Fitness Distributions: Tools for Designing Efficient Evolutionary Computations. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 9. MIT Press, 1999

Christian Igel and Kumar Chellapilla

Fitness distributions are employed as tools for understanding the effects of variation operators in Genetic Programming. Eleven operators are analyzed on four common benchmark problems by estimating generation dependent features of the fitness distributions, e.g. the probability of improvement and the expected average fitness change.

9.1 Introduction

Evolutionary optimization methods can be described as computation procedures that operate on a population of candidate solutions through an iterated process of variation and selection. Different paradigms for evolutionary algorithms (EAs), such as evolutionary programming (EP), evolution strategies (ES), genetic algorithms (GAs), and genetic programming (GP), differ in their representation of solutions in the population, the selection scheme and the representation specific variation operators.

There exists a variety of possible operators that could be used to generate offspring given a representation of the solutions in the population. Operators that provide good rates of finding acceptable solutions through efficient search are desired. From the theoretical point of view, the operators in conjunction with the applied selection scheme should guarantee convergence to the global optimum with probability one at least for finite search spaces [Rudolph, 1997]. The choice of suitable operators has traditionally relied on theoretical models of the dynamics induced by these operators under selection or heuristic information regarding the desired parent-offspring fitness changes [Rechenberg, 1994; Rosca and Ballard, 1994; Utecht and Trint, 1994; Rosca and Ballard, 1995]. In practice, multi-parent multi-offspring self-adaptive EAs on multi-modal objective functions are used to solve problems. For the sake of mathematical tractability and ease of analysis, most theoretical models work with simplified versions of EAs and simple objective functions and the applicability of the following result, in light of the simplifying assumptions used to obtain the result, becomes questionable. Hence, as long as the used EAs cannot be analyzed analytically, heuristic design methodologies have to be applied. However, general heuristic design methodologies such as the principle of strong causality [Rechenberg, 1994] may be difficult to apply to discrete and discontinuous search spaces without an empirical analysis of the interaction of the variation operator under consideration and the problem representation [Rosca and Ballard, 1995; Sendhoff et al., 1997; Igel, 1998]. In GP we are in general concerned with such search spaces and confronted with several variation operators whose effects are far from being understood completely.

In this investigation, we utilize the concept of fitness distribution analysis as a tool for characterizing the behavior of variation operators in GP; a method that may lead to a better understanding of the search process and to simple design heuristics that improve the evolutionary computation.

9.2 Background

Variation operators may be broadly classified into two groups, namely exploratory (or global) search operators and exploitative (or local) search operators. Global optimization requires operators of both types. While local search operators are needed for gradually approaching optima, global search operators are needed for identifying basins of attraction of various optima and also escaping local optima. Exploration and exploitation may be incorporated into an evolutionary scheme in different ways. The same operator may be used to probabilistically act both as a local (exploitative) and global (exploratory) search operator, e.g. Gaussian mutations for real-valued parameter optimization, wherein changes of varying degrees are probabilistically generated even when the variances are fixed. Small changes are generated more often and result in local search whereas large step sizes occur less frequently and constitute a more global search.

Furthermore, the explorative or exploitative nature of an operator may be controlled by parameters like the variances in case of Gaussian mutation. Variation may also be conducted through the use of a set of operators, some of which are tuned to perform local search, while others perform global search. Examples of such a strategy include the use of multiple variation operators for the evolution of the structure and parameters of finite state machines [Chellapilla and Fogel, 1998], neural networks [Yao and Liu, 1996], and parse trees [O'Reilly, 1995; Angeline, 1996; Chellapilla, 1998].

Fitness landscape analysis has been proposed as a tool for analyzing evolutionary computations, see [Macken and Stadler, 1993] for an overview. The fitness landscape is determined by the fitness function and a distance measure on the genotype space, i.e., mathematically it is defined as a map from a (finite) metric space into the real numbers. The results obtained using fitness landscape analysis only hold for the distance measure used. Therefore, the analysis of an evolutionary algorithm based on the fitness landscapes approach requires a metric that is defined with regard to the genetic operators. Examples of such metrics include the Hamming distance for point mutations when using binary representations and the tree edit distance for program spaces [O'Reilly, 1997]. Instead of a standard metric when given a representation, the metric should be defined in terms of the variation operators themselves, with each application of a variation operator generating a unit change in distance. However, the calculation of such a distance measure can be very complex and computationally expensive. Further, if an operator depends on the composition of the current population, as does crossover, the distance measure is different every generation and the distance computations become hard to handle. Kinnear examined the fitness landscapes on which GP operates by comparing the autocorrelation functions of random walks and analyzing adaptive walks with respect to the problem difficulty. The "unusually low" autocorrelations appeared to be "not a particularly good way" to indicate the difficulty of the test problems [Kinnear, Jr., 1994].

The fitness correlation coefficient (FCC) has been proposed as a means for the analysis of variation operators in EAs [Manderick et al., 1991]. The FCC of an operator v is defined as

$$\mathrm{FCC}_v = \frac{\mathrm{cov}(F_{\overline{p}}, F_o)}{\sigma_{F_{\overline{p}}} \sigma_{F_o}} \quad , \tag{9.1}$$

where $\sigma_{F_{\overline{p}}}$ and σ_{F_o} represent the standard deviations of the mean parent fitness and offspring fitness, respectively, and cov represents the covariance function. An absolute FCC value of one indicates a linear dependency between parent and offspring fitness and a value of zero indicates linear independence. An FCC close to zero is regarded as an indicator of a rugged landscape and therefore of weak performance of an EA based on the operator in question. However, instructive counter examples exist that show that the fitness correlation coefficient does not necessarily reflect evolvability [Altenberg, 1995; Fogel and Ghozeil, 1996], one of the main properties determining the quality of evolutionary search. Evolvability means the ability of a parent population to produce variants fitter than any yet existing [Altenberg, 1994].

9.3 Fitness distributions

Fitness distributions (FDs) have been proposed as a means of experimentally estimating the true nature of variation operators on different performance or objective functions. The FD of an operator was described in [Grefenstette, 1995] as the distribution of the offspring fitness given the mean parent fitness. For a variation operator v that produces an offspring from one or more parents the fitness distribution FD_v was defined by the conditional probability

$$\mathrm{FD}_v(F_{\overline{p}}) = P(F_o | F_{\overline{p}}) \quad , \tag{9.2}$$

where the random variables $F_{\overline{p}}$ and F_o denote the mean fitness of the parents used by v to generate the offspring, o, and the resulting offspring fitness, respectively. More generally, the FD_v can depend on all the individual parent fitnesses. Mathematically,

$$\mathrm{FD}_v(F_{\{p\}}) = P(F_o | F_{\{p\}}) \quad , \tag{9.3}$$

where $F_{\{p\}}$ depends on the set of fitness values of all parents that produced the offspring.

The FD_v is generally quite complex and difficult to compute. However, it is usually sufficient to be able to focus on estimating important features of the FD [Grefenstette, 1995; Fogel and Ghozeil, 1996]. In the remainder of this section, we introduce such features of the FD.

Adaptation does not only depend on how often offspring better than their parents are produced, but also on how much better they are [Altenberg, 1994]. Both aspects of the offspring FD must be considered while investigating the properties of variation operators. Therefore, the probability of improvement and expected improvement have been proposed as tools for designing efficient evolutionary computations. Improvement has been defined in different ways. Firstly, improvement can be regarded as the distance covered *on the gradient path* towards the (known) optimum [Rechenberg, 1994]. Secondly, it can be defined as the distance covered to the optimum [Schwefel, 1995]. Thirdly, improvement can simply be defined as the increase in fitness. The first two definitions measure improvement in the search space. An explicit metric on the search space is needed and the optima have to be known. For our purpose, the analysis of GP operators, these definitions (that were designed for real-valued parameter optimization) would not work, so we define improvement in the fitness space. The expected improvement EI_v of an operator is therefore defined as the expectation of the fitness difference between parent(s) and offspring [Fogel and Ghozeil, 1996]. A positive EI indicates that on average the offspring fitness is greater than the parent fitness and a negative EI indicates that the mean offspring fitness is lower than the mean parent fitness. An alternative way to define the expected improvement is to calculate the average change in fitness only when an offspring was fitter than its parent(s), i.e. mathematically the expected improvement is given by $\mathcal{E}(\max\{0, F_o - F_{\{p\}}\})$, where $\mathcal{E}(.)$ denotes the expectation. In this case, the expected improvement is always non-negative [Tuson and Ross, 1998; Fogel and Ghozeil, 1996].

The probability of improvement of a variation operator v, IP_v, is defined as the fraction of the generated offspring that is better than their parents. While an IP value close to one indicates that on average fitter offspring are generated through the use of the operator, an IP value close to zero indicates that on average offspring worse than their parents are generated.

The IP and EI values can be computed analytically for very simple objective functions, e.g. IP and EI for Gaussian mutations using real-valued representation [Rechenberg, 1994] or IP for point mutations using binary representation [Bäck, 1996]. In general, these FD features have to be estimated numerically through Monte-Carlo methods. This has been done for real-valued representation and Gaussian mutations in [Fogel and Ghozeil, 1996].

If the computational effort to produce an offspring is different for the used operators, then the features of the FD should be normalized by the needed effort for comparison.

Unlike with most fitness landscape analysis approaches, the analysis of evolutionary computations using dynamic FD features does not need a specific metric on the search space.

The features of the FD are not static but change with the problem at hand and also during the evolutionary process (e.g. it gets progressively more difficult to create a fitter offspring). In view of this, dynamic (i.e. generation dependent) FD measures are presented that better capture the properties of variation operators during different phases of evolution. In [Nordin and Banzhaf, 1995] the change in fitness (measured in percent) using crossover was calculated in a generation dependent manner for a symbolic regression problem in GP.

Here we estimate seven different features of the FD for investigating the effects of eleven different variation operators for four common test problems in GP. These features are the

Table 9.1
Summary of the investigated features of the fitness distributions, $F_{\{p\}}$ and F_o depend on the fitness of the parent(s) and the corresponding offspring, respectively, $\sigma_{F_{\{p\}}}$ and σ_{F_o} denote the standard deviations, F_b stands for the fitness of the current best individual and $\mathcal{E}(.)$ denotes the expectation

AC	average change	$\mathcal{E}(\lvert F_o - F_{\{p\}}\rvert)$
EI	expected improvement	$\mathcal{E}(F_o - F_{\{p\}})$
IP	improvement probability	$P(F_o > F_{\{p\}})$
WP	worsening probability	$P(F_o < F_{\{p\}})$
SVP	silent variations probability	$P(F_o = F_{\{p\}})$
IP*	probability of being better than the best	$P(F_o > F_b)$
FCC	fitness correlation coefficient	$\dfrac{\mathrm{cov}(F_{\{p\}}, F_o)}{\sigma_{F_{\{p\}}}\sigma_{F_o}}$

expected absolute fitness change (AC), the expected improvement (EI), probability of improvement (IP), the probability of silent variation (SVP), the worsening probability (WP), the fitness correlation coefficient (FCC), and the probability of global improvement (IP*).

• The absolute change in fitness, AC$_v$, of an operator v is defined as the expectation of the absolute change in fitness between the parent and offspring caused by the application of operator v. The AC$_v$ can be used as an indicator of local (small AC values) and global search (large AC values). This feature can be coupled with other information such as the region of application of the variation operator (such as the depth of the node in the parse tree that was used as the crossover point) to determine the sensitivity of different parts of the genotype to the application of the operator.

• In this investigation, the expected improvement, EI$_v$, is defined as the average change in fitness between the parent and offspring caused by the application of operator v.

• The probability of improvement, IP$_v$, of an operator v is defined as the fraction of successful applications of the operator, where an application of the operator v is considered successful if it generates an offspring that is better than its parent.

• The probability of worsening is denoted by WP$_v$ and defined as the fraction of unsuccessful applications of the operator.

• The probability of silent variations, SVP$_v$, measures the fraction of the application of the operator v that produce no change in fitness.

• The global improvement probability IP$_v^*$ is defined as the frequency with which offspring fitter than any existing parent in the population are produced by the operator. This FD feature describes the evolvability.

- The fitness correlation coefficient, FCC_v, see Eq. (9.1).

The mathematical definitions of these FD features are summarized in Table 9.1.

In the current study, the FD features are viewed as dynamic variables that change during evolution. Their definitions are extended to make them functions of the generation. For example, $\mathrm{AC}_v(t)$ is defined as the average absolute change in fitness between the offspring and their corresponding parent(s) at generation t. These generation dependent FD properties for any problem may be conveniently estimated using a set of independent Monte-Carlo trials.

When using multi-parent operators, only the fitness of the *first* parent is used for the calculation of the FD features, i.e., $F_{\{p\}} = F_{p_1}$, see next section. Therefore, since the application of an operator v either increases, decreases, or does not change fitness, we have $\mathrm{IP}_v(t) + \mathrm{WP}_v(t) + \mathrm{SVP}_v(t) = 1$. Using the fitness of the first parent as the reference fitness may not be the best choice for all investigations, e.g. the fitness of the best parent involved may be used instead.

The FD values are amenable to statistical tests. In this investigation, the EI and AC differences were validated using t-tests, the IP, WP, SVP and IP* differences were validated using a χ^2-test, and the FCC differences were tested using a t-test after a z-transformation [Press et al., 1994].

9.4 Evolving computer programs using evolutionary programming

Experiments were conducted by varying the operator(s) used to generate offspring in an evolutionary programming framework. Computer programs were represented as parse trees whose structure and elements were to be evolved. A population of trial computer programs was maintained, variation operators produced changes in these programs, and selection was used to determine which programs were to survive to the next generation and which programs were to be culled from the pool of trials.

9.4.1 Initialization

There are several methods for generating subtrees which can be used to initialize the population. The *full*, *grow* and *ramped half-and-half* methods of subtree generation were introduced in [Koza, 1992] and are based on tree depth. The ramped half-and-half method is the most commonly used method of generating random parse trees because of its relatively higher probability of generating subtrees of varying depth and size [Koza, 1992; Banzhaf et al., 1998]. However, these methods do not produce a uniform sampling of the search space [Iba, 1996]. The subtree generation method used here was based on the length of the subtree, rather than the depth. First the length of the program was randomly selected to be between 3 and $L_{\max}$. A random program was generated with approximately that length to initialize the parent. Suppose a tree of length 24 was to be generated. In the beginning, as the tree was constructed, nodes and leaves were selected with a 50% probability. No discrimination was made between which nodes and leaves were to be added to the tree. As the length of the tree came close to 24, the selection of functions and terminals was restricted to only those that would make the length of the program exactly 24 (cf. [Chellapilla, 1998]). In this study, each of the simulations were carried out with a set of 500 parents, as e.g. in [Koza, 1992]. Each initial tree was randomly generated using the above subtree generation method with a length between 3 and 50.

9.4.2 Offspring generation through variation

A set of eleven variation operators $V_{\text{all}} = \{$*OneNode*, *AllNodes*, *Swap*, *Grow*, *Trunc*, *OneC*, *AllC*, *Macro*, *CrossU*, *CrossSHC*, *CrossWHC*$\}$ was employed to generate offspring from the parents. From the various GP variation operators introduced so far, we chose operators based on [Angeline, 1997a; Chellapilla, 1998] for our investigation.

In the EP framework presented here, each parent generated a single offspring through an application of one randomly selected operator with uniform probability from V_{all}. The operators are described as operating on a single parent, say p_1, to generate a single offspring. The two-parent variation operators operated on the parent p_1, and a mating parent, say p_2, that was either randomly selected from the population or randomly generated. If the offspring genotype did not differ from the parent p_1, this offspring was discarded and a new offspring was generated using p_1.

- *OneNode* randomly selected a node in the program and replaced it with another node of the same arity. *AllNodes* selected each and every node in the program and replaced it with a random node of the same arity.

- *Swap* selected a node that took more than one argument, randomly selected two of its arguments and swapped them.

- *Grow* selected a random leaf in the program and replaced it with a newly generated subtree.

- *Trunc* randomly selected a function node in the program and replaced it with a terminal, thus effectively clipping the tree at that node.

- The *OneC* operator was the same as Gaussian mutation [Chellapilla, 1998] and was applied only to those terminal nodes in the tree that were numeric constants. It perturbed a randomly selected numeric constant by a Gaussian random number with zero mean and a standard deviation of 0.1.

- The *AllC* operator perturbed every numeric constant with independent and identically distributed Gaussian random numbers with mean zero and standard deviation of 0.1.

• The *Macro* operator [Chellapilla, 1998] applied a sequence of simple mutation operators to generate an offspring from a parent using the following steps:

1. A Poisson random number N, with mean λ, was generated.

2. N random variation operators were uniformly selected with replacement from the set of variation operators, $V_{Macro} = \{OneNode, AllNodes, Swap, Grow, Trunc, OneC, AllC\}$. If there were no numeric constants in the terminal set for the problem, then *OneC* and *AllC* operators were excluded from the set.

3. These N mutation operators were applied in sequence one after the other to the parent to generate the offspring. As an example, if the value N were 2 and the operators selected were *OneNode* and *Grow*, then the offspring would be given by

$$\text{Offspring} = \textit{Grow}(\textit{OneNode}(\text{Parent}))$$

For the experiments in this study, the mean value λ was selected to be 4.

• The *CrossU* operator was a modified version of the subtree crossover operator defined in [Koza, 1992]. When a parent, p_1, had to be varied using *CrossU*, a mate, p_2, was selected uniformly at random from the existing population at that generation. Randomly selected subtrees were swapped between p_1 and p_2 to generate two intermediate offspring, say o_1 and o_2. Both crossover points are selected in an unbiased manner, i.e., there was no bias towards selecting function nodes more often than terminal nodes as in [Koza, 1992]. If either of the two intermediate offspring violated the size constraint $L_{\max} = 50$ it was considered infeasible. If only one intermediate offspring was feasible, it became p_1's offspring. If both the intermediate offspring were feasible, then one was selected at random to become the offspring. An equal probability of selecting functions and terminal nodes for crossover coupled with an upper limit of $L_{\max}$ nodes provided a parsimony bias to the crossover operator.

• The strong and weak versions of the random mate crossover, *CrossSHC* and *CrossWHC*, were inspired by the headless-chicken crossover operators in [Angeline, 1997a; Angeline, 1997b]. These operators exchanged randomly selected subtrees between the parent, p_1, and a randomly generated mate, p_2, to obtain two intermediate offspring, say o_1 and o_2. The randomly generated mate, p_2, comprised a randomly selected parent, say p', from the population that was subsequently modified through the application of *AllNodes*. Thus, p_2, had the structure of the parent p' but any "content information" was completely randomized. *CrossSHC* returned the intermediate offspring that had the same root node as p_1 whereas *CrossWHC* returned one of the two intermediate offspring at random. If the intermediate offspring to be returned by *CrossWHC* violated the size constraint, then the other offspring which would definitely be feasible was returned. On the other hand, if the offspring to be returned by *CrossSHC* violated the size constraint, the process was repeated by regenerating a new set of intermediate offspring through the selection of different crossover points.

9.4.3 Parent selection

EP-style tournament selection [Fogel, 1995] with ten opponents was applied to select the parents for the next generation: Every program in the population was compared with ten randomly selected opponents out of the population. For each comparison in which the program's fitness was better or equal, it received a win. The better half of the population with the largest number of wins became the parents for the next generation. This process of variation and selection was repeated every generation for a predefined number of generations, $k_{\max}$.

9.4.4 Test problems

Our experiments were conducted on a suite of four problems: the 6-bit multiplexer problem, the artificial ant problem (Santa Fe trail), the cart centering problem, and the sunspot modeling problem. Brief descriptions of these problems follow. For a more detailed description the reader is referred to [Koza, 1992] and [Angeline, 1996].

The goal of the 6-multiplexer [Koza, 1992] problem is to find a computer program consisting of primitive Boolean functions, namely not, and, or, and if, that computes the output of a four-input two-select multiplexer. The fitness of an individual is the number of correct outputs that the program generates when tested on all the 64 possible inputs. A successful program correctly maps all 64 inputs to their corresponding outputs.

In the artificial ant problem, the goal is to evolve a computer program that would act as a move generating rule for guiding an ant to find all food packets lying on an irregular trail. The "Santa Fe trail" [Koza, 1992] containing 89 food items was used. The function and terminal sets were $\{\mathrm{left}, \mathrm{right}, \mathrm{move}\}$ and $\{\text{If-food-ahead}, \mathrm{Prog2}, \mathrm{Prog3}\}$, respectively. A move rule was considered to be successful if it could guide the ant to collect all 89 food packets on the trail.

The cart centering problem requires the discovery of a control law that centers a cart that is free to move to the left or right on a frictionless surface. The terminal set consisted of the two state variables of the system (the position x and the velocity $\dot{x}$) whereas the function set was $\{+, -, *, \%, \mathrm{abs}, \mathrm{gt}\}$. At any given time, the control law determined the direction of a force (of 1.0 N) to be applied. The total time needed to center the cart from a set of twenty initial states $(x, \dot{x})$ selected uniformly at random from $[-0.75, 0.75]^2$ were used to compute the quality of the control law in centering the cart. The set of initial states was kept fixed during evolution. A trial was considered to be successful if a control law was found that could center the cart with a total time that was within 1% of the total time taken by the mathematically solution that is optimal over $[-\infty, \infty]^2$. The cart was considered to be centered when $\sqrt{x^2 + \dot{x}^2} < 0.1$. Each controller was given a maximum of 10 seconds (500 steps of 0.02 seconds each) to center the cart. Any control law that failed to center the cart within this time was given a time score of 10 seconds.

Table 9.2
Computational effort results for the artificial ant, 6-bit multiplexer, cart centering, and sunspot modeling problems using all variations operators, V_{all}. Computation effort analysis indicates that $R(z)$ number of independent trials, each lasting N generations, need to be conducted with a population size of $M(= 500)$, to achieve a success probability of $z = 0.99$, resulting in a total of $I(M, N, z)$ number of individuals being processed (see [Koza, 1992] for further details). $(*)$ indicates that all trials were successful and the results correspond to $z = 1.0$. The first two columns give the mean fitness and the corresponding standard deviation of the fitness at generation 200.

	Mean	sd	N	$R(z)$	$I(M, N, z)$
Artificial Ant	89	0	143	1	72,000*
6-bit Multiplexer	64	0	104	1	52,500*
Cart Centering	38.990	5.012	155	1	78,000*
Sunspot Modeling	2991.03	384.82		——	

The goal of the sunspot series modeling problem is to compute the average number of sunspots observed in year y using the average number of sunspots observed in the years $y-1$, $y-2$, $y-4$, and $y-8$, denoted by S_{y-1}, S_{y-2}, S_{y-4}, S_{y-8}, respectively. Sunspot data from the years 1700 to 1989 containing 290 samples were used as the training set. The terminal set was $\{S_{y-1},\ S_{y-2},\ S_{y-4},\ S_{y-8},$ numeric constants$\}$. The functions set was $\{+, -, *, \%, \sin, \cos\}$. The mean square error over the 290 samples in the training set was taken to be the error score of the individual. The ability of the solutions to generalize was not considered: We only judged how well the solutions fitted the data set and not, as should be done in real world application, how well the solution really modeled the time series.

9.4.5 Experiments

Parent and offspring fitness data were collected in every generation, for each application of every operator, on each of the four problems, over 50 independent trials of the above described algorithm. Each trial used a population size of 500 and evolution lasted for $k_{\max} = 200$ generations. The collected data were used to estimate the FD features in Table 9.1 of the eleven different operators.

9.5 Results

All 50 artificial ant and 6-bit multiplexer trials were successful by generations 143 and 104, respectively. The sum squared error on the sunspot modeling problem quickly decreases in the first 75 generations. Even at generation 200, when the trials were terminated, the sum squared error continues to decrease as models and parameters that better fit the training data are found. This does not necessarily imply that the found solutions are better models of the time series, because they may overfit the training data (the same holds for the cart centering results). At the end of 200 generations, the mean best sum squared error (over all 50 trials) was 2991.03 with a standard deviation of 384.82. It appears that models with higher fitness

Table 9.3
Mean cumulative FD features (averaged over all generations and 50 independent trials) on the 6-bit multiplexer problem for the different variation operators used to generate offspring in an evolutionary programming procedure for evolving computer programs. These seven fitness distribution features are absolute change in fitness (AC), expected improvement (EI), improvement probability (IP), silent variation probability (SVP), worsening probability (WP), fitness correlation coefficient (FCC), and global improvement probability (IP*). The values in parentheses indicate the rank of the operator in terms of the FD feature. Smaller ranks imply a larger FD feature value. All differences in feature values were statistically significant ($p < 0.05$) except for those marked by daggers: (†) indicates that the difference between the table entry and the next higher ranking entry in the same column were not statistically significant.

	AC	EI	IP	SVP	WP	FCC	IP*/10^{-3}
OneNode	$4.32^{(7)}$	$-4.25^{(3)}$	$0.013^{(3)}$	$0.371^{(2)}$	$0.616^{(8)}$	$0.759^{(2)}$	$0.126^{(3)\dagger}$
AllNodes	$24.53^{(1)}$	$-24.48^{(9)}$	$0.005^{(8)}$	$0.007^{(9)}$	$0.988^{(1)}$	$0.045^{(9)}$	$0.022^{(9)}$
Swap	$3.75^{(9)}$	$-3.73^{(1)}$	$0.004^{(9)}$	$0.678^{(1)}$	$0.319^{(9)}$	$0.596^{(3)}$	$0.037^{(8)\dagger}$
Grow	$4.28^{(8)}$	$-4.16^{(2)}$	$0.023^{(2)}$	$0.341^{(3)}$	$0.636^{(7)}$	$0.855^{(1)}$	$0.168^{(2)\dagger}$
Trunc	$7.55^{(4)}$	$-7.48^{(6)}$	$0.012^{(5)}$	$0.287^{(6)}$	$0.701^{(4)}$	$0.495^{(6)}$	$0.098^{(5)\dagger}$
Macro	$18.74^{(2)}$	$-18.68^{(8)}$	$0.008^{(7)}$	$0.098^{(8)}$	$0.894^{(2)}$	$0.174^{(8)}$	$0.038^{(7)\dagger}$
CrossU	$6.77^{(5)}$	$-6.62^{(5)}$	$0.027^{(1)}$	$0.328^{(5)}$	$0.645^{(6)}$	$0.513^{(5)}$	$0.176^{(1)\dagger}$
CrossWHC	$15.98^{(3)}$	$-15.92^{(7)}$	$0.009^{(6)}$	$0.154^{(7)}$	$0.837^{(3)}$	$0.216^{(7)}$	$0.078^{(6)}$
CrossSHC	$6.06^{(6)}$	$-5.99^{(4)}$	$0.013^{(3)}$	$0.333^{(4)}$	$0.654^{(5)}$	$0.583^{(4)}$	$0.110^{(4)\dagger}$

Table 9.4
Mean cumulative fitness distribution features on the artificial ant problem

	AC	EI	IP	SVP	WP	FCC	IP*/10^{-3}
OneNode	$37.09^{(6)}$	$-36.99^{(4)}$	$0.009^{(3)}$	$0.423^{(2)}$	$0.569^{(7)}$	$0.325^{(4)}$	$2.154^{(2)\dagger}$
AllNodes	$74.04^{(1)}$	$-73.99^{(9)}$	$0.006^{(9)}$	$0.003^{(9)}$	$0.991^{(1)}$	$-0.012^{(9)}$	$2.005^{(6)\dagger}$
Swap	$39.74^{(4)}$	$-39.66^{(6)}$	$0.008^{(5)}$	$0.405^{(4)}$	$0.587^{(5)}$	$0.295^{(6)}$	$1.898^{(8)}$
Grow	$35.95^{(8)\dagger}$	$-35.80^{(2)}$	$0.016^{(2)}$	$0.405^{(4)}$	$0.579^{(6)}$	$0.363^{(1)}$	$4.668^{(1)}$
Trunc	$39.25^{(5)}$	$-39.16^{(5)}$	$0.008^{(5)}$	$0.392^{(6)}$	$0.600^{(4)}$	$0.308^{(5)}$	$1.655^{(9)}$
Macro	$66.07^{(2)}$	$-66.01^{(8)}$	$0.007^{(7)}$	$0.087^{(8)}$	$0.906^{(2)}$	$0.115^{(8)}$	$2.048^{(3)\dagger}$
CrossU	$35.81^{(9)}$	$-35.35^{(1)}$	$0.028^{(1)}$	$0.412^{(3)}$	$0.560^{(9)}$	$0.329^{(2)\dagger}$	$2.045^{(4)\dagger}$
CrossWHC	$57.42^{(3)}$	$-57.35^{(7)}$	$0.007^{(7)}$	$0.193^{(7)}$	$0.801^{(3)}$	$0.178^{(7)}$	$1.909^{(7)\dagger}$
CrossSHC	$36.75^{(7)}$	$-36.65^{(3)}$	$0.009^{(3)}$	$0.429^{(1)}$	$0.562^{(8)}$	$0.327^{(3)\dagger}$	$2.032^{(5)\dagger}$

Table 9.5
Mean cumulative FD features on the cart centering problem

	AC	EI	IP	SVP	WP	FCC	IP*/10^{-3}
OneNode	$39.46^{(7)}$	$-39.19^{(3)}$	$0.012^{(3)}$	$0.274^{(3)}$	$0.714^{(7)}$	$0.309^{(3)}$	$0.185^{(4)}$
AllNodes	$137.10^{(1)}$	$-136.90^{(9)}$	$0.004^{(8)}$	$0.007^{(9)}$	$0.988^{(1)}$	$0.063^{(9)}$	$0.027^{(9)}$
Swap	$26.25^{(9)}$	$-26.14^{(1)}$	$0.004^{(8)}$	$0.712^{(1)}$	$0.285^{(9)}$	$0.354^{(2)}$	$0.031^{(8)\dagger}$
Grow	$37.12^{(8)}$	$-36.38^{(2)}$	$0.027^{(2)}$	$0.277^{(2)}$	$0.696^{(8)}$	$0.484^{(1)}$	$0.523^{(1)}$
Trunc	$49.85^{(4)}$	$-49.61^{(6)}$	$0.010^{(5)}$	$0.193^{(6)}$	$0.797^{(4)}$	$0.270^{(6)}$	$0.128^{(5)\dagger}$
Macro	$93.69^{(3)}$	$-93.48^{(7)}$	$0.006^{(7)}$	$0.167^{(7)}$	$0.826^{(3)}$	$0.145^{(7)}$	$0.087^{(7)}$
CrossU	$45.46^{(5)}$	$-44.77^{(5)}$	$0.034^{(1)}$	$0.200^{(5)}$	$0.766^{(5)}$	$0.280^{(5)}$	$0.292^{(2)\dagger}$
CrossWHC	$94.43^{(2)}$	$-94.19^{(8)}$	$0.008^{(6)}$	$0.103^{(8)}$	$0.889^{(2)}$	$0.142^{(8)}$	$0.099^{(6)\dagger}$
CrossSHC	$43.40^{(6)}$	$-43.13^{(4)}$	$0.012^{(3)}$	$0.228^{(4)}$	$0.760^{(6)}$	$0.301^{(4)}$	$0.234^{(3)\dagger}$

Table 9.6
Mean cumulative FD features on the sunspot modeling problem

	$\mathrm{AC}/10^4$	EI	IP	SVP	WP	FCC	$\mathrm{IP}^*/10^{-3}$
OneNode	$10.90^{(5)}$	$-109000.0^{(7)}$	$0.137^{(5)}$	$0.029^{(4)}$	$0.834^{(7)}$	$0.066^{(6)}$†	$0.990^{(4)}$†
OneC	$0.00432^{(11)}$	$-30.13^{(1)}$	$0.340^{(1)}$	$0.149^{(2)}$	$0.511^{(10)}$	$0.998^{(1)}$	$1.676^{(2)}$
AllC	$0.01091^{(10)}$	$-70.85^{(2)}$	$0.300^{(2)}$	$0.032^{(3)}$	$0.667^{(9)}$	$0.984^{(2)}$	$2.126^{(1)}$
AllNodes	$52.25^{(1)}$	$-521400.0^{(11)}$	$0.009^{(11)}$	$0.000^{(11)}$	$0.991^{(1)}$	$-0.066^{(11)}$	$0.006^{(11)}$
Swap	$3.42^{(9)}$	$-33970.0^{(3)}$	$0.033^{(10)}$	$0.554^{(1)}$	$0.412^{(11)}$	$0.175^{(3)}$	$0.144^{(10)}$
Grow	$11.96^{(4)}$	$-119000.0^{(8)}$	$0.149^{(3)}$	$0.023^{(8)}$	$0.828^{(8)}$	$0.086^{(5)}$	$0.859^{(6)}$†
Trunc	$6.55^{(8)}$	$-64810.0^{(4)}$	$0.110^{(7)}$	$0.026^{(5)}$	$0.864^{(4)}$	$0.093^{(4)}$†	$0.683^{(7)}$
Macro	$28.62^{(3)}$	$-285400.0^{(9)}$	$0.062^{(9)}$	$0.026^{(5)}$	$0.912^{(3)}$	$-0.009^{(9)}$†	$0.431^{(9)}$
CrossU	$8.01^{(7)}$	$-78760.0^{(5)}$	$0.141^{(4)}$	$0.013^{(9)}$	$0.846^{(6)}$	$0.041^{(8)}$	$1.062^{(3)}$†
CrossWHC	$29.53^{(2)}$	$-294300.0^{(10)}$	$0.063^{(8)}$†	$0.012^{(10)}$	$0.925^{(2)}$	$-0.012^{(10)}$	$0.490^{(8)}$†
CrossSHC	$10.09^{(6)}$	$-100500.0^{(6)}$	$0.125^{(6)}$	$0.025^{(7)}$†	$0.850^{(5)}$	$0.061^{(7)}$	$0.960^{(5)}$†

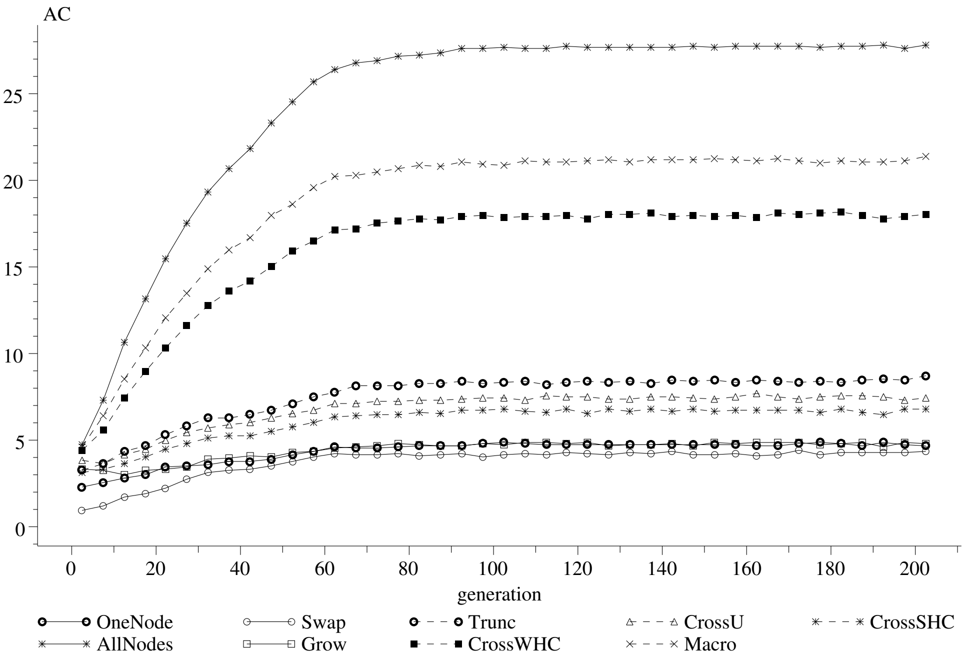

Figure 9.1
Expected absolute change, $\mathrm{AC}_v(t)$, on the 6-bit multiplexer problem. As evolution progresses, the amount of change generated by an operator increases. In view of the fact that the maximum fitness is 64 and that randomly selected individuals in the search space have a mean fitness around 32, the *AllNodes* operator almost completely degenerates the solutions to random samples from the search space. The AC values for *AllNodes* are followed by those of *Macro*, *CrossWHC*, *Trunc*, *CrossU*, *CrossSHC*, *CrossSHC*, *Grow*, *OneNode*, and *Swap*. The lowest AC values for Swap were generated due to a large percentage of silent variations (see Figure 9.6).

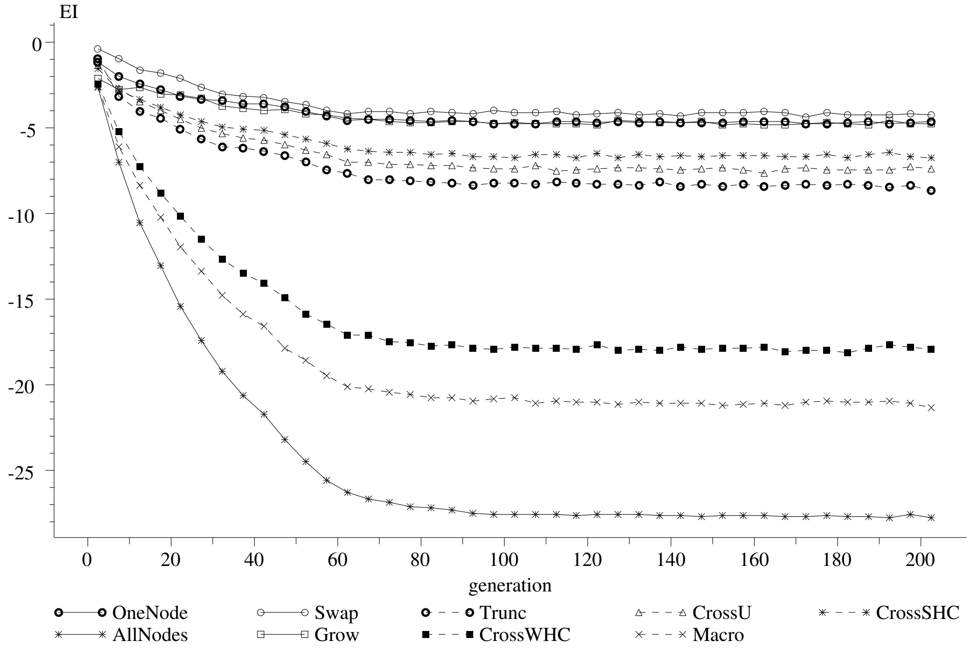

Figure 9.2
Expected improvement, $\mathrm{EI}_v(t)$, on the 6-bit multiplexer problem. The EI ranking of the operators are exactly opposite to the AC ranking indicating that on the multiplexer problem small changes in fitness were likely to lead to greater improvement, i.e. in this case lesser degradation.

are likely to be found if evolution were continued beyond 200 generations. In the cart centering problem, the mathematically optimal rule centered the cart in roughly 2.0088 seconds [Chellapilla, 1997]. We observe that our algorithm consistently found solutions that were better than the optimal solution. The reason for this was that the mathematically optimal solution is optimal over initial states $(x, \dot{x})$ in $[-\infty, \infty]^2$ and not over the subset $[0.75, 0.75]^2$. Detailed analysis of the cart centering problem may be found in [Chellapilla, 1997]. By generation 155, all 50 cart centering trials were successful. The computational effort results of the four test problems are presented in Table 9.2.

Figures 9.1–9.7 graph the binned, generation dependent values of the seven features of the FD for the 6-bit multiplexer problem. The corresponding graphs for the artificial ant problem were very similar. The curves for the cart centering and sunspot modeling problems were similar to each other but differed from those for discrete problems. Where necessary, some figures for the sunspot modeling problem have been included.

In order to obtain smoother graphs that were easier to interpret, these feature values were distributed into consecutive bins five generations wide and the mean value in each bin was

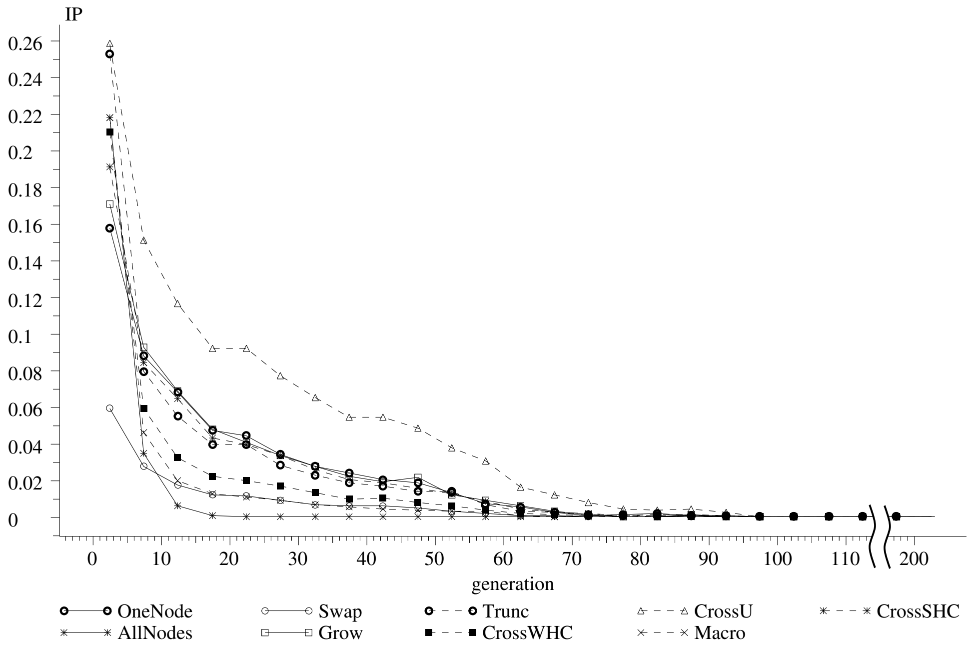

Figure 9.3
Probability of improvement, $\mathrm{IP}_v(t)$, on the 6-bit multiplexer problem. Among the operators investigated, the *CrossU* operator produces significantly higher IP values and is followed by the *Grow* operator.

plotted as a function of the middle generation of the bin. It is interesting to note that all the smoothened FD curves, with the exception of the IP* and some FCC curves, for the various operators maintain a relatively consistent ranking throughout evolution indicating that the FD features were identifying properties intrinsic to these variation operators. Operators that generated relatively large values of these features during initial phases of evolution continued to do so till the end of the evolutionary process. The relative ranking of these operators on different problems also appears to be similar.

The AC curves (Figure 9.1) start out small during the first few generations and gradually increase and attain their highest values towards the end of the trial. Similarly, the EI curves start out high and progressively drop down to low values (see Figure 9.2). Further, the expected improvement was always negative for all four problems. In comparison with the EI results on optimization problems in the continuous domain [Fogel and Ghozeil, 1996] where positive EI values are common, program evolution problems appear to be much more difficult.

Figure 9.8 shows the IP graph for the sunspot modeling problem. Similar to the EI values in the first few generations the IP values (Figures 9.3 and 9.8) start out high and quickly

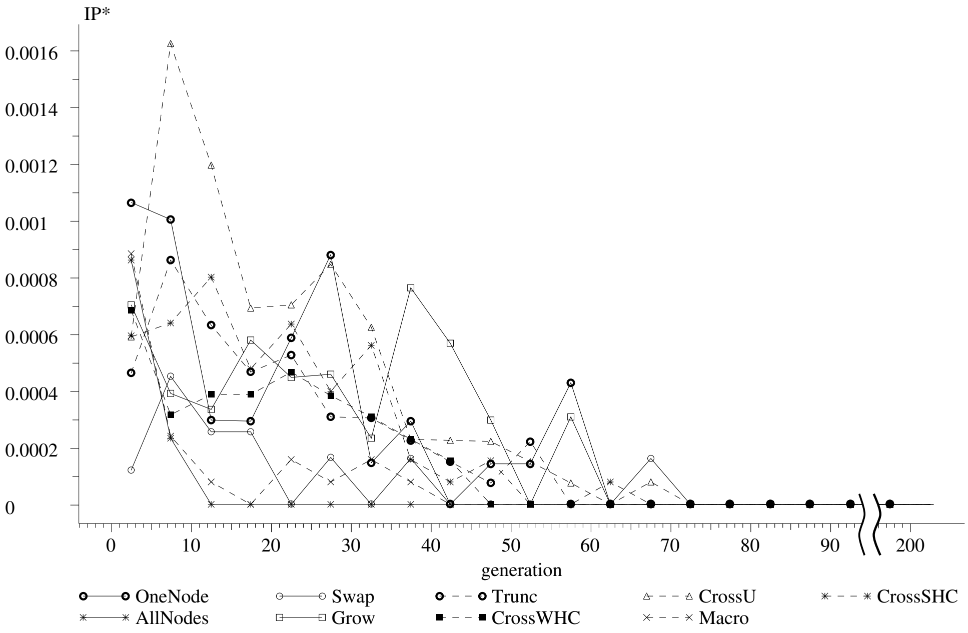

Figure 9.4
Global improvement probability, $\mathrm{IP}^*_v(t)$, on the 6-bit multiplexer problem. *CrossU*, *OneNode*, and *Grow* appear to produce the most global improvements, while *AllNodes* and *Macro* generate the lowest number of global improvements.

decrease. The initial IP values were relatively lower on the discrete problems (ant, mux) than those for continuous the problems (cart, sunspot). The rate of decrease was also much higher on discrete problems than on the continuous problems. On continuous optimization problems gradual changes in fitness values are possible. This gradual change in fitness and error scores allows evolutionary search to generate frequent enhancements resulting in higher IP values.

The WP curves were completely determined by the IP and SVP curves. When the IP values were low and the SVP values were high (e.g. on the ant, mux, and cart problems) the WP curves were inverted versions of the SVP curves, whereas when the IP values were high and the SVP values were low (e.g. on the sunspot problem) the WP curves were inverted versions of the IP curves.

On all four problems, every operator generated global improvements in at least one generation. Of all the features, the IP^* was directly related to the rate of finding good solutions of the algorithm especially when the range of the objective function is discrete and bounded. For example, on the 6-bit multiplexer problem, there are just 64 possible test

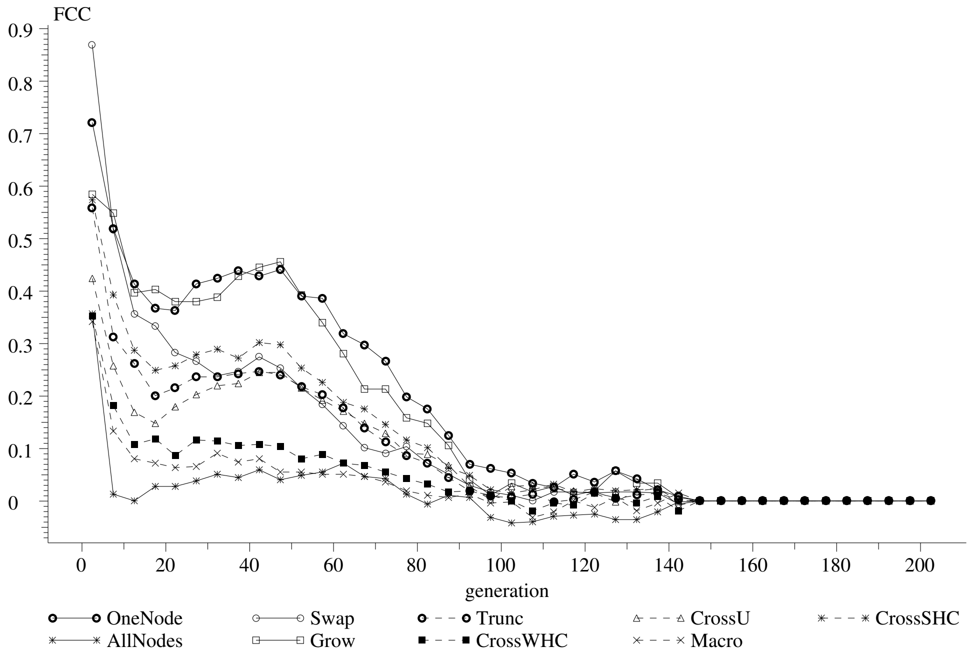

Figure 9.5
Fitness correlation coefficient, $\mathrm{FCC}_v(t)$, on the 6-bit multiplexer problem. The FCC values decrease because it becomes progressively difficult to generate offspring that have a fitness close to the parents fitness as the parent fitness increases.

cases that determine the fitness and the minimum improvement in fitness is one point. This implies that at most 64 global improvements per individual can be generated before finding an optimal solution.

The IP* graphs for the sunspot problem are shown in Figure 9.9. On continuous problems high IP* values should also occur with high EI values. Unfortunately, global improvements occur very rarely, the IP* probabilities are orders of magnitude lower than corresponding IP values. Hence, it is difficult to obtain significant results and therefore it is problematic to rely on IP* as a measure for estimating operator usefulness.

On finding the global best solution all IP* values fall to zero. As a result, the IP* curves for the mux and ant problems reached zero at generations 104 and 143, respectively. The IP* curves for the cart centering and sunspot modeling problems show persistent changes even at the end of the trials, indicating that if evolution is continued further solutions with higher fitness will be obtained.

On the discrete problems (ant and mux) the FCC curves (Figure 9.5) started out high, rapidly fell in the first ten generations, then increased, peaked, and finally gradually de-

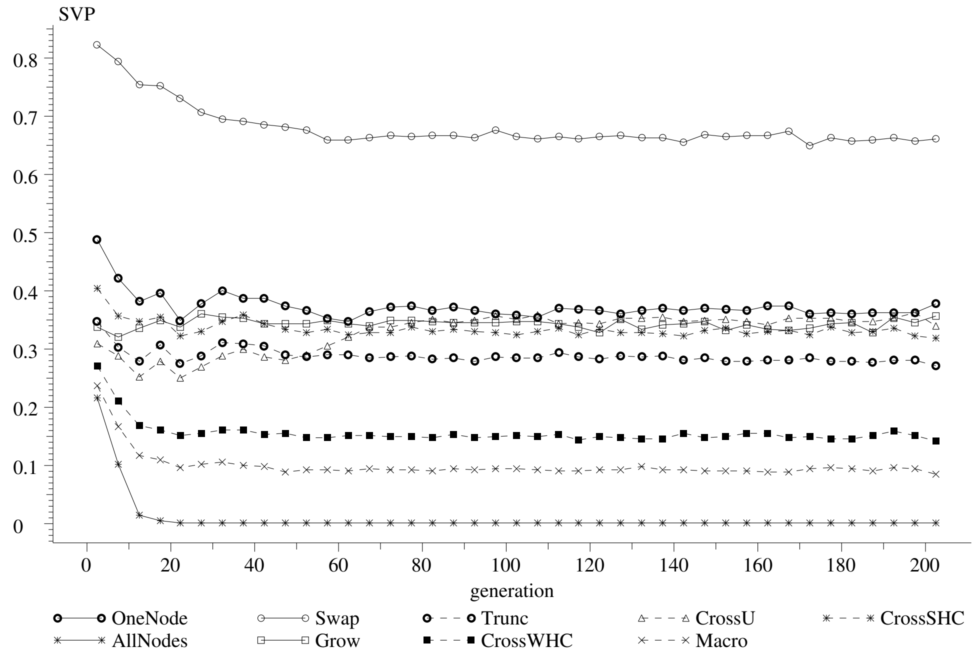

Figure 9.6
Probability of silent variations, $\mathrm{SVP}_v(t)$, on the 6-bit multiplexer problem. The high SVP values for the *Swap* operator were caused by the commutative boolean functions $(\mathrm{and}, \mathrm{or})$. *AllNodes* had the lowest SVP values followed by *Macro* (which probabilistically uses *AllNodes*) and *CrossWHC*. After the first 15 generations the SVP values undergo little change.

creased resulting in a hump during generations 10–70. Towards the end of the runs the FCC values became very small and reached zero. On the continuous problems (cart and sunspot) the FCC curves for all operators, with the exception of those for the *OneC* and *AllC*, rapidly fell down to very low (< 0.15) values indicating that there was no correlation between the parent and offspring fitnesses and the relative ranking of the various operators juggled rapidly. This lack of correlation appears to be caused by large changes in fitness and error scores between the parent and offspring. On the contrary, the *OneC* and *AllC* operators used in the sunspot modeling problem, produced small changes in the sum squared error and consequently exhibited large FCC values close to one. Overall, we did not find that the FCC predicted interesting characteristics of the operators that were not already captured by the other FD features.

For all operators with the exception of *CrossU*, the SVP values showed a slight decrease in the initial stages of evolution and remained nearly constant in subsequent generations, see Figure 9.6. For *CrossU* we measured a slight increase in later generations. Not surpris-

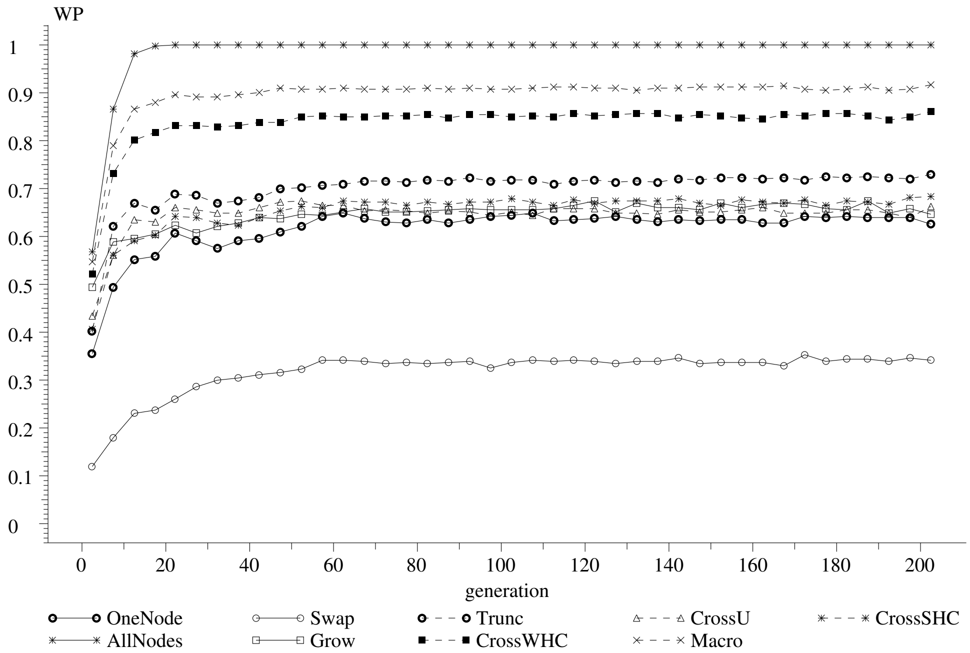

Figure 9.7
Probability of worsening, $\mathrm{WP}_v(t)$, on the 6-bit multiplexer problem. The WP was lowest for the *Swap* operator to due the large number of silent variations it generates (see Figure 9.6). On the other hand, the WP for *AllNodes* reaches 1.0 after generation 25 and stays there throughout evolution, depicting its extremely destructive nature. The high WP values for *AllNodes* are followed by those for *Macro* (which probabilistically uses *AllNodes*) and *CrossWHC*. For most of the operators, due to the really low IP values (see Figure 9.3), the WP curves are simply the inverted version of the $\mathrm{SVP}_v(t)$ curves in Figure 9.6.

ingly, the *Swap* operator generated really high (SVP > 0.65) rates of silent variations when commutative functions were present in the function set. *AllNodes* did not generate any silent variations after generation 20 in any of the four problems studied here. *Macro* and *CrossSHC* generated very few silent variations. All remaining operators generated silent variations in one out of every three times they were applied when numeric constants were not included in the terminal set. However, when numeric constants were included in the terminal set, the rate of silent variations for these operators were considerably lower (< 0.1). In symbolic regression problems, when numeric constants are also evolved, the range space typically spans the whole real line or a dense subset of the real line. The likelihood of these remaining operators generating changes to the symbolic expression that produce no change in the fitness falls and silent variations become less probable, especially in the absence of conditionals such as if.

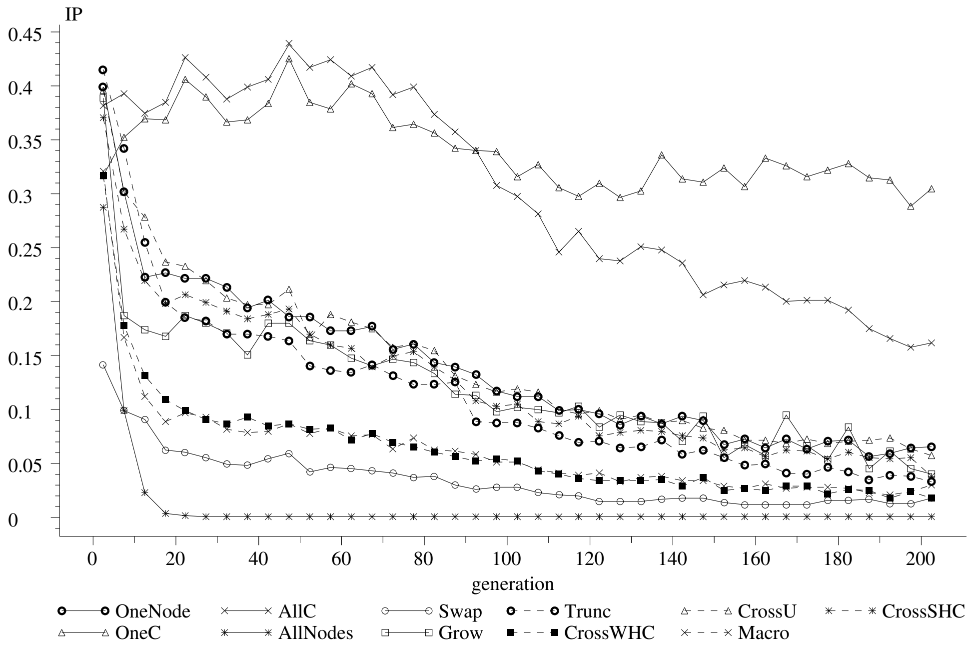

Figure 9.8
Probability of improvement, $\mathrm{IP}_v(t)$, on the sunspot problem. In the first 100 generations *AllC* generates the highest IP values and is closely followed by *OneC*. However, during the later stages of evolution, as would be expected *OneC* generates larger IP, values indicating that smaller changes are preferable in the later stages of evolution. *OneNode* and *CrossU* also produce relatively high $\mathrm{IP}_v(t)$ values and are closely followed by the *Grow* operator.

As expected, all the FD features indicate that as evolution progresses it becomes increasingly difficult to find better solutions.

As the relative ranking of the operators varied only very slightly on all four problems during evolution, the cumulative FD features, averaged over all generations and all trials, can be used as a concise and useful measure for analyzing the properties of the examined variation operators. These cumulative FD features for the eleven variation operators on the four test problems are presented in Tables 9.3, 9.4, 9.5, and 9.6, respectively. The relative ranking of each operator on each cumulative feature is also presented in these tables. For all FD features, lower ranks imply a larger feature value.

Most differences in AC, EI, IP, SVP, WP, and FCC were statistically significant. The number of global improvements were very few in comparison with the number of samples collected over all 50 trials and as a result the IP* differences were mostly statistically not significant.

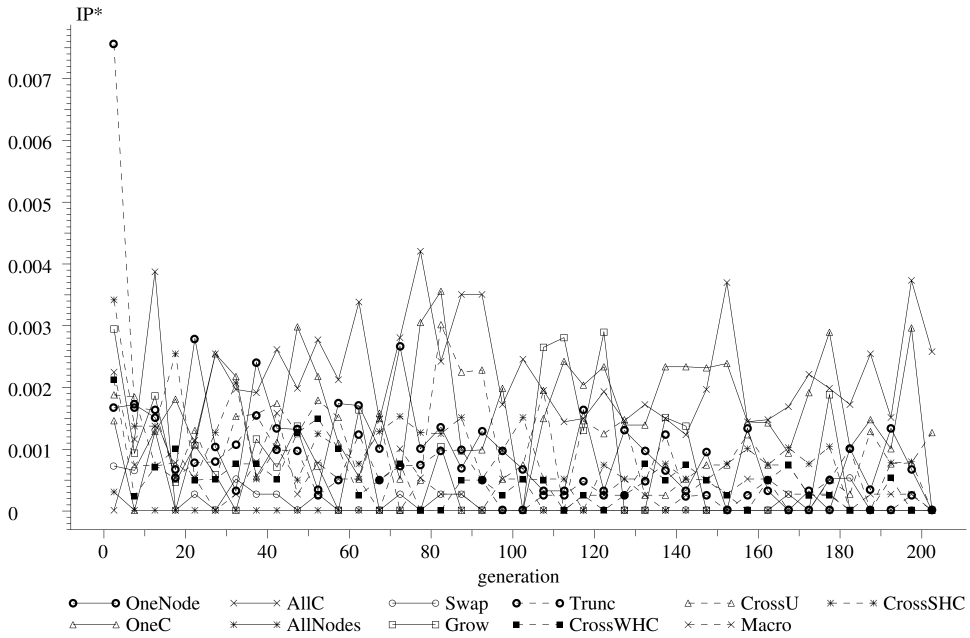

Figure 9.9
Global improvement probability, $\text{IP}_v^*(t)$, on the sunspot modeling problem. *OneC* and *AllC* generated far higher global improvements indicating importance of evolving appropriate constants during evolution. *OneNode*, *Grow*, and *CrossU* also appear to generate relatively large IP* values. Because global improvements occur rarely, the generation dependent IP* plots are very noisy and not very informative.

In the experiments without numeric constants, *OneNode* performs the smallest structural change in the genotype with respect to the tree edit distance [O'Reilly, 1997]. In view of this, it was considered to be a rather local search operator with most applications of the operator resulting in small changes in fitness and error scores, i.e. small AC values. However, the results show that it ranked higher (with larger AC values) than *Swap* and *Grow* operators on three of the four problems, and occasionally ranked higher than *CrossU*, *CrossSHC*, and *Trunc*. Thus, contrary to what was initially expected, the changes in the fitness space produced by *OneNode* were not small. Furthermore, it consistently ranked third in terms of IP and between second and fourth in terms of IP*.

The *AllNodes* operator had the lowest rate of silent mutations and the highest probability of worsening in all test problems. It generated a less fit offspring nearly every time it was applied with cumulative WP values of 0.998, 0.991, 0.988 and 0.991 on the four problems. It almost never generated silent variations with cumulative SVP values of 0.007, 0.003, 0.007 and zero on the four problems. Additionally, *AllNodes* had the lowest EI values on all four problems. The FCC scores close to zero indicate that there was no linear correlation between parental and offspring fitness when *AllNodes* was used to generate offspring. The only similarity between the offspring generated using *AllNodes* and the corresponding parent is the shape of the parse tree. The *AllNodes* results indicate that the spatial structure, i.e. the shape of the parse tree without its labels, carries negligible information about the quality of the computer program. *AllNodes* rarely generated any offspring that were better than the best parent in that generation and as a result ranked last (mux, cart, sunspot) or third last (artificial ant) in terms of IP*. On all four problems after the first 30 generations the *AllNodes* operator never generated any global enhancements. Similarly, after 30 generations, on the mux, ant, and cart centering problems, the *AllNodes* operator never generated any improvements or silent variations, i.e., the IP, SVP, and WP values were zero, zero, and one.

The *Swap* operator had the highest number of silent variations on the mux, cart centering, and sunspot modeling problems. This was caused by the function sets for these problems containing commutative functions such as and, or (mux), $+$, and $*$ (cart and sunspot), wherein swapping the order of the arguments did not produce any change in the behavior and the corresponding fitness of the program. As a result, in two out of three applications, the *Swap* operator generated no change in fitness. For the same reason, on these problems, *Swap* ranked last in terms of WP, and had poor IP values. However, as would be expected, the corresponding EI values that were negative moved closer to zero resulting in the *Swap* operator ranking highest on the EI curves. On the artificial ant problem, none of the functions in the function set, $\{\mathrm{If\text{-}food\text{-}ahead}, \mathrm{Prog2}, \mathrm{Prog3}\}$, were commutative and as a result, *Swap* ranked fourth in terms of silent variations, and fifth in terms of IP and WP (smaller ranks imply larger feature values). Over all four problems, the *Swap* operator rarely produced offspring that did better than the population-best at that generation and ranked next to *AllNodes* in terms of IP*.

The *Grow* operator had the highest FCC score on the ant, mux, and cart problems. On these problems, it ranked first (ant and cart) or second (mux) in generating offspring that were better than the best parent at that generation (i.e. in terms of IP*). *Grow* consistently ranked second in terms of the IP and EI. Thus, on the first three problems, the *Grow* operator appears to be very efficient in generating better solutions. On the sunspot problem, the performance of *Grow* in terms of IP*, IP, and EI was average. Except for the sunspot problem, *Grow* ranked eighth in terms of AC, i.e. led to small changes in the fitness space most of the time. Taking into consideration that *Grow* is the only structure changing operator in our investigation that is solely applied to the leaves of the trees, these results coincide with the hypothesis in [Rosca and Ballard, 1995] that the longer the path from the application point of a variation to the root node the less is the probability of large changes in the fitness.The *Grow* operator, i.e. adding code at the leaves of a tree, showed a comparatively low probability of worsening. The effect that on average growing is less destructive than pruning is one reason for the observed code growth in GP.

In the sunspot series modeling problem, the *OneC* and *AllC* operators ranked highest in terms of the IP, EI, FCC, and IP* features. As would be expected *OneC* generated changes that were more local than those generated using *AllC*. As a result, *OneC* ranked higher than *AllC* in terms of the IP, EI and FCC, whereas *AllC* ranked better in terms of IP*. The results show that on symbolic regression problems that involve numeric constants, most of the global improvements are generated by operators used to evolve constants. Structure optimization is obviously harder than parameter adaptation.

It is very important to have operators that can efficiently evolve not only the structure of the symbolic expression but also the right set of constants in the solution. The *OneC* and *AllC* operators appear to be well suited for optimizing such constants during evolution. Most of the changes generated by *OneC* and *AllC*, as depicted by their AC values, were orders of magnitude lower than those generated by all other operators. In the first 100 generations *AllC* generated the highest IP values and was closely followed by *OneC*. However, during the later stages of evolution, as would be expected *OneC* generates larger IP values indicating that smaller changes were preferred in the later stages of evolution.

The *CrossU* operator had the highest IP values on the ant, mux, and cart problems and the differences between the IP values for *CrossU* and the other operators on these problems were statistically significant. In terms of IP*, it ranked first on the artificial ant and mux problems, second on the cart centering, and third on the sunspot problems. Moreover, *CrossU* had the highest expected improvement on the ant problem, and ranked fifth in terms of EI, on the mux, cart, and sunspot problems. The large IP* and IP values indicate that the *CrossU* operator is well suited for generating, with a high probability, offspring, that are better both than their parents and the population-best parent. Consequently *CrossU* appears to be a useful operator on the four problems tested here. Encouraged by the good results, we ran the algorithm described in Section 9.4 with *CrossU* as the only operator used for variation. Again, 50 runs were conducted per test problem. The results are shown in Table 9.7. As the good FD values indicated, the algorithm showed very good performance, not only compared to the results in Table 9.2. Although *CrossU* can be used as the only search operator, this setting does not fulfill the conditions necessary for the theoretical convergence property briefly mentioned in the introduction.

The *CrossSHC* showed consistent above average FD results (this coincides with the good performance shown in [Angeline, 1997a; Angeline, 1997b] and in own experiments) and the *Trunc* operator ranked below average on most features while the *Macro* and *CrossWHC* operators performed poorly on all four problems, but were always better than *AllNodes*. This poor performance of *Macro* which probabilistically uses *AllNodes* appears to be caused by the extremely destructive nature of *AllNodes*.

Most of the differences between the FD features of the headless-chicken operators, i.e. *CrossSHC* and *CrossWHC*, and *CrossU*, were statistically significant. This indicates different dynamics of these closely related operators.

Table 9.7
Computational effort results for the artificial ant, 6-bit multiplexer and cart centering problem using only crossover. The population size was 500 in all the trials. The computational effort corresponds to $z = .99$ except for the multiplexer problem where all runs were successful by generation 56 and the result corresponds to $z = 1.0$. Two of the artificial ant runs failed to find an optimal solution by generation 200 and only 41 of the 50 cart centering trials were successful by generation 200. The reason why not all of the ant and cart centering trials were successful may be premature convergence due to the exclusion of mutation.

	Mean	sd	N	$R(z)$	$I(M, N, z)$
Artificial Ant	88.4	3.71	59	2	60,000
6-bit Multiplexer	64	0	56	1	28,500*
Cart Centering	38.00	4.91	1	19	19,000
Sunspot Modeling	2987.5	431.57		——-	

9.6 Discussion

When evolutionary search is conducted through the use of a set of variation operators, in the most general case, the performance of the algorithm is determined not only by the effects generated by each operator but also by the interactions between these constituent operators. Operators that perform well when used alone might not work constructively when used together. On the other hand, certain operators might synergistically interact with other operators resulting in enhanced performance.

Some operators may not be able to traverse the whole search space of desired computer programs. For example, among the operators investigated, the *OneNode*, *AllNodes*, *Swap*, *Grow*, *Trunc*, *OneC*, and *AllC*, are not capable of searching the entire program tree space when used alone. In such cases there is a need for the design of suites of operators that can traverse the whole program space.

As it is desirable to find as good a solution as possible in reasonable time, operators that simultaneously generate large IP, EI, and IP* values over successive generations are needed. Operators such as *AllNodes* that rarely or never generate offspring better than their parents and the population-best parent may be excluded from the set of operators. However, these FD features do not describe all necessary properties of an EA. For example, operators that show good rates of improvement may increase the speed of the search process but on the other hand may decrease the solution quality due to the loss of diversity in the population which might lead to premature convergence when using certain operators.

Operators such as *Swap* that mostly generate silent variations may be acting as catalysts enhancing the performance of other operators. It might therefore be more appropriate to use them with a lower probability than completely excluding them. Furthermore, efficient evolutionary computations may be designed by varying the probabilities of using various operators such that small changes in fitness are produced more often and large changes are produced less often.

The above analysis of the fitness distributions features indicates that *CrossU*, *Grow*, and *CrossSHC* possess a high probability of generating beneficial variations and variation may be generated by selecting these operators more often. The performance of the *Macro* operator may be enhanced by (a) dropping *AllNodes* form the set of operators used during an application of the *Macro* operator (V_{Macro}), and (b) decreasing the probability of using *Swap*. The *CrossWHC* operator is better suited for exploration and its probability of application may be decreased.

Results from the FD analysis may enable us to tune the evolutionary search, e.g. to make it more exploitative or exploratory. The off-line results of the FD analysis can be used to improve the performance of an EA for program induction, as shown briefly by improving the search performance of the used algorithm by using just one operator that showed over average good FD values. However, the FD of an operator may potentially vary with the set of variation operators being used with it. Additionally, it may depend on the composition of the population (as in the case of crossover).

Instead of an iterated procedure of off-line FD analysis followed by a subsequent change in design of the evolutionary algorithm and further experimentation, the FD analysis may be used on-line. In such an approach the FD results from the previous generation(s) would guide the usage of various operators in subsequent generation(s). Overviews of operator adaptation in GAs can be found in [Tuson and Ross, 1998; Smith and Fogarty, 1997]. The design of such on-line FD analysis tools and procedures in GP remains an area of further research.

9.7 Conclusion

Fitness distributions (FDs) have been utilized as tools for understanding the behavior and dynamics of variation operators when evolving computer programs represented as parse trees. A set of seven dynamic FD features has been proposed that describes the effects of variation operators during evolution. Using these FD features, the behavior of eleven different single- and multi-parent operators has been analyzed on four common benchmark problems. Results indicate that these FD features help us to empirically investigate the dynamic behavior of various operators and contribute to a better understanding of how these variation operators work. Instructive examples were found that show that certain assumptions, such as that *OneNode* generates small changes most of the time or *AllNodes* is useful for search, may not hold. Such enhanced understanding of operator effects may enable us to choose (problem dependent) the appropriate evolutionary algorithm to obtain better solutions and reduce computation time. Future work will be directed towards designing both off-line and on-line methods of incorporating the knowledge obtained from FD analysis for the design of more efficient evolutionary computations.

Acknowledgments

The authors would like to thank M. Kreutz, B. Sendhoff, and P. Stagge for their stimulating discussions and D. B. Fogel for his valuable comments on the work. Christian Igel would like to acknowledge support from the BMBF under grant SONN II 01IB701A0.

Bibliography

Altenberg, L. (1994), "The evolution of evolvability in genetic programming," in *Advances in Genetic Programming*, K. E. Kinnear, Jr. (Ed.), Chapter 3, pp 47–74, MIT Press.

Altenberg, L. (1995), "The Schema Theorem and Price's Theorem," in *Foundations of Genetic Algorithms 3*, L. D. Whitley and M. D. Vose (Eds.), pp 23–49, Estes Park, Colorado: Morgan Kaufmann.

Angeline, P. J. (1996), "An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), pp 21–29, Stanford University, CA: MIT Press.

Angeline, P. J. (1997a), "Comparing subtree crossover with macromutation," in *The Sixth Conference on Evolutionary Programming*, P. Angeline, R. Reynolds, J. McDonnel, and R. Eberhart (Eds.), pp 101–111, Indianapolis, Indiana: Springer-Verlag.

Angeline, P. J. (1997b), "Subtree crossover: Building block engine or macromutation?," in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), pp 9–17, Stanford University, CA: Morgan Kaufmann.

Bäck, T. (1996), *Evolutionary Algorithms in Theory and Practice*, Oxford University Press.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998), *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*, Morgan Kaufmann, dpunkt.verlag.

Chellapilla, K. (1997), "Evolutionary programming with tree mutations: Evolving computer programs without crossover," in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), pp 431–438, Stanford University, CA: Morgan Kaufmann.

Chellapilla, K. (1998), "Evolving computer programs without subtree crossover," *IEEE Transactions on Evolutionary Computation*, 1(3):209–216.

Chellapilla, K. and Fogel, D. B. (1998), "Revisiting evolutionary programming," in *Proceedings of the SPIE: Application and Science of Computational Intelligence*, volume 3390, pp 2–11, Orlando, Florida: SPIE – The International Society for Optical Engineering.

Fogel, D. B. (1995), *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press.

Fogel, D. B. and Ghozeil, A. (1996), "Using fitness distributions to design more efficient evolutionary computations," in *Proceedings of 1996 IEEE Conference on Evolutionary Computation*, pp 11–19, Nagoya: IEEE Press.

Grefenstette, J. J. (1995), "Predictive models using fitness distributions of genetic operators," in *Foundations of Genetic Algorithms 3*, L. D. Whitley and M. D. Vose (Eds.), pp 139–161, Estes Park, Colorado: Morgan Kaufmann.

Iba, H. (1996), "Random tree generation for genetic programming," in *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (Eds.), pp 144–153, Berlin: Springer-Verlag.

Igel, C. (1998), "Causality of hierarchical variable length representations," in *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pp 324–329, Anchorage, Alaska: IEEE Press.

Kinnear, Jr., K. E. (1994), "Fitness landscapes and difficulty in genetic programming," in *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano (Eds.), volume 1, pp 142–147, Orlando, Florida: IEEE Press.

Koza, J. R. (1992), *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA: MIT Press.

Macken, C. A. and Stadler, P. F. (1993), "Evolution on fitness landscapes," in *Lectures in Complex Systems*, L. Nadel and D. L. Stein (Eds.), Santa Fe Institute Studies in the Science of Complexity, pp 43–86, Addison-Wesley.

Manderick, B., d. Weger, M., and Spiessens, P. (1991), "The genetic algorithm and the structure of the fitness landscape," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. K. Belew and L. B. Booker (Eds.), pp 143–150, UCSD, La Jolla, CA: Morgan Kaufmann.

Nordin, P. and Banzhaf, W. (1995), "Complexity compression and evolution," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, L. J. Eshelman (Ed.), pp 310–317, Pittsburgh, PA: Morgan Kaufmann.

O'Reilly, U.-M. (1995), *An Analysis of Genetic Programming*, PhD thesis, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada.

O'Reilly, U.-M. (1997), "Using a distance metric on genetic programs to understand genet ic operators," in *Late Breaking Papers at the Genetic Programming 1997 Conference*, J. R. Koza (Ed.), pp 188–198, Stanford University, CA: Stanford University Bookstore.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1994), *Numerical Recipes in C*, Cambridge University Press, 2nd edition.

Rechenberg, I. (1994), *Evolutionsstrategie '94*, Werkstatt Bionik und Evolutionstechnik, Stuttgart: Frommann-Holzboog.

Rosca, J. P. and Ballard, D. H. (1994), "Hierarchical self-organization in genetic programming," in *Proceedings of the Eleventh International Conference on Machine Learning*, pp 251–258, New Brunswick, NJ: Morgan Kaufmann.

Rosca, J. P. and Ballard, D. H. (1995), "Causality in genetic programming," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, L. J. Eshelman (Ed.), pp 256–263, Pittsburgh, PA: Morgan Kaufmann.

Rudolph, G. (1997), *Convergence Properties of Evolutionary Algorithms*, Hamburg: Kovač.

Schwefel, H. (1995), *Evolution and Optimum Seeking*, New York: John Wiley & sons.

Sendhoff, B., Kreutz, M., and von Seelen, W. (1997), "A condition for the genotype-phenotype mapping: Causality," in *Proceedings of the Seventh International Conference on Genetic Algorithms*, T. Bäck (Ed.), pp 73–80, MSU, East Lansing, MI: Morgan Kaufmann.

Smith, J. E. and Fogarty, T. C. (1997), "Operator and parameter adaptation in genetic algorithms," *Soft Computing*, 1(2):81–87.

Tuson, A. and Ross, P. (1998), "Adapting operator settings in genetic algorithms," *Evolutionary Computation*, 6(2):161–184.

Utecht, U. and Trint, K. (1994), "Mutation operators for structure evolution of neural networks," in *Parallel Problem Solving from Nature III*, Y. Davidor, H.-P. Schwefel, and R. Männer (Eds.), pp 492–501, Jerusalem: Springer-Verlag.

Yao, X. and Liu, Y. (1996), "Fast evolutionary programming," in *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, L. J. Fogel, P. J. Angeline, and T. Baeck (Eds.), pp 451–460, San Diego, CA: MIT Press.

Index