniform crossover and mutation
24:       end
25:   end
```

Figure 1. Pseudo-code of the algorithm in each node

a probability of 0.1 (v) uniform crossover with a probability of 0.2[3]

2. *Crossover rate*: In the experiments, five values for crossover rate are used ranging from 0.2 to 1 in increments of 0.2.

3. *Mutation operator*: The five mutation types used in the experiments are as follows: (i) invert a bit (ii) random bit value (iii) swap two values (iv) random bit value with a bias toward zero (probability of 0.9) (v) random bit value with a bias toward one (probability of 0.9)

4. *Mutation rate*: Six mutation rates are allowed varying from 0 to 0.1 in increments of 0.02.

The population size plays an important role in determining the success and the computational cost in finding a solution to a particular problem. Thus, all experiments are conducted over a range of population sizes. In all algorithms, the selection scheme is roulette-wheel selection. All reported results are averaged over 20 runs with different random seeds. The exchanges of migrants and parameter sets are synchronized. The exchange interval is 5 generations. In the migration of individuals, six solutions selected by using roulette-wheel selection from each subpopulation are exchanged. The received migrants are incorporated to the new pool. The maximum number

[3]Uniform crossover typically swaps two corresponding values of two parents with a probability of 0.5. Uniform crossover with probabilities other than 0.5 is inspired by the studies [17,18].

of generations for all experiments is 500. The test problems used in this study are as follows: (1) 300-bit onemax problem, (2) 300-bit contiguous bits problem, (3) 50 copies of minimal deceptive problem (MDP), (4) zero/one multiple knapsack problem and (5) royal road problem.

We refer readers to [19] for the definitions of the problems 1 and 2. The description of the third problem can be found from [2]. For the knapsack problem, we use the "sento1-60" problem which was introduced in [20]. The optimal solution is 7772. The problem instance is available from OR-Library [21]. The fitness function proposed in [22] is used. For the royal road problem, the description of the problem was presented in [23]. By using Holland's default settings, the optimal solution is 12.8.

4.2. Experimental Results

We adopt two measurements from the study by Deb and Agrawal [8]: *Performance* and *Unuse Factor*. The performance is the ratio of the number of runs yielding the optimal solution to the total number of runs, except that the number of runs reaching 1% from the optimal is used in the knapsack problem. The unuse factor (u) is calculated as follows:

$$u = 1 - \frac{g}{g_{max}} \tag{1}$$

where g is the number of generations required to solve the problem, g_{max} is the maximum number of generations [4]

Figure 2a and 2c show the performance on the onemax problem and the contiguous bits problem respectively. The results on the onemax problem and the contiguous bits problem are nearly identical. The adaptive algorithm can find the optimal solution in all runs over the range of population sizes. The diverse random algorithm occasionally finds the optimal solution in all runs. The uniform random algorithm achieves the moderate performance. The static algorithm attains the lowest performance. Figure 2b and 2d illustrate the unuse factor on the onemax problem and the contiguous bits problem respectively. The unuse factor graphs indicate that the proposed method has the highest remaining generation number. This means that the proposed method uses the shortest period in finding the optimal solution. The static algorithm has the lowest convergence rate. When increasing the population size, the unuse factor reduces to zero. This means that the static algorithm is unable to find the optimal solution in the given time.

The performance on the minimal deceptive problem is depicted in figure 2e. The adaptive algorithm finds the optimal solution in all runs when the population size increases to 40. The uniform random algorithm cannot find the optimal solution in all runs. The diverse random algorithm requires the population size at least 220 to find the optimal solution in all runs. The performance of the static algorithm increases considerably as the population size increases. Moreover, the static algorithm finds the optimal solution in all runs by using the population size at least 160. The unuse factor on the minimal deceptive problem is illustrated in figure 2f. The proposed method uses the shortest duration in finding the optimal solution.

In figure 3, the results on the problems which are more difficult than the first three problems are presented. For the knapsack problem, the proposed method achieves significant better performance in finding the optimal solution than the other competing methods. The proposed method has the highest remaining generation number as well. For the royal road problem, the ratios of runs reaching the optimal solution of the proposed method and the diverse random algorithm are nearly equivalent. The remaining generation numbers of both algorithms are also approximately equivalent. The uniform random algorithm cannot find the optimal solution in the given period. The static algorithm is able to find the optimal solution in some runs when the population size is sufficient.

In all problems, the population size notably effects the results for both performance measurements. Although this study does not adapt the population size, this does not mean that finding the appropriate population size is unimportant. Choosing the suitable population size is a com-

[4] The unuse factor is originally calculated by using the number of function evaluations.

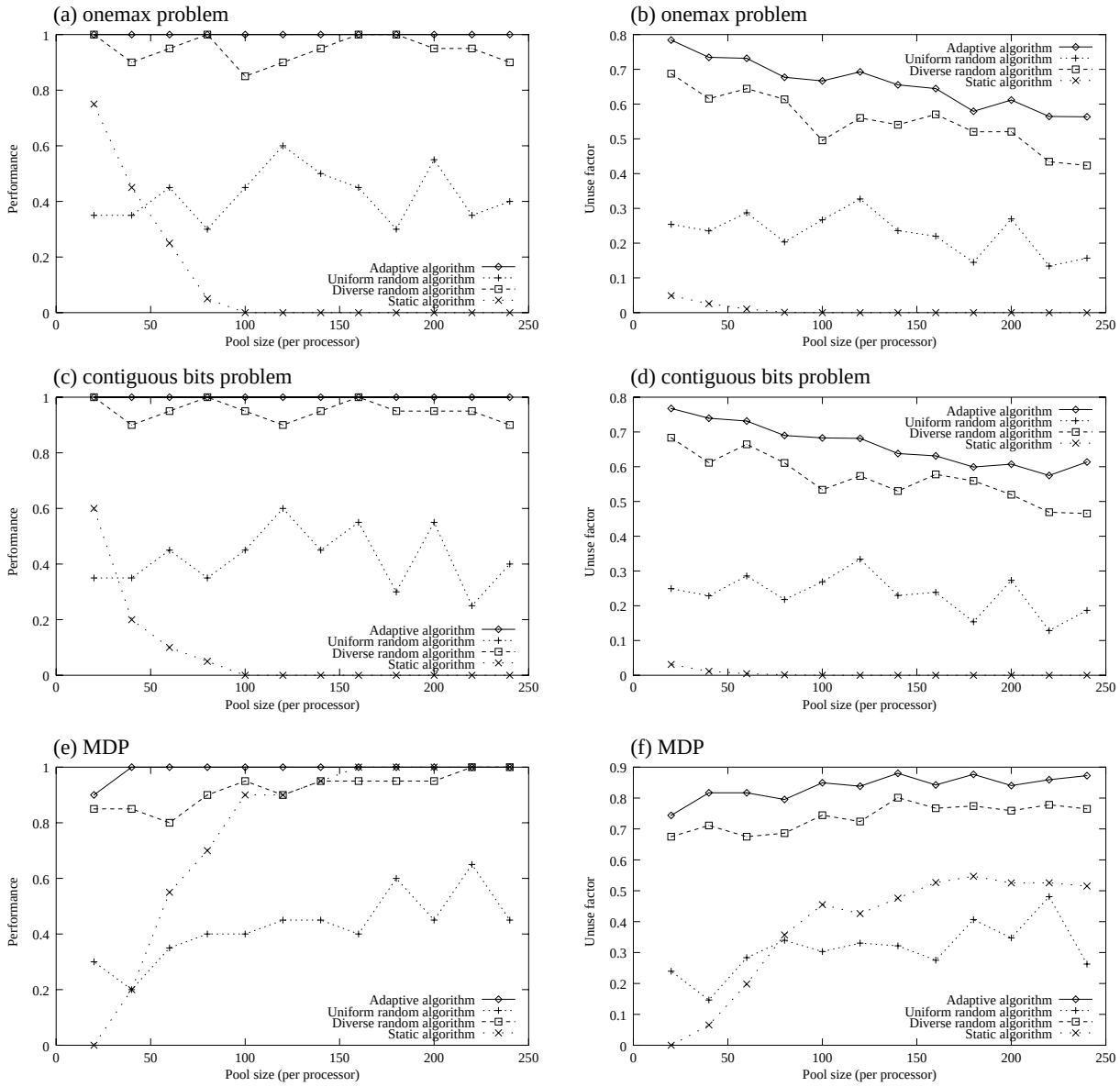

Figure 2. Performance and Unuse factor for (a,b) the onemax problem, (c,d) the contiguous bits problem, (e,f) the minimal deceptive problem

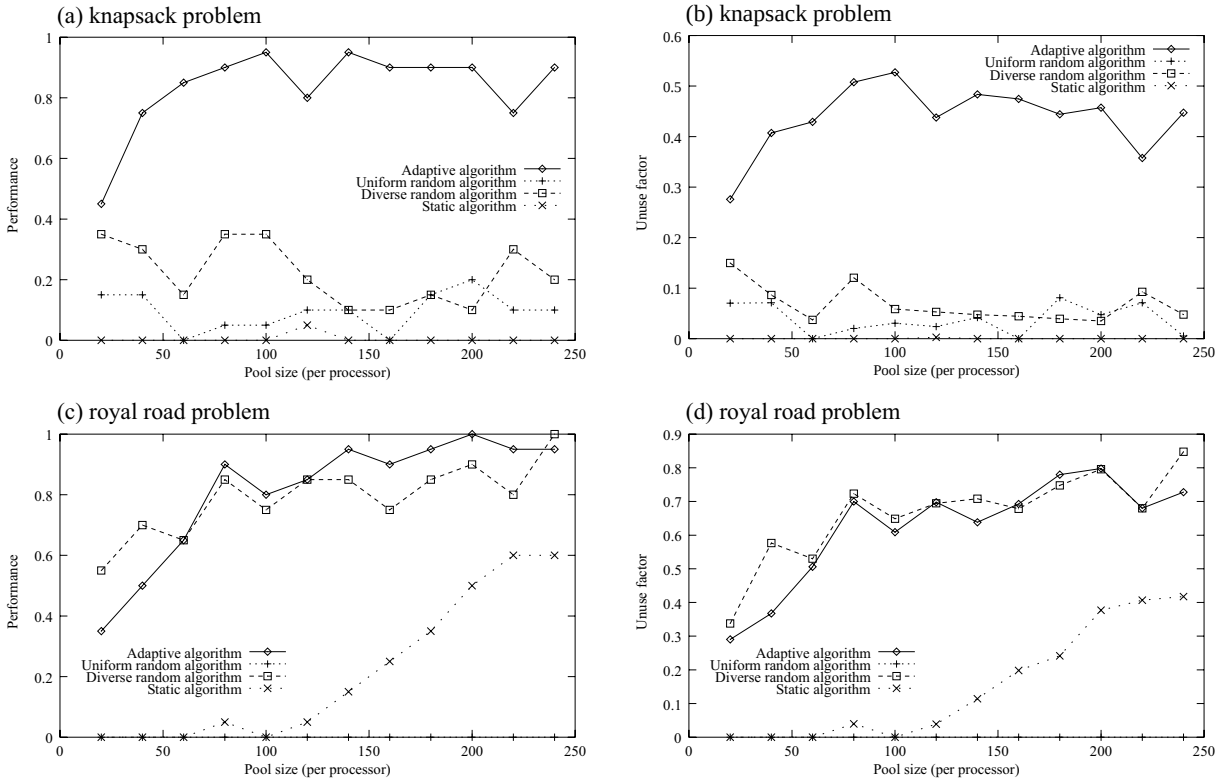

Figure 3. Performance and Unuse factor for (a,b) the knapsack problem, (c,d) the royal road problem

mon problem encountered by the GA practitioners. Harik and Lobo [4] introduced a method for choosing the population size. Multiple populations with different population sizes were maintained on a single processor. The smaller populations received more function evaluations than the larger ones. When a larger population had an average fitness greater than that of a smaller population, the smaller population was eliminated. The method of Harik and Lobo is based on competitive populations similar to our method. It is interesting to investigate how it can be applied to adapt the population size in our case.

5. CONCLUSIONS

This paper presents a method for automatically adjusting control parameters of GA. The results show its effectiveness in the following points: (i) Our adaptive method is shown to be more reliable in finding the optimal solution than the others. (ii) Our approach uses the lowest number of generations in finding the optimal solution. (iii) In the third problem, our method finds the optimal solution in all runs by using the smallest population size. The smaller population size helps in reducing the computational time.

REFERENCES

1. J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, Michigan, 1975.
2. D. E. Goldberg. *Genetic Algorithm in search, optimization and machine learning.* Addison-Wesley, 1989.

3. J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60, 1989.
4. G. R. Harik and F. G. Lobo. A parameter-less genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 258–265, 1999.
5. Á. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, July 1999.
6. F. G. Lobo. *The parameter-less genetic algorithm: Rational and automated parameter selection for simplified genetic algorithm operation.* PhD thesis, University of Lisbon, Portugal, 2000.
7. K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.
8. K. Deb and S. Agrawal. Understanding interactions among genetic algorithm parameters. In *Foundations of Genetic Algorithms 5*, pages 265–286, 1998.
9. J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(1):122–128, Jan/Feb 1986.
10. Q. T. Pham. Competitive evolution: a natural approach to operator selection. In *Progress in Evolutionary Computation, Lecture Notes in Artificial Intelligence, X. Yao (ed.), Springer-Verlag, Heidelberg*, volume 956, pages 49–60, 1995.
11. J. Lis. Parallel genetic algorithm with the dynamic control parameter. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 324–329, 1996.
12. D. Schlierkamp-Voosen and H. Mühlenbein. Strategy adaptation by competing subpopulations. In *Parallel Problem Solving from Nature 3, Lecture Notes in Computer Science*, volume 866, pages 199–208, 1994.
13. Á. E. Eiben, I. G. Sprinkhuizen-Kuyper, and B. A. Thijssen. Competing crossovers in an adaptive GA framework. In *Proceedings of the 5th IEEE Conference on Evolutionary Computation*, pages 787 –792, 1998.
14. A. L. Corcoran and R. L. Wainwright. *Using LibGA to Develop Genetic Algorithms for Solving Combinatorial Optimization Problems*, volume 1 of *Practical Handbook of Genetic Algorithms, L. Chambers (ed.)*, chapter 6, pages 143–172. CRC Press, 1995.
15. F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43–63, April 2000.
16. P. Adamidis and V. Petridis. Co-operating populations with different evolution behaviours. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 188 –191, 1996.
17. W. M. Spears and K. A. De Jong. On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.
18. W.-C. Huang, C.-Y. Kao, and J.-T. Horng. A genetic algorithm approach for set covering problems. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 569–574, 1994.
19. G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, 1989.
20. S. Senyu and Y. Toyoda. An approach to linear programming with 0-1 variables. *Management Science*, 15:B196–B207, 1967.
21. J. E. Beasley. OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.
22. S. Khuri, T. Bäck, and J. Heitkötter. The zero/one multiple knapsack problem and genetic algorithms. In *Proceedings of the 1994 ACM Symposium on Applied Computing*, pages 188–193, 1994.
23. T. Jones. A description of holland's royal road function. *Evolutionary Computation*, 2(4):409–415, 1994.