Adaptation in Evolutionary Computation: A Survey

Robert Hinterding, Zbigniew Michalewicz, and Agoston E. Eiben

Abstract— **Adaptation of parameters and operators is one of the most important and promising areas of research in evolutionary computation; it tunes the algorithm to the problem while solving the problem. In this paper we develop a classification of adaptation on the basis of the mechanisms used, and the level at which adaptation operates within the evolutionary algorithm. The classification covers all forms of adaptation in evolutionary computation and suggests further research.**

I. Introduction

As evolutionary algorithms (EAs) implement the idea of evolution, and as evolution itself must have evolved to reach its current state of sophistication, it is natural to expect adaptation to be used not only for finding solutions to a problem, but also for tuning the algorithm to the particular problem.

In EAs, we not only need to choose the algorithm, representation, and operators for the problem, but we also need to choose parameter values and operator probabilities for the evolutionary algorithm so that it will find the solution and, what is also important, find it efficiently. This process of finding appropriate parameter values and operator probabilities is a time-consuming task and considerable effort has gone into automating this process.

Researchers have used various ways of finding good values for the strategy parameters as these can affect the performance of the algorithm in a significant way. Many researchers experimented with various problems from a particular domain, tuning the strategy parameters on the basis of such experimentation (tuning "by hand"). Later, they reported their results of applying a particular EA to a particular problem, stating:

> For these experiments, we have used the following parameters: population size = 80, probability of crossover = 0.7, etc.

without much justification of the choice made.

Note that (a run of) an EA is an intrinsically dynamic, adaptive process. The use of rigid, i.e. constant, parameters is thus in contrast to the general evolutionary spirit. Besides, there are also technical drawbacks to the traditional approach:

- the users' mistakes in setting the parameters can be sources of errors and/or sub-optimal performance;
- parameter tuning costs a lot of time;
- the optimal parameter value may vary during the evolution.

Therefore it is a natural idea to try to modify the values of strategy parameters [1] during the run of the algorithm. It is possible to do this by using some (possibly heuristic) rule, by taking feedback from the current state of the search, or by employing some self-adaptive mechanism. Note that these changes may affect a single component of a chromosome, the whole chromosome (individual), or even the whole population. Clearly, by changing these values while the algorithm is searching for the solution of the problem, further efficiencies can be gained.

Self-adaptation, based on the evolution of evolution, was developed in Evolution Strategies to adapt mutation parameters to suit the problem during the run. The method was very successful in improving efficiency of the algorithm for some problems. This technique has been extended to other areas of evolutionary computation, but fixed representations, operators, and control parameters are still the norm.

Other research areas based on the inclusion of adapting mechanisms are the following.

- Representation of individuals (as proposed by Shaefer [32]; the Dynamic Parameter Encoding technique, Schraudolph & Belew [29] and messy genetic algorithms, Goldberg et al.[16] also fall into this category).
- Operators. It is clear that different operators play different roles at different stages of the evolutionary process. The operators should adapt (e.g., adaptive crossover, Schaffer & Morishima [27], Spears [34]). This is true especially for time-varying fitness landscapes.
- Control parameters. There have been various experiments aimed at adaptive probabilities of operators [7], [22], [35], [36]. However, much more remains to be done.

In this paper we develop a comprehensive classification of adaptation and give examples of their use. The classification is based on the mechanism of adaptation and level (in the EA) it occurs. Such a classification can be useful to the evolutionary computation community, since many researchers use the terms "adaptation" or "self-adaptation" in an arbitrary way; in a few instances some authors (including ourselves!) used the term "self-adaptation" where there was a simple (deterministic and heuristic) rule for changing some parameter of the process.

The paper is organised as follows: the next section we develop classification of adaptation in Evolutionary Algo-

R. Hinterding is with the Department of Computer and Mathematical Sciences, Victoria University of Technology, PO Box 14428 MMC, Melbourne 3000, Australia. email: rhh@matilda.vut.edu.au

Z. Michalewicz is with the Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA, *and* with Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland. email:zbyszek@uncc.edu

A.E. Eiben is with the Department of Computer Science, Leiden University, Leiden, The Netherlands. email:gusz@wi.leidenuniv.nl

[1] By strategy parameters, we mean the parameters of the EA, not those of the problem

Type / *Level*	Static	Dynamic		
		Deterministic	Adaptive	Self-adaptive
Environment	S	E-D	E-A	E-SA
Population	S	P-D	P-A	P-SA
Individual	S	I-D	I-A	I-SA
Component	S	C-D	C-A	C-SA

TABLE I
Classification of adaptation in EAs

rithms (EAs). Section III looks at types of adaptation, whereas Section IV — at the levels of adaptation. Section V discusses the combination of types and levels of adaptation and Section VI presents the discussion and conclusion.

II. Classification of Adaptation

The action of determining the variables and parameters of an EA to suit the problem has been termed *adapting the algorithm* to the problem, and in EAs this can be done while the algorithm is searching for a problem solution.

We give classifications of adaptation in Table I; this classification is based on the mechanism of adaptation (*adaptation type*) and on which level inside the EA adaptation occurs (*adaptation level*). These classifications are orthogonal and encompass all forms of adaptation within EAs. Angeline's classification [1] is from a different perspective and forms a subset of our classifications.

The *Type* of adaptation consists of two main categories: static (no change) and dynamic, with the latter divided further into deterministic (D), adaptive (A), and self-adaptive (SA) mechanisms. In the following section we discuss these types of adaptation.

The *Level* of adaptation consists of four categories: environment (E), population (P), individual (I), and component (C). These categories indicate the scope of the changed parameter; we discuss these types of adaptation in Section IV.

Whether examples are discussed in Section III or in Section IV is completely arbitrary. An example of adaptive individual level adaptation (I-A) could have been discussed in Section III as an example of adaptive dynamic adaptation or in Section IV as an example of individual level of adaptation.

III. Types of Adaptation

The classification of the *type of adaptation* is made on the basis of the mechanism of adaptation used in the process; in particular, attention is paid to the issue of whether or not a feedback from the EA is used.

A. Static

Static adaptation is where the strategy parameters have a constant value throughout the run of the EA. Consequently, an external agent or mechanism (e.g., a person or a program) is needed to tune the desired strategy parameters and choose the most appropriate values. Typically this happens by running numerous tests and trying to find a link between parameter values and EA performance. This method is commonly used for most of the strategy parameters.

De Jong [9] put considerable effort into finding parameter values which were good for a number of numeric test problems using a traditional GA. He determined experimentally recommended values for the probability of using single-point crossover and bit mutation. Grefenstette [17] used a GA as a meta-algorithm to optimise some of the parameter values.

B. Dynamic

Dynamic adaptation happens if there is some mechanism which modifies a strategy parameter without external control. The class of EAs that use dynamic adaptation can be sub-divided further into three classes where the *mechanism of adaptation* is the criterion.

B.1 Deterministic

Deterministic dynamic adaptation takes place if the value of a strategy parameter is altered by some deterministic rule; this rule modifies the strategy parameter deterministically without using any feedback from the EA. Usually, a time-varying schedule is used, i.e. the rule will be used when a set number of generations have elapsed since the last time the rule was activated.

This method of adaptation can be used to alter the probability of mutation so that the probability of mutation changes with the number of generations. For example:

$$p_m = 0.5 - 0.3 \cdot \frac{g}{G},$$

where g is the generation number from $1 \ldots G$. Here the mutation probability $mut\%$ will decrease from 0.5 to 0.2 as the number of generations increases to G. Early examples of this approach are the varying mutation rates as used by Fogarty [13], or Hesser & Männer [18] in GAs. This method of adaptation was used also in defining a mutation operator for floating-point representations [24]: non-uniform mutation. For a parent $\vec{x}$, if the element x_k is selected for this mutation, the result is $\vec{x}' = (x_1, \ldots, x'_k, \ldots, x_n)$, where

$$x'_k = \begin{cases} x_k + \triangle(t, right(k) - x_k) \\ \qquad \text{if a random binary digit is 0} \\ x_k - \triangle(t, x_k - left(k)) \\ \qquad \text{if a random binary digit is 1.} \end{cases}$$

The function $\triangle(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\triangle(t, y)$ being close to 0 increases as t increases (t is the generation number). This property causes this operator to search the space uniformly initially (when t is small), and very locally at later stages.

Deterministic dynamic adaptation was also used for changing the objective function of the problem. For constrained problems it can be applied by increasing the penalties for violated constraints with evolution time [21], [25]. Joines & Houck used the following formula:

$$F(\vec{x}) = f(\vec{x}) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(\vec{x}),$$

whereas Michalewicz & Attia experimented with

$$F(\vec{x}, \tau) = f(\vec{x}) + \frac{1}{2\tau} \sum_{j=1}^{m} f_j^2(\vec{x}).$$

In both cases, functions f_j measure the violation of the j-th constraint. Eiben & Ruttkay [10] described an implementation of an evolutionary algorithm for constraint satisfaction problems, where the penalty coefficients of violated constraints were increased after each run and used in a following run on the same problem.

B.2 Adaptive

Adaptive dynamic adaptation takes place if there is some form of feedback from the EA that is used to determine the direction and/or magnitude of the change to the strategy parameter. The assignment of the value of the strategy parameter may involve credit assignment, and the action of the EA may determine whether or not the new value persists or propagates throughout the population.

Early examples of this type of adaptation include Rechenberg's '1/5 success rule' in Evolution Strategies, which was used to vary the step size of mutation [26]. This rule states that the ratio of successful mutations to all mutations should be 1/5, hence if the ratio is greater than 1/5 then increase the step size, and if the ratio is less than 1/5 then decrease the step size. An example for GAs is Davis's 'adaptive operator fitness': where feedback on the success of a larger number of reproduction operators is utilised to adjust their probability of being used [8]. Julstrom's adaptive mechanism regulates the ratio between crossovers and mutations based on their performance [22]. An extensive study of this kind of "learning-rule" mechanisms was done by Tuson & Ross [36].

Adaption was also used to change the objective function by increasing or decreasing penalty coefficients for violated constraints. For example, Bean & Hadj-Alouane [4] designed a penalty function where its one component takes a feedback from the search process. Each individual is evaluated by the formula:

$$F(\vec{x}) = f(\vec{x}) + \lambda(t) \sum_{j=1}^{m} f_j^2(\vec{x}),$$

where $\lambda(t)$ is updated every generation t in the following way:

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t), \\ \qquad \text{if } \vec{b}(i) \in \mathcal{F} \text{ for all} \\ \qquad t-k+1 \le i \le t \\ \beta_2 \cdot \lambda(t), \\ \qquad \text{if } \vec{b}(i) \in \mathcal{S} - \mathcal{F} \text{ for all} \\ \qquad t-k+1 \le i \le t \\ \lambda(t), \text{ otherwise}, \end{cases}$$

where $\vec{b}(i)$ denotes the best individual, in terms of function *eval*, in generation i, $\beta_1, \beta_2 > 1$ and $\beta_1 \neq \beta_2$ (to avoid cycling). In other words, the method (1) decreases the penalty component $\lambda(t+1)$ for the generation $t+1$, if all best individuals in the last k generations were feasible, and (2) increases penalties, if all best individuals in the last k generations were infeasible. If the best individuals in the last k generations contains both feasible and infeasible solutions, then $\lambda(t+1)$ is not changed.

Recent work of Eiben & van der Hauw on solving (discrete) constraint satisfaction problems is also based on an adaptive penalty technique that periodically increases the penalty of those constraints that are violated. This mechanism highly improved GA performance on 3-SAT problems, [12], and on graph 3-colouring problems [11].

Other examples include adaptation of probabilities of eight operators for adaptive planner/navigator [38], where the feedback from the evolutionary process includes, through the operator performance index, effectiveness of operators in improving the fitness of a path, their operation time, and their side effect to future generations.

B.3 Self-adaptive

The idea of the evolution of evolution can be used to implement the self-adaptation of parameters. Here the parameters to be adapted are encoded onto the chromosome(s) of the individual and undergo mutation and recombination. These encoded parameters do not affect the fitness of individuals directly, but "better" values will lead to "better" individuals and these individuals will be more likely to survive and produce offspring and hence propagate these "better" parameter values.

Schwefel [30], [31] developed this method to self-adapt the mutation step size and the mutation rotation angles in Evolution Strategies. Theoretical analysis of σ-control (and the 1/5-Rule) done by Beyer can be found in [5]. Self-adaptation was extended to EP by Fogel et al. [14] and to GAs by Bäck [3], Hinterding [19] and Smith & Fogarty [33].

The parameters to self-adapt can be parameter values that control the operation of the EA, values that control the operation of reproduction or other operators, or probabilities of using alternative processes, and as these are numeric quantities this type of self-adaptation has been used mainly for the optimisation of numeric functions. This has been the case when single chromosome representations are used (which is the overwhelming case), as otherwise numerical and non-numerical representations would need to be combined on the same chromosome. Examples of self-adaptation for non-numerical problems are Fogel et al. [15] where they self-adapted the relative probabilities of five mutation operators for the components of a finite state machine. The other example is Hinterding [?], where a multi-chromosome GA is used to implement the self-adaptation in the Cutting Stock Problem with contiguity. Here self-adaptation is used to adapt the probability of using one of the two available mutation operators, and the strength of the group mutation operator.

IV. Levels of Adaption

We can also define at what level within the EA and the solution representation adaptation takes place. We define four levels: environment, population, individual, and component. These levels of adaptation can be used with each of the types of adaptation, and a mixture of levels and types of adaptation can be used within an EA.

A. Environment Level Adaption

Environment level adaptation is where the response of the environment to the individual is changed. This covers cases such as when the penalties in the fitness function change, where weights within the fitness function change and the fitness of an individual changes in response to niching considerations (some of these were discussed in the previous section, in the context of types of adaptation).

Darwen & Yao [6], explore both deterministic environmental adaptation and adaptive environmental adaptation in their paper comparing fitness sharing methods.

B. Population Level Adaption

In EAs some (or all in simple EAs) of the parameters are global, modifying these parameters when they apply to all members of the population is *population level adaptation*.

Dynamic adaptation of these parameters is in most cases deterministic or adaptive. No cases of population level self-adaptation have been seen yet. The example of deterministic modification of the mutation rate given above is deterministic population level adaptation, and Rechenberg's '1/5 success rule' is an example of adaptive population level adaptation.

Population level adaptation also covers cases where a number of populations are used in a parallel EA or otherwise, Lis [23] uses feedback from a number of parallel populations to dynamically adapt the mutation rate. The feedback from populations with different mutation probabilities was used to adjust the mutation probabilities of all the populations up or down. Schlierkamp-Voosen & Mühlenbein [28] use competition between sup-populations to determine which populations will lose or gain individuals. Hinterding et al.[20] use feedback from three sub-populations with different population sizes to adaptively change some or all of the sub-population sizes.

C. Individual Level Adaption

Individual level adaptation adjusts strategy parameters held within individuals and whose value affects only that individual. Examples are: the adaptation of the mutation step size in ESs, EP, and GAs; the adaptation of crossover points in GAs [27] and [37].

Arabas et al.[2] describe a method for adapting population size by defining age of individuals; the size of the population after single iteration is

$$PopSize(t+1) = PopSize(t) + N(t) - D(t),$$

where $D(t)$ is the number of chromosomes which die off during generation t and $N(t)$ is the number of offspring produced during the generation t (for details, see Michalewicz [24]). The number of produced offspring $N(t)$ is proportional to the size of the population at given generation t, whereas the number of individuals "to die" $D(t)$ depends on age of individual chromosomes. There are several heuristics one can use for the age allocation for individuals [2]; all of them require a feedback from the current state of the search.

D. Component Level Adaption

Component-level adaptation adjusts strategy parameters local to some component or gene of an individual in the population. The best known example of component level adaptation is the self-adaptation of component level mutation step sizes and rotation angles in ESs.

Additionally, in Fogel et al.[15] the mechanism of adapting probabilities of mutation for each component of a finite states machine is discussed.

V. Combining forms of adaptation

The classic example of combining forms of adaptation is in ESs, where the algorithm can be configured for individual level adaptation (one mutation step size per individual), component level adaptation (one mutation step size per component) or with two types of component level adaptation where both the mutation step size and rotation angle is self-adapted for individual components [30].

Hinterding et al.[20] combine global-level adaptation of the population size with individual level self-adaptation of the mutation step size for optimising numeric functions.

Combining forms of adaptation has not been used much as the interactions are complex, hence deterministic or adaptive rules will be difficult to work out. But self-adaptation where we use evolution to determine the beneficial interactions (as in finding solutions to problems) would seem to be the best approach.

VI. Discussion

The effectiveness of evolutionary computation depends on the interaction of representation used for the problem solutions, the reproduction operators used, and the configuration of the evolutionary algorithm used.

Adaption provides the opportunity to customise the evolutionary algorithm to the problem and to modify the configuration and the strategy parameters used while the problem solution is sought. This enables us to not only incorporate domain information and multiple reproduction operators into the EA more easily, but can allow the algorithm itself to select those values and operators which give better results. Also these values can be modified during the run of the EA to suit the situation during that part of the run.

Information about which of the operators available are most suitable to a particular problem is not easily determined, adaptation can be used here to provide feedback or to determine when they should be used.

More research on the combination of the types and levels of adaptation needs to be done as this could lead to significant improvements to finding good solutions and the speed of finding them.

References

[1] P.J. Angeline. Adaptive and self-adaptive evolutionary computation. In M. Palaniswami, Y. Attikiouzel, R.J.II Marks, D. Fogel, and T. Fukuda, editors, *Computational Intelligence, A Dynamic System Perspective.* IEEE Press, 1995. pp.152–161.

[2] J. Arabas, Z. Michalewicz, and J. Mulawka. GAVaPS — a genetic algorithm with varying population size. In *Proceedings of the First IEEE Conference on Evolutionary Computation.* IEEE Press, 1994. pp. 73–78.

[3] T. Bäck. Self-adaption in genetic algorithms. In *Proceedings of the First European Conference on Artificial Life*, Cambridge, 1992. MIT Press. pp. 263–271.

[4] J.C. Bean and A.B. Hadj-Alouane. A dual genetic algorithm for bounded integer programs. Tr 92-53, Department of Industrial and Operations Engineering, The University of Michigan, 1992.

[5] H.-G. Beyer. Toward a theory of evolution strategies. *Evolutionary Computation*, 1(2):165–188, 1993.

[6] P Darwen and X. Yao. Every niching mehtod has its niche: Fitness sharing and implicit sharing compared. In H-M. Voigt, W. Ebeling, I. Rechenberg, and H-P. Schwefel, editors, *Parellel Problem Solving from Nature – PPSV IV*, volume 1141 of *Lecture Notes in Computer Science.* Springer, Berlin, 1996. pp. 398–407.

[7] L. Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms.* Morgan Kaufmann, pp.61–69 1989.

[8] L. Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[9] K. A. De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive System.* Doctoral dissertation, University of Michigan, 1975. Dissertation Abstract International, 36(10), 5140B. (University Microfilms No 76-9381).

[10] A.E. Eiben and Zs. Ruttkay. Self-adaptivity for constraint satisfaction: Learning penalty functions. In *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation.* IEEE Press, 1996. pp. 258–261.

[11] A.E. Eiben and J.K. van der Hauw. Graph coloring with adaptive evolutionary algorithms. Technical Report TR-96-11, Leiden University, August 1996. also available as http://www.wi.leidenuniv.nl/~gusz/graphcol.ps.gz.

[12] A.E. Eiben and J.K. van der Hauw. Solving 3-SAT by GAs adapting constraint weights. In *Proceedings of the 4th IEEE International Conference on Evolutionary Computation.* IEEE Press, 1997.

[13] T. Fogarty. Varying the probability of mutation in the genetic algorithm. In J.D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 104–109. Morgan Kaufmann, 1989.

[14] D.B. Fogel, L.J. Fogel, and J.W. Atmar. Meta-evolutionary programming. In R.R. Chen, editor, *Proc. of the 25th Asilomar Conf. on Signals, Systems, and Computers*, San Jose, 1991. Maple Press. pp. 540–545.

[15] L.J. Fogel, P.J. Angeline, and D.B. Fogel. An evolutionary programming approach to self-adaption on finite state machines. In J.R. McDonnell, R.G. Reynolds, and D.B. Fogel, editors, *Proceedings of the Forth Annual Conference on Evolutionary Programming*, Massachusetts, 1995. MIT Press. pp. 355–365.

[16] D.E. Goldberg, K. Deb, and B. Korb. Do not worry, be messy. In *Proceedings of the 4th International Conference on Genetic Algorithms.* Morgan Kaufmann, 1991. pp. 24–30.

[17] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16(1):pp. 122–128, 1986.

[18] J. Hesser and R. Männer. Towards an optimal mutation probability for genetic algorithms. In H.-P. Schwefel and R. Männer, editors, *Proceedings of the 1st Conference on Parallel Problem Solving from Nature*, number 496 in Lecture Notes in Computer Science, pages 23–32. Springer-Verlag, 1990.

[19] R. Hinterding. Gaussian mutation and self-adaption in numeric genetic algorithms. In *IEEE International Conference on Evolutionary Computation.* IEEE Press, 1995. pp 384–389.

[20] R. Hinterding, Z. Michalewicz, and T.C. Peachey. Self-adaptive genetic algorithm for numeric functions. In H-M. Voigt, W. Ebeling, I. Rechenberg, and H-P. Schwefel, editors, *Parellel Problem Solving from Nature – PPSV IV*, volume 1141 of *Lecture Notes in Computer Science*, Berlin, 1996. Springer. pp. 420–429.

[21] J.A. Joines and C.R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. In *Proceedings of the First IEEE Conference on Evolutionary Computation.* IEEE Press, 1994. pp. 579–584.

[22] B.A. Julstrom. What have you done for me lately? adapting operator probabilities in a steady-state genetic algorithm. In *Proceedings of the Sixth International Conference on Genetic Algorithms.* Morgan Kaufmann, 1995. pp. 81–87.

[23] J. Lis. Parallel genetic algorithm with dynamic control parameter. In *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*, Piscataway,NY, 1996. IEEE Press. pp. 324–329.

[24] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer - Verlag, New York, 3rd edition, 1996.

[25] Z. Michalewicz and N. Attia. Evolutionary optimization of constrained problems. In A.V. Sebald and L.J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming.* World Scientific, 1994. pp. 98–108.

[26] R. Rechenberg. *Evolutionsstrategie: Optimierung technischer Syseme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, 1973.

[27] J.D. Schaffer and A. Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In *Proceedings of the 2nd International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates, 1987. pp. 36–40.

[28] D Schlierkamp-Voosen and Mühlenbein. Adaption of population sizes by competing subpopulations. In *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*, Piscataway,NY, 1996. IEEE Press. pp. 330–335.

[29] N. Schraudolph and R. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, vol. 9(1):pp. 9–21, 1992.

[30] H-P. Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, volume 26 of *Interdisciplinary systems research.* Birhäuser, Basel, 1977.

[31] H-P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology Series. Wiley, 1995.

[32] C.G. Shaefer. The argot strategy: Adaptive representation genetic optimizer technique. In *Proceedings of the 2nd International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates, 1987. pp. 50–55.

[33] J. Smith and T. Fogarty. Self-adaptation of mutation rates in a steady-state genetic algorithm. In *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation.* IEEE Press, 1996. pp. 318–323.

[34] W.M. Spears. Adapting crossover in evolutionary algorithms. In J.R. McDonnell, R.G. Reynolds, and D.B. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming.* The MIT Press, 1995. pp. 367–384.

[35] M. Srinivas and L.M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24(4):pp. 17–26, 1994.

[36] A. Tuson and P. Ross. Cost based operator rate adaptation: An investigation. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, number 1141 in Lecture Notes in Computer Science, pages 461–469. Springer, Berlin, 1996.

[37] T. White and F. Oppacher. Adaptive crossover using automata. In H.-P. Schwefel Y. Davidor and R. Männer, editors, *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, number 866 in Lecture Notes in Computer Science, pages 229–238. Springer-Verlag, 1994.

[38] J. Xiao, Z. Michalewicz, and L. Zhang. Evolutionary planner/navigator: Operator performance and self-tuning. In *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation.* IEEE Press, May 20–22 1996. pp. 366–371.