Self-Adapting Evolutionary Parameters: Encoding Aspects for Combinatorial Optimization Problems

Marcos H. Maruo[1], Heitor S. Lopes[1], and Myriam R. Delgado[1]

Centro Federal de Educação Tecnológica do Paraná -CEFET/PR
Av. Sete de Setembro, 3165 – 80230-901 Curitiba, Brazil
{ maruo, hslopes, myriam}@cpgei.cefetpr.br

Abstract. Evolutionary algorithms are powerful tools in search and optimization tasks with several applications in complex engineering problems. However, setting all associated parameters is not an easy task and the adaptation seems to be an interesting alternative. This paper aims to analyze the effect of self-adaptation of some evolutionary parameters of genetic algorithms (GAs). Here we intend to propose a flexible GA-based algorithm where only few parameters have to be defined by the user. Benchmark problems of combinatorial optimization were used to test the performance of the proposed approach.

1 Introduction

The two major definitions in applying any heuristic search algorithm to a particular problem are the representation and the evaluation (fitness) function. When using an evolutionary algorithm (EA) it is also needed to specify how candidate solutions will be changed to generate new solutions. This encompasses the definition of genetic operators (mainly mutation and crossover) suited to the encoding, and a selection method to enforce the survival-of-the-fittest evolutionary rule. Each of these components may have parameters, for instance: the probability of mutation, the tournament size of selection, or the population size. Values of these parameters greatly determine the quality of the solution found and the efficiency of the search [1]. Frequently, choosing suitable parameter values, is problem-dependent and requires previous experience of the user. Since this can be a time-consuming task, considerable effort has been applied to develop good heuristics for it, so as to avoid trial-and-error [2]. Despite its crucial importance, there is no consistent methodology for the determination of the running parameters of an EA, which are, most time, arbitrarily set within predefined ranges.

Globally, we distinguish two major forms of setting parameter values: parameter *tuning* and parameter *control*. The first means the commonly practised approach that tries to find good values for the parameters *before* running the algorithm, and then tuning the algorithm using these values, which remain fixed during the run. A general drawback of the parameter tuning approach, regardless of how the parameters are tuned, is based on the observation that a run of an EA is an intrinsically dynamic, adaptive process. The use of rigid parameters that do not change their values is thus in contrast with this spirit [1]. Therefore, parameter control is an alternative. In this approach, a run is started with initial values for the parameters, and then they are dynamically changed *during* the run.

This paper aims to analyze the parameter control in contrast with the parameter tuning technique while solving combinatorial optimization problems, taking into account three important issues:

- Does real-valued encoding take any benefit for a discrete problem using parameter control in a GA?
- How the performance of the GA is affected by the adaptation in their parameters?
- Is there any advantage for the user using parameter control in a GA?

In this work the parameter control technique is based on self-adaptation of several parameters associated with the evolutionary process. The main goal here is to produce a flexible GA, in which only few running parameters need to be defined by the user.

2 Parameter Control

In classifying parameter control techniques of an evolutionary algorithm, many aspects can be taken into account: *What* is changed? *How* the change is made? The *scope/level* of change and the *evidence* upon which the change is carried out.

According to Eiben et al. [1], the change can be categorized into three classes:

- *Deterministic* parameter control: this take place when the value of a parameter is altered by some deterministic rule.
- *Adaptive* parameter control: this take place when there is some form of feedback from the search that is used to determine the direction and/or the magnitude of the change to the parameter.
- *Self-adaptive* parameter control: the idea of "evolution of the evolution" can be used to implement the self-adaptation of parameters. Here the parameters to be adapted are encoded into the chromosome and undergo the action of genetic operators. The better values of these encoded parameters lead to better individuals which, in turn, are more likely to survive and produce offspring and hence propagate these better parameter values.

Some authors have introduced a different terminology based on the level of change [3, 4] and in how the change is made [5]. A more detailed discussion of parameter control can be found in [6].

The straightforward way to control parameters in a deterministic way is by using parameters that may change over time, that is, by replacing a parameter p_{stat} by a function $p_{dyn}(t)$, where t is the generation counter. However, this process presents some disadvantages: the difficulty in designing an optimal function

$p_{dyn}(t)$, and the fact that this function does not take into account any clue of the actual progress in solving the problem. Hence, it is thus seemingly natural to use an evolutionary algorithm not only for finding solutions to a problem, but also for tuning the (same) algorithm to the particular problem. Technically speaking, it is tried to modify the values of parameters during the run of the algorithm by taking the actual search progress into account. As discussed in [1], there are two ways to do this. The first way is to use some heuristic rule which takes feedback from the current state of the search and modifies the parameter values accordingly (adaptive parameter control), such as the credit assignment process presented by [7]. A second way is to incorporate parameters into the chromosomes, thereby making them subject to evolution (self-adaptive parameter control), like the approach presented in [8].

2.1 Mutation Parameters Control

De Jong recommended $pm = 0.001$ [9], the meta-level GA used by Grefenstette [10] indicated $p_m = 0.01$, while Schaffer et al. [11] came up with $p_m \in [0.005, 0.001]$. Folowing the earlier work of Bremmermann [12], Mühlenbein derived a formula for p_m which depends on the length of the bitstring (L), namely $p_m = \frac{1}{L}$ should be a generally "optimal" static value for p_m. This rate was compared with several fixed rates by Smith and Fogarty [13]who found that $p_m = \frac{1}{L}$ outperformed other values for p_m in their comparison. The same was found by Bäck [14] using gray coding. However, as pointed by [15, 13], there is an increasing body of evidence that the optimal rate of mutation is not only different for every problem encoding but will vary with evolutionary time according to the state of the search and the nature of the fitness landscape being explored.

These ideas have been applied to a generational GA by adding a further 20 bits to the problem genotype, which were used to encode the mutation rate [16]. The results showed that in generational setting the mechanism proved competitive with a genetic algorithm using a fixed (optimal) mutation rate, provided that a high selection pressure was maintained (this is referred to as "extinctive" selection). In [17] they proposed two simple adaptive mutation rate control schemes and show their feasibility in comparison with several other fixed and adaptive schemes applied to combinatorial optimization problems.

2.2 Crossover Parameters Control

Effectiveness of crossover has been frequently discussed in the literature, and some interesting results were reported by De Jong [9]. More recently, Schaffer and Eshelman [18] empirically compared mutation and crossover and concluded that the latter is capable of exploring epistatic problems more efficiently, in contrast with the mutation alone.

There are several types of crossover, but GAs use more frequently only one- or two-point crossovers. However, there are some situations when using a multi-point crossover can be beneficial [19, 20]. Then, an interesting option is the uniform crossover, that produces, in average, $\frac{L}{2}$ combinations in L-long strings [21, 20]. Besides the empirical research, many efforts have been directed to the theoretical comparisons between different crossover types [22, 23]. However, conclusions are not general enough to foresee which crossover is the best for a given problem. For instance, such theoretical approaches do not consider population size, although this parameter can affect directly the utility of crossover [24]. Furthermore, there are evidences that the utility of mutation operators can also be affected by the population size: it seems to be more useful than crossover when the population is small, and crossover seems to be more effective when the population is large [25].

Spears [8] proposed a self-adapting mechanism that chooses between two crossover types: two-point and uniform crossovers. An extra bit is added to the chromosome indicating which type of crossover it will be used for this particular individual. Descendants will inherit the crossover type from parents. Some experiments indicate that the GA using adaptive crossover had performance equal or better than a classic GA for a set of test problems [11].

When we use multi-parent operators [26], a new parameter is included: the number of parents used in the crossover. Eiben [27] presents an adjustment mechanism for the recombination arity based on competing sub-populations. In particular, the population is divided into disjoint sub-populations, and each one uses a crossover with different arity. These sub-populations evolve independently during a time-window and then they interchange information by allowing migrations between them. Migration favors those sub-populations that evolved better within the time-window, allowing them to be increased, accepting migrants. Conversely, sub-populations that evolved worse, loose individuals and decrease in size. This method achieved similar performance than the conventional GA using a 6-parents crossover. However, this algorithm does not succeed to identify clearly the best operators, regarding the population sizes, thus agreeing with Spears' experiments [8].

3 Proposed Approach: Self-Adapting Parameters

As discussed before, parameter control in evolutionary algorithm is a poorly structured, ill-defined, complex problem [1], and then, self-adapting appears as an interesting alternative. Most works in recent literature discuss the adaptation of just one evolutionary parameter at a time (e.g., probability of crossover and mutation). In this work it is aimed to self-adapt concomitantly several parameters associated with the evolutionary process. It is also evaluated their influence on the performance for different encodings. The main objective is to develop a flexible GA with few user-defined parameters.

In self-adapting GA parameters, regardless of the adopted encoding, the chromosome must be modified to accommodate genes encoding parameters' values that will be fine-tuned during the evolutionary process. Considering the parameters that will be changed, each individual is codified into a chromosome with $n+p$ genes, where n and p are, respectively, the number of genes that encode the

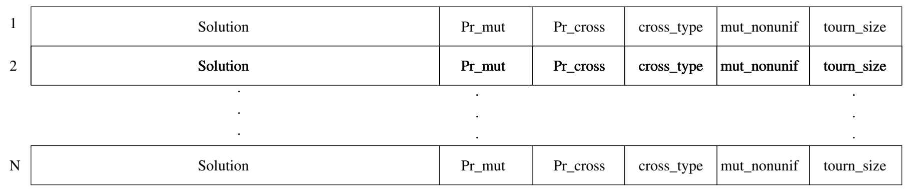

Fig. 1. Self-adapting: encoding aspects

problem solution and those that encode parameter values. In the population N individuals, they are encoded with a single chromosome, as shown in Figure 1.

In figure 1 the strategic parameters that will be adjusted by means of evolution are: mutation rate (Pr_mut), crossover rates (Pr_cross), crossover type ($cross\_type$), mutation step (for real encoding) ($mut\_nonunif$), and the tournament size ($tour\_size$), resulting in $p = 5$ genes. Mutation rate, mutation step and crossover rate are applied at individual level, whereas the tournament size and crossover type are determined by the mean among all individuals and are applied at population level. To compare the behavior of parameters adaptation in continuous and discrete search spaces, two encoding schemes will be analyzed: binary and real encoded chromosomes.

During the evolutionary process, genetic operators treat indistinctly all genes of the chromosome, despite what it encodes (part of the solution or any strategic parameter). For crossover, a N-length random vector is generated, where N is the population size and each element of the vector is within $[0, 1]$. Each value of this vector is compared with the crossover rate encoded in an individual. If it is lower, the individual is selected for crossover. For mutation, a $N \times (n+p)$ matrix is generated with random values. For each individual there is a corresponding line of $n + p$ random numbers in the matrix, which are compared with the mutation rate encoded in the individual. If it is lower, the associated gene of the individual will undergo mutation. Both crossover type and tournament size are chosen based on the frequency of the corresponding value in the population. The most frequent values are accepted as parameter for the next generation. It is important to note that with this encoding, whenever an individual has the genes that encode the strategic parameters modified by genetic operators (but not the solution genes), the corresponding fitness does not need to be re-calculated.

4 Experiments, Results and Discussion

This paper aims to analyze the performance of the self-adapting approach when applied to a benchmark of a well-known combinatorial optimization problem: the multiple knapsack problem (MKP), that is a generalization of the simple 0/1 knapsack problem [28]. The 0/1 KP involves selecting from among various items those that will be most profitable, given the knapsack has limited capacity. The 0/1 MKP involves m knapsacks of capacities c_1, c_2, $\cdots$, c_m. Every object selected must be placed in all m knapsacks, although neither the weight of an object o_j nor its profit is fixed, and, probably, they will have different values in each knapsack (for additional details see [29]).

In our implementation, each gene that encodes the problem solution should indicate the presence or absence of an item in all knapsacks. For binary encoding, each gene corresponds to a single bit, and for real encoding, each gene corresponds to a number in the range [0..1] that is rounded to the closest integer, as shown in figure 2.

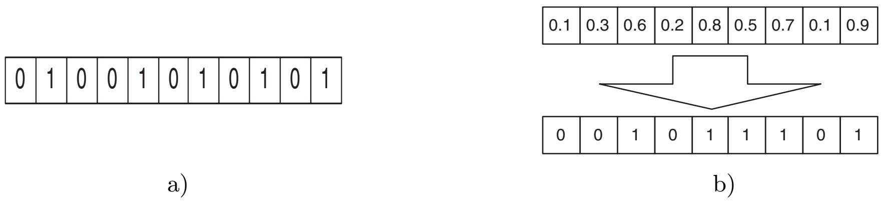

Fig. 2. Solution encoding using: a) binary encoding, b) real encoding

When using binary encoding, the search space (ss) is defined by the number of items i as $ss = 2^i$. Although the number of knapsacks does not influence the search space, a large number of knapsacks implies in more complexity in computing the fitness function.

For analyzing the behavior of the algorithm in different difficulty levels, simulations were done for nine MKP problems, detailed in Table 1.

Table 1. Characteristics of MKP problems used in this work

Problem	Items	Knapsacks	Optimal solution
Weing7	105	2	1095445
Pb6	40	30	776
Pet6	39	5	10618
Weish18	70	5	9580
Weish22	80	5	8947
Weish26	90	5	9584
Flei	20	10	2139
Hp2	35	4	3186
Sent01	60	30	7772

In our experiments, the value of GA parameters were based on those recommended by De Jong [9], and were set as follows:

- Number of generations: 2000;
- Population size (N): 100 individuals;

- Crossover rate: $Pr\_cross \in [0.4, 1]$ for adaptive approach and $Pr\_cross = 0.6$ for fixed approach;
- Mutation rate: $Pr\_mut \in [0.0001, 0.3]$ for adaptive approach and $Pr\_mut = 0.001$ for fixed approach;
- Tournament size: $s \in \{2, \ldots, 5\}$ for adaptive approach and $s = 5$ for fixed approach;
- Crossover type: one-point and uniform for adaptive approach and one-point for fixed approach;
- Stop criterion: Maximum number of generations.

Table 2 compares the performance of our self-adapting parameters algorithm (self-adapt) and fixed parameters algorithm (fixed) for all problems described in Table 1. As shown in Figure 2, for real encoding, rounding to the closest integer is used to transform a real-valued chromosome to a binary string.

Self-adaptation is based in the expectation that the best strategic parameters will be able to produce more adapted individuals. Table 2 shows the results of self-adaptation in both cases, real and binary encoding. The problems were evaluated considering 500 runs [1] and the problem files were taken from [31]. For each problem instance, we evaluated the distribution tendency [2]. Besides, we also presented the best individual found in all runs (Best) and when this individual has been found (Gen_Best).

Table 2 clearly shows that the self-adapt approach out-performed the fixed approach, independently of the encoding scheme. Therefore, we can answer the first question pointed in the Introduction: "Does real-valued encoding take any benefit for a discrete problem using parameter control in a GA?". For the discrete problems dealt in this work real encoding does seems to be less appropriate than binary encoding. The hypothesis that making transitions between values of genes smoother before rounding (using real values instead of discrete ones) can facilitate GA to find better solutions was not confirmed.

To better investigate the effects of self-adaptation in the system performance, Table 3 shows normalized results [3] considering binary encoding and different levels of system autonomy. Results from [32] that self-adapts mutation rates are also presented.

It is important to point out that, due to the penalties applied to the fitness function, all the solutions obtained from approaches with adaptation were feasible. It is a strong restriction that could be relaxed in the future to improve the system performance. In [32], for instance, they reported the elimination of all unfeasible solutions. As can be noted, the version with the highest level of autonomy

[1] Different random seeds were generated using L'Ecuyer with Bays-Durham shuffling [30]

[2] Measures of central tendency are measures of the location of the middle or the center of a distribution. For symmetric distributions, mean and median are the same. In general, the mean will be higher than the median for positively skewed distributions and less than the median for negatively skewed distributions.

[3] The optimal known value (profit=1095445) for the Weing7 problem is the reference value 1.00

Table 2. Results for multi knapsack 0/1 optimization problems

Problem	Encoding	Adaptation	Mean	Median	SD	Best	Gen(Best)
Weing7	Binary	Self-Adapt	1095264	1092466	48892.98	1095445	231
		Fixed	1075147	1071868	48891.31	1091327	1721
	Real	Self-Adapt	1079962	1083937	49065.43	1079880	63
		Fixed	1034557	1038552	50443,06	1094957	9
Pb6	Binary	Self-Adapt	714.0699	729	62.27475	776	24
		Fixed	407.0459	405	104.9299	657	57
	Real	Self-Adapt	681.1497	704	79.87887	776	17
		Fixed	470.5289	468	91.4316	745	3751
Pet6	Binary	Self-Adapt	10468.01	10504	474.841	10618	34
		Fixed	8749.058	8271	992.6105	10396	1328
	Real	Self-Adapt	10460.21	10496	475.6101	10618	19
		Fixed	10152.5	10201	497.6526	10584	5526
Weish18	Binary	Self-Adapt	9510.048	9548	433.288	9580	37
		Fixed	8102.982	8157	534.998	9109	1915
	Real	Self-Adapt	9218.266	9282	463.818	9580	61
		Fixed	7535.108	7556	641.971	8938	46
Weish22	Binary	Self-Adapt	8834.535	8857	400.092	8947	924
		Fixed	7435.112	7458	514.995	8589	1407
	Real	Self-Adapt	8239.800	8334	540.689	8929	45
		Fixed	5735.299	5764	810.009	7629	37
Weish26	Binary	Self-Adapt	9493.311	9539	428.442	9584	97
		Fixed	8115.074	8130	515.070	9117	1865
	Real	Self-Adapt	8716.764	8840	657.194	9533	55
		Fixed	6080.405	6120	867.144	8429	62
Flei	Binary	Self-Adapt	2067.920	2068	94.8643	2139	8
		Fixed	1944.948	1956	102.612	2059	60
	Real	Self-Adapt	2052.653	2059	98.2860	2139	1
		Fixed	1918.399	1922	117.593	2139	1
Hp2	Binary	Self-Adapt	3080.966	3089	149.404	3186	18
		Fixed	2855.920	2868	158.267	3119	13
	Real	Self-Adapt	3038.818	3048	148.173	3169	19
		Fixed	2894.940	2910	161.950	3157	29
Sent01	Binary	Self-Adapt	7622.100	7698	383.915	7772	44
		Fixed	4559.451	4573	816.850	6698	1939
	Real	Self-Adapt	7111.149	7295	650.846	7772	36
		Fixed	3843.950	3758	973.846	6505	42

(self-adapt approach) out-performed the other approaches with a high degree of confidence (see t-Student test column [4]), confirming that the self-adaptation is a good alternative to release user from arbitrarily defining evolutionary parameters. By answering the second question "How the performance of the GA is

[4] t-Student tests were performed comparing all the approaches with the fixed one.

Table 3. Normalized results for Weing7 problem

Adaptation	Mean	Median	SD	Best	Gen(Best)	t-Student
Self-Adapt	0.99983	0.99728	0.044633	1.000000	231	1.0e-190
Kimbrough [32]	0.99934	-	0.000364	-	-	-
Mutation	0.99906	0.99638	0.044590	1.000000	375	2.8e-183
Crossover/crossover-type	0.98897	0.98581	0.044439	0.99866	1591	1.64e-40
Tournament size	0.98235	0.97883	0.044704	0.99719	945	0.27
Fixed	0.98147	0.97848	0.044631	0.99624	1721	-

affected by the adaptation in their parameters?", results show that almost all forms of adaptation resulted in a GA with better performance when compared with the fixed one. Also, the self-adaptation of mutation plays a main role when compared with the remaining parameters.

Regarding the third question: "Is there any advantage for the user using parameter control in a GA?", the better results found with self-adapting parameters point out a clear advantage. Nevertheless, it could be argued that the final results using self-adaptation of (only) mutation are not striking better than using self-adaptation of all parameters together. In fact this is not really the point because it should be taken into account the fact that having less parameters to adjust, user can focus in understanding/tuning the remaining ones.

To analyze the behavior of evolutionary operators during the process, Figure 3 depicts the evolution of crossover and mutation rates of the self-adaptive method. It is also shown the evolution of fitness for both methods [5].

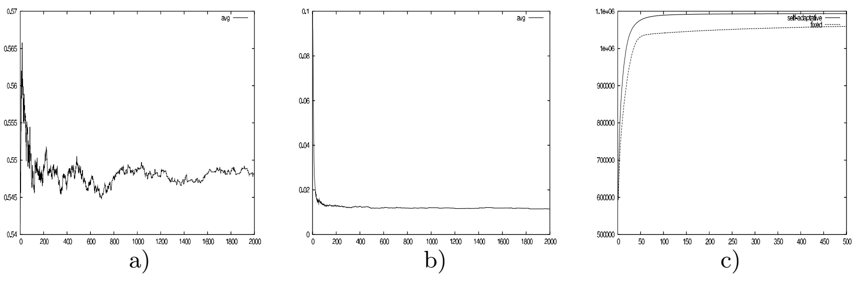

Fig. 3. Evolution of: a) Crossover Rate, b) Mutation Rate, c) Fitness

It can be noted in Figure 3 that the oscillation of crossover rate is higher during the early stages of the evolutionary process, and observe the convergence

[5] Since the fitness evolution has not shown significant improvement after 500 generations, the following generations were omitted from the graphics

of mutation rates to small values at the end of evolution. This fact confirms the hypothesis that using fixed parameters during the evolutionary process is not adequate for all the problems solved by GAs.

5 Conclusions

This paper presented a method for self-adapting several parameters associated with the evolutionary process of a GA. After answering the questions asked in the Introduction, we conclude that the proposed approach has shown a good performance with a high degree of autonomy, where only few evolutionary parameters have to be defined by the user. Also, we verified that, for the MKP instances tested, the search in a continuous space resulted in a worse performance when compared with the search in a binary space. So, the possible benefits associated with the fact of producing more robust algorithms (which can perform independently in continuous or discrete spaces) are reduced in performance.

Finally, simulation results confirmed that self-adaptation is an interesting alternative in the search for parameterless heuristic optimization systems, since it is able to generate models that explore the space of parameters looking for the best ones at the same time as well as it searches for the problem's solution. Results obtained encourage further research towards a fully self-adaptive GA, relieving user from the burden of adjusting parameters for each problem.

Acknowledgements. The authors would like to thank the anonymous referees for their helpful suggestions. Thanks to Fundação Araucária/Funcefet-PR 018/2004 for the financial support, for CNPq for a grant to M.H.Maruo, and CNPq for a research grant to H.S.Lopes, process 350053/03-0.

References

1. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation **3** (1999)
2. Rojas, I., Gonzalez, J., Pomares, H., Merelo, J., Castillo, P., Romero, G.: Statistical analysis of the main parameters involved in the design of a genetic algorithm. IEEE Transactions on Systems Man and Cybernetics **32** (2002) 31–37
3. Angeline, P.J.: Adaptive and self-adaptive evolutionary computation. In M., P., Attikiouzel, Y., Marks, R., D., F., Fukuda, T., eds.: Computational Intelligence, A Dynamic System Perspective, IEEE Press (1995) 152–161
4. Hinterding, R., Michalewicz, Z., Eiben, A.E.: Adaptation in evolutionary computation: A survey. In: Proceedings of the 4th IEEE International Conference on Evolutionary Computation. (1997) 65–69
5. Smith, J., Fogarty, T.C.: Operator and parameter adaptation in genetic algorithms. Soft Computing **1** (1997) 81–87
6. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer (2003)
7. Davis, L., ed.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, New York (1991)

8. Spears, W.M.: Adapting crossover in evolutionary algorithms. In: Proceedings of the Fourth Annual Conference on Evolutionary Programming, San Diego, CA (1995)
9. De Jong, K.: The Analysis of the behaviour of a Class of Genetic Adaptive systems. PhD thesis, Department of Computer Science, University of Michigan (1975)
10. Grefenstette, J.J.: Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics **16** (1986) 122–128
11. Schaffer, J.D., Morishima, A.: An adaptive crossover distribution mechanism for genetic algorithms. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, Lawrence Erlbaum Associates, Inc. (1987) 36–40
12. Bremermann, H.J., Rogson, M., Salaff, S.: Global properties of evolution processes. In Pattee, H.H., Edlsack, E.A., Fein, L., Callahan, A.B., eds.: Natural Automata and Useful Simulations. Spartan Books, Washington D.C. (1966) 3–41
13. Smith, J., Fogarty, T.C.: Self-adaptation of mutation rates in a steady-state genetic algorithm. Proceedings of IEEE International Conference on Evolutionary Computation (1996) 318–323
14. Bäck, T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press (1996)
15. Fogarty, T.C.: Varying the probability of mutation in the genetic algorithm. In: Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc. (1989) 104–109
16. Bäck, T.: Self-adaptation in genetic algorithms. In Varela, F.J., Bourgine, P., eds.: Proc. of the 1st European Conf. on Artificial Life, Cambridge, MA, MIT Press (1992) 227–235
17. Thierens, D., Goldberg, D.E.: Mixing in genetic algorithms. In: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc. (1993) 38–47
18. Schaffer, J.D., Eshelman, L.J.: On crossover as an evolutionary viable strategy. In Belew, R., Booker, L., eds.: Proceedings of the Fourth International Conference on Genetic Algorithms. (1991) 61–68
19. Eshelman, L.J., Caruana, R., Schaffer, J.D.: Biases in the crossover landscape. In: Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc. (1989) 10–19
20. Sywerda, G.: Uniform crossover in genetic algorithms. In: Proceedings of the third international conference on Genetic algorithms, Morgan Kaufmann Publishers Inc. (1989) 2–9
21. Spears, W.M., De Jong, K.: On the virtues of parameterized uniform crossover. In Belew, R.K., Booker, L.B., eds.: Proc. of the Fourth Int. Conf. on Genetic Algorithms, San Mateo, CA, Morgan Kaufmann (1991) 230–236
22. De Jong, K., Spears, W.M.: A formal analysis of the role of multi-point crossover in genetic algorithms. Annals of Mathematics and Artificial Intelligence **5** (1992) 1–26
23. Spears, W.M.: Crossover or mutation? In Whitley, L.D., ed.: Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA (1993) 221–237
24. De Jong, K., Spears, W.M.: An analysis of the interacting roles of population size and crossover in genetic algorithms. In Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1. Volume 496., Dortmund, Germany, Springer-Verlag, Berlin, Germany (1991) 38–47

25. Spears, W.M., Anand, V.: A study of crossover operators in genetic programming. In Ras, Z.W., Zemankova, M., eds.: Proceedings of the Sixth International Symposium on Methodologies for Intelligent Systems ISMIS 91, Springer-Verlag (1991) 409–418
26. Eiben, A.E.: 3.7. In: Multi-parent Recombination. IOP Publishing Ltd and Oxford University Press (1995)
27. Eiben, A.E.: Multiparent recombination in evolutionary computing. Advances in evolutionary computing: theory and applications (2003) 175–192
28. Mumford, C.L.: Comparing representations and recombination operators for the multi-objective 0/1 knapsack problem. In: CEC2003: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE. (2003) 854–861
29. Thierens, D.: Adaptive mutation rate control schemes in genetic algorithms. In: CEC2002: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE. (2002) 980–985
30. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press (1992)
31. Khuri, S., Bäck, T., Heitkötter, J.: SAC 94: suite of 0/1 multiple knapsack problem instances, http://elib.zib.de/pub/packages/mp-testdata/ip/sac94-suite (1994)
32. Kimbrough, S.O., Lu, M., Wood, D.H., Wu, D.J.: Exploring a two-market genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers Inc. (2002) 415–422