Adaptive Genetic Operators

Vladimir Estivill-Castro
Intelligent Cooperative Information Systems
Research Centre
Queensland University of Technology,
Brisbane, 4000, Australia.
vlad@icis.qut.edu.au

Abstract

Many intelligent systems search concept spaces that are explicitly or implicitly predefined by the choice of knowledge representation. In effect, the knowledge representation serves as a strong bias. Biases heuristically direct search towards favored regions in the search space. Genetic algorithms are a powerful general-purpose search method based on mechanisms abstracted from natural evolution. The effectiveness of the Genetic Algorithm depends heavily on the synergy of the crossover operators and selected representation. In this paper, we discuss the robustness of recombination operators for genetic operators and propose a new family of crossover operators. We present experimental results that indicate these new operators strike a superior balance between exploration and exploitation. We provide an analysis that sheds some light on why the new genetic operators are more effective and show that they are robust towards the encoding scheme.
KEYWORDS: Genetic Algorithms, Search, Bias, Crossover operators, Evolutionary Computation.

1 Introduction

Many intelligent systems search concept spaces that are explicitly or implicitly predefined by the choice of knowledge representation. In effect the knowledge representation serves as a strong bias. Biases heuristically direct search towards favored regions in the search space. Genetic algorithms are a powerful general-purpose search method based on mechanisms abstracted from natural evolution. The effectiveness of the Genetic Algorithm depends heavily on the synergy of the crossover operators and selected representation. Genetic algorithms use a population of chromosomes (genotypes) to individually encode potential solutions (phenotypes). The chromosomes are submerged in a process that simulates some aspects of our understanding of population genetics. With respect to a simulated environment, chromosomes with above average fitness have probabilistically more opportunity of survival. In this case, the fitness with respect to the environment is based on an objective function that evaluates the quality of a chromosome. Fitness is proportional to how good is the encoded solution for the problem being explored. Survival of encoded information to later generations is controlled by *selection* to become a potential parent. Parental chromosomes exchange encoded information via *cross-over* mechanisms that produce new offspring. Offspring are subject to *mutation* before they become chromosomes of the population. The next generation uses a *replacement* rule to remove some older chromosomes and replace them with offspring [3, 6, 7].

Representation is a key issue in genetic algorithms work because genetic algorithms directly manipulate the coded representation of the problem and because the representation scheme provides the window by which the system observes the problem. Most genetic algorithms operate on fixed-length chromosomes that constitute a fixed-size population [3, 6, 7].

Selecting a representation scheme that facilitates solution of the problem by a genetic algorithm often requires considerable insight into the problem and good judgement. In general, there are several characteristics we would like of this mapping between encodings (genotypes) and solution (phenotypes). For example, the encodings must be able to represent desired solutions (otherwise, the algorithm will never find them). However, the encoding mechanism introduces redundancies and/or invalid solutions. These difficulties usually impact the speed of the search and result in a higher demand for computational resources. For example, consider the well-know Euclidian travel-salesman problem. Given a set of n points in the plane, a tour visiting each site (once and only once) with shortest distance is desired. If we label the points with the numerals from 1 to n; a tour may be represented by the list of points in the order they are visited. Since all possible permutations can be represented in this way; this is one possible encoding. However, we quickly observe two problems with this representation. If we have 4 sites, then the lists $\langle 1,2,3,4\rangle$, $\langle 2,3,4,1\rangle$, $\langle 3,4,1,2\rangle$, and $\langle 4,1,2,3\rangle$ are 4 different representations for the same tour (this is the redundancy problem of the representation). Secondly, classical genetic operators, like simple crossover or two point crossover, are not closed in the set of valid tours. For example, the tours $\langle 1,2,3,4\rangle$ and $\langle 4,3,2,1\rangle$ when crossed on the third cut-point results in the offspring $\langle 1,2,3,1\rangle$ and $\langle 4,3,2,4\rangle$. Neither of these represents a valid salesman tour (a site is missing while another is duplicated).

The crucial question is how well do the genetic operators explore the encoded space because of their dependence to physical positions on the chromosome. The classical genetic algorithm uses binary strings as chromosomes. The *building blocks hypothesis* [3, 7, 10] indicates that interacting bits that are relatively close together are less likely to be disrupted. Thus, genes that are thought to interact should be placed near each other on the chromosome. However, there are also problems with finding such representation. In the first place, we may have no knowledge of what features are related and which are not. In fact, Genetic Algorithms are used for large, complex and poorly understood search spaces where there is little or no *a priori* information on the structure of the search space. Second, the linear structure of the binary strings used as chromosomes may be unable to represent the degree of interaction of any two bits in proportion to their proximity in the string.

It is well accepted that some undesired disruption effects occur within any Genetic Algorithm as well as premature convergence of some genes [5]. However, these effects can be tempered by encoding schemes that avoid *Hamming cliffs* [2] and by further understanding of the biases in different genetic operators [5] In this paper, we discuss the robustness of recombination operators for genetic operators and propose a new crossover operator. We present experimental results that indicate this new operator strikes a superior balance between exploration and exploitation. We provide an analysis that sheds light on why the new genetic operator is more effective and show that it is robust towards the encoding scheme.

2 A geometric view of crossover operators

> "*Intuitively, if two computer programs are somewhat effective is solving a problem, then some of their parts probably have some merit. By recombining randomly chosen parts of somewhat effective programs, we sometimes produce new computer programs that are even more fit at solving the given problem than either parent.*" J.R. Koza [9].

Consider the two parents of a crossover operator as suggestions, proposals or partial solutions to the given problem. Those aspects that they have in common are highly likely to appear in their offspring (in some cases, only mutation could change this). Moreover, these common features are also likely to be features of a near-optimal solution. Since parents are usually the result of a fitness-based probabilistic selection, those features in parents are representatives of schema regarded as worth further exploration by the probabilistic selection. In the case of common features of two parents, we have two (probabilistic) votes for those features, and therefore, there is reason to believe their corresponding schema is part of near-optimal solutions. Here, we show that common crossover operators for binary strings actually act on the differences of the parents. Let us illustrate this with the three most popular crossover operators, namely, simple crossover, two-point crossover and uniform crossover.

Simple crossover randomly chooses a cut-point k between 1 and $L-1$, where L is the length of the binary string. Two offsprings are constructed. The first child has bits at positions $0, 1, \ldots, k-1$ from the first parent and bits at positions $k, \ldots, L-1$ from the second parent. Respectively, the second child has those parental bits not used by the first child. Two-point crossover chooses two distinct cut-points k_1 and k_2, now in $\{0, 1, 2, \ldots, L-1\}$. The position of the bits are treated cyclicly by considering position $L-1$ followed by position 0. The first child has the bits in positions $k_1, k_1+1, \ldots, k_2-1$ from bits in respective positions in the first parent while the bits in positions $k_2, \ldots, k_1-1$ come from the bits in corresponding positions of the second parent. Similarly, the second child has those parental bits not used in the first child. Uniform crossover randomly and uniformly selects, for each position, the parent from where to copy the bit. Again, the second offspring uses the parental bits not used by the first off-spring. Figure 1 illustrates these operators.

Note that, for the same parental pair, these crossover operators can potentially produce a different set of offspring. However, those positions with a common bit value are common features transmitted with no alteration to the offspring. In the particular example of Figure 1,

Parents		c_1: 10001011001	
		c_2: 11000010011	
	Simple crossover with $k=6$	Two-point crossover with $k_1=3$ & $k_2=7$	Uniform crossover with mask 01101001110
Bits selected	$\boxed{100010}11001$	$110\boxed{0001}0011$	$\boxed{1}00\boxed{0}1\boxed{01}100\boxed{1}$
	$110000\boxed{10011}$	$\boxed{100}0101\boxed{1001}$	$1\,\boxed{10}\,0\,\boxed{0}\,01\,\boxed{001}\,1$
First offspring	10001010011	10000011001	11000010011
Second offspring	11000011001	11001010011	10001011001

Figure 1: Illustration of crossover operators.

Original space		Difference space
$c_1 = \boxed{100010}11001$	project on positions 1, 4, 7 and 9 $\xrightarrow{\pi_{\{1,4,7,9\}}}$	$\pi_{\{1,4,7,9\}}(c_1) = \boxed{01}10$
$c_2 = 110000\boxed{10011}$		$\pi_{\{1,4,7,9\}}(c_2) = 10\boxed{01}$
$\downarrow$ simple crossover		$\downarrow$ simple crossover
10001010011	$\xrightarrow{\pi_{\{1,4,7,9\}}}$ project on positions 1, 4, 7 and 9	0101
11000011001		1001

Figure 2: Simple crossover commutes with projection.

positions 0, 2, 3, 5, 6, 8 and 10 have a common value in the parents. Therefore, those positions have unaltered values and each offspring will posses these same features as with their parents. That is, the crossover operators are working with the differences. In our example, they are working with positions 1, 4, 7 and 9.

If we project the parental strings into the space of the differences we observe the following. The crossover on the parents of length L corresponds to a crossover on the strings of the differences. Figure 2 illustrates this for our running example with crossover (however, the same commutative diagram is valid for two-point crossover, uniform crossover and other crossover operators like those proposed by Eshelman et al [5]).

Observation 2.1 *If $c : \{0,1\}^L \times \{0,1\}^L \to \{0,1\}^L \times \{0,1\}^L$ denotes a crossover operator for pairs of binary strings of length L and π_I is a projection operator on the set $I \subset \{0,\ldots,L-1\}$ of positions, then $c \circ \pi_I = \pi_I \circ c$. Moreover, when I is the set of positions for which the parents disagree; then every position indicated by I is relevant for defining the offspring and this is the minimum such set of positions.*

This observation carries more meaning when we look at the implications for each crossover

Parents in difference space	$\pi_I(c_1) = 0110$		$\pi_I(c_2) = 1001$		
Cut-point in original space	$k \in \{1\}$	$k \in \{2,3,4\}$	$k \in \{5,6,7\}$	$k \in \{8,9\}$	$k \in \{10\}$
1st offspring in difference space	1001	0001	0101	0111	0110

Table 1: An example of the range of simple crossover.

operator in the difference space (the range of the projection).

Observation 2.2 *In the space of difference positions, the crossover operators can be regarded as choosing the first offspring and then negating each bit of this chromosome to produce the second offspring. That is, the first offspring determines the second offspring by bit-wise negation.*

Because of this observation we consider crossover operators as choosing only the first offspring. In fact, the behavior of crossover operators is the behavior on the difference positions. We characterize such behavior using the following structure. We can consider the space $\{0,1\}^d$ as a hypercube (a graph) of dimension d. In this hypercube, each binary string corresponds to a node and there is a link between two nodes if the respective binary strings differ in just one position (they have Hamming distance 1).

First consider simple crossover. If the chosen cut-point falls before the positions with a difference, then (in the difference space) the first offspring is a copy of the second parent. In our example, the difference space is the subspace of positions 1, 4, 7 and 9 where there is a difference of parental information. If the cut-point had been $k = 1$, then the simple crossover has cut-point $\pi(k)_{\{1,4,7,9\}} = 0$ (occurring in the difference space) and the offspring is 1001 as the second parent (refer to Figure 2). If the chosen cut-point falls before the i-th difference, but before the $(i+1)$ difference, then the offspring is a copy of the first i bits of the first parent and the rest from the second parent when regarded in the difference space. Thus, the behavior is equivalent to a crossover in the difference space with $\pi_I(k) = i$. For example, if $k = 5$ or $k = 6$, the diagram of Figure 2 has the same behavior for the crossover in the difference space with a cut-point at position $2 = \pi_{\{1,4,7,9\}}(k)$. A summary of this with our running example is provided in Table 1. Figure 3 illustrates the hypercube $\{0,1\}^4$. The range of $c_{simple} \circ \pi_{\{1,4,7,9\}}$ consists of the elements in the third row in Table 1. This is the highlighted path from 1001 to 0110 (in difference space, this is a path between the parents).

Thus, we can interpret simple crossover in the geometry of the hypercube as follows. Given two parents p_1 and p_2 with d bit-wise differences, consider the hypercube $\mathcal{H}_d = \{0,1\}^d$ and locate $\pi_I(p_1)$ and $\pi_I(p_2)$ on $\mathcal{H}_d$ by projecting on the positions with differences. Among the $\log_2 d$ shortest paths in $\mathcal{H}_d$ between $\pi_I(p_1)$ and $\pi_I(p_2)$ mark the path that is obtained by starting on $\pi_I(p_2)$ and flipping the bit in the left-most position (position 0), then the second left-most (position 1), and so on until flipping the right-most bit (position $d-1$). Choose the i-th node in this path with probability $(t_i + 1)/(L-1)$ (where t_i is the number of positions where p_1 and p_2 agree between the i-th and the $(i+1)$-th difference). Build the first offspring by using the label of the chosen node on the difference positions (on other

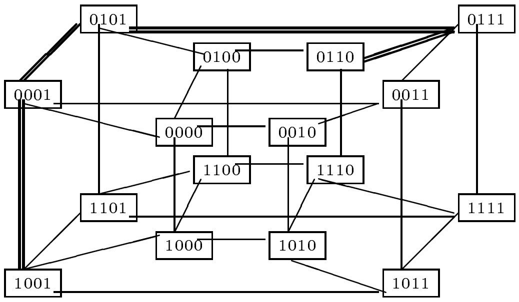


Figure 3: A geometric interpretation of the space of differences.

positions use the respective common value from either parent). The second offspring uses the bit-wise negation of the label of the chosen node on the difference positions.

For our example, the path in $\mathcal{H}^4$ is given by

$$p_2 = 1001 \stackrel{flip(0)}{\longrightarrow} 0001 \stackrel{flip(1)}{\longrightarrow} 0101 \stackrel{flip(2)}{\longrightarrow} 0111 \stackrel{flip(3)}{\longrightarrow} p_1 = 0110.$$

Here $flip(i)$ means move to the neighbor with the i-th position negated (recall we use the convention that the left-most bit is position 0).

We now describe an equivalent interpretation for uniform crossover in the geometry of the hypercube. Given two parents p_1 and p_2 with d differences, consider the hypercube $\mathcal{H}^d$. Among the 2^d nodes in the hypercube, choose randomly and uniformly one node. Build the first offspring by using the label of the chosen node on the difference positions (on other positions use the respective common value from either parent). The second offspring uses the bit-wise negation of the label of the chosen node on the difference positions.

An equivalent interpretation for two-point crossover in the geometry of the hypercube is possible, but somewhat more complex and will be omited here. The range for the bits in the difference position of the first offspring when applying two-point crossover included a path between parents (as simple crossover) but other nodes are also possible. This range is a tree of size $\Theta(d^2)$ nodes that results from ramifications from the range in simple crossover by flipping bits in different orders.

3 Physical independence

> " *In practice, the genetic algorithm is surprisingly rapid, effectively searching complex, highly nonlinear, multidimensional search spaces. This is all the more surprising because the genetic algorithm does not know anything about the problem domain or the internal workings of the fitness measure being used.*" J.R. Koza [9].

Mutation and crossover are the main sources of exploration in a genetic algorithm. Mutation flips bits at random with a uniform distribution along the string and it has little room for bias. Caruana and Schaffer have shown, however, that a hidden bias is introduced by the binary coding of parameters and this can be moderated by Gray coding [2].

Crossover operators provide a more subtle form of exploration and two main types of bias have been identified [5]. The so-called *distributional bias* is observed when the amount of material to be exchanged is concentrated around some value rather than uniformly distributed over the range 1 to $L-1$. *Positional bias* is the creation of new schema dependent on location of the bits in the chromosome. To ameliorate the effects that these biases have on the balance between exploration and exploitation other crossover operators have emerged [5]. Two-point crossover is a well-know example that attempts to free bias towards schemas defined in central positions away from where the chromosome ends and starts [6]. Simple crossover is regarded as an operator with high positional bias but low distributional bias while uniform crossover is regarded as having low positional bias and high distributional bias.

Eshelman *et al* results [5] show that moderating bias can have significant beneficial impacts on performance. In the previous section we revealed more biases in the most common crossover operators. The first type of newly identified bias is a distributional bias. As the population converges, chromosomes would be more similar (they will share the same value for many positions). The difference space would have smaller dimensions. With larger probabilities, the crossover operators will produce offspring that are the untouched parents (although the difference space deserves further exploration). The genetic algorithm will less often evaluate a new individual although it is doing the same amount of work as in early generations (in terms of calls to a random generator or swap of portions of chromosomes). It invests the same amount of computational resources for little exploration and little progress. Thus, this type of bias is clearly undesirable.

The second newly identified bias is a physical bias and is best illustrated by simple crossover. We showed that the i-th node in the range of simple crossover is chosen with probability $(t_i + 1)/(L - 1)$ (where t_i is the number of **positions** where parents p_1 and p_2 agree between the i-th and the $(i + 1)$-th difference.

To ameliorate the effects of this bias we propose the uses of what we call *adaptive genetic operators* that have a uniform distribution in the difference space. There will be an adaptive version for each crossover operator. Thus, two-point cross over, simple crossover or multi-point crossover, each has its adaptive variant. Simply, apply the original crossover operator on the difference space and on all positions common to both parents use the bit from either parent. Consider our running example and simple crossover in Figure 4. The adaptive crossover requires a uniform random choice from a cut point in $\{0, 1, \ldots, d-1\}$ (when there are d differences in the parents) and performs simple crossover in the difference space.

Adaptive crossover operators not only have the effect that less random bits are required (as chromosomes become more alike) with an immediate beneficial impact of computational requirements. It also means that the Genetic Algorithm preserves the same energy for exploration (in the space of differences) during later generations as for early generations. It does not jeopardize exploitation of values in genes since the adaptive version of each genetic operator does not increase the disruptness of the original operator.

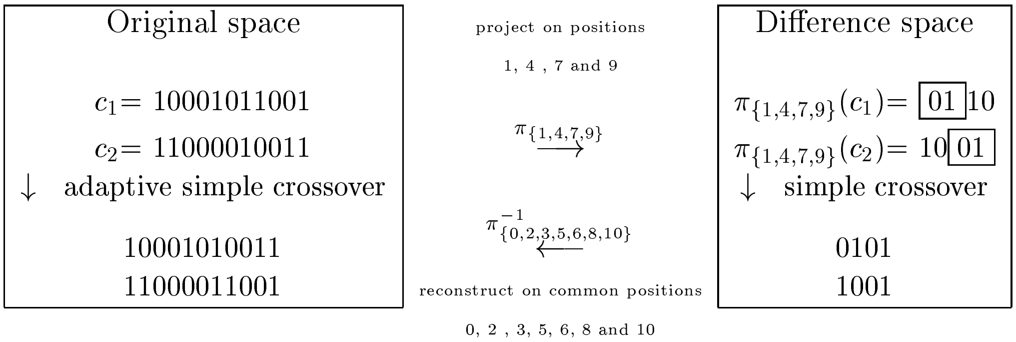


Figure 4: Adaptive simple crossover.

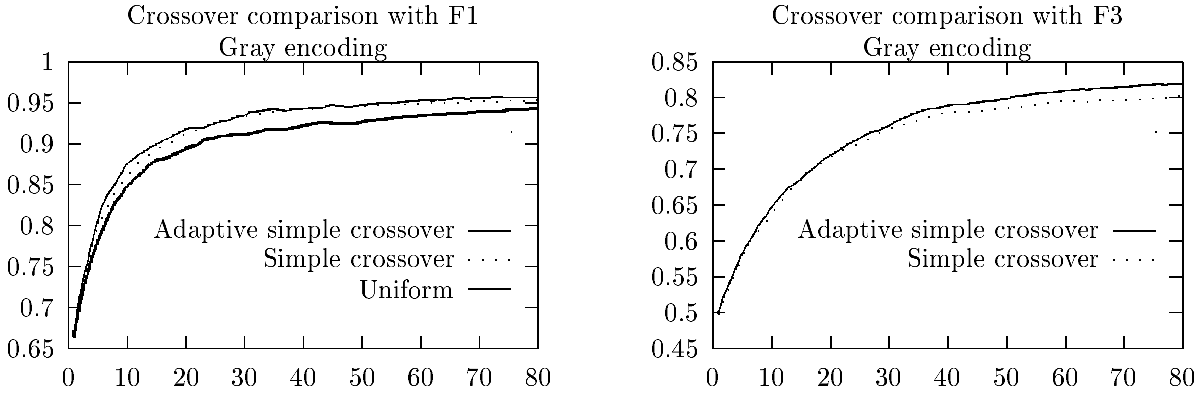


Figure 5: Execution of the GA under gray encoding.

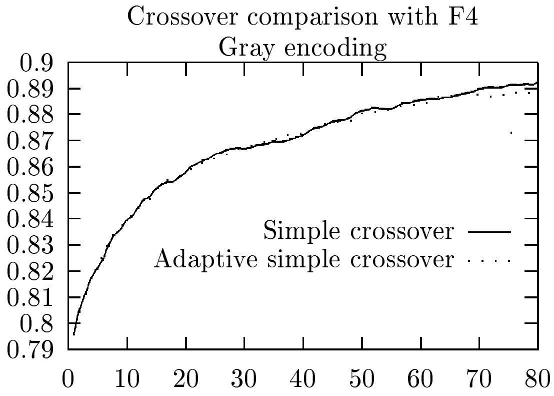
4 Empirical tests

In this section we demonstrate that our analysis and derivation of adaptive genetic operators results in a more effective genetic search.

We have carried out a set of extensive simulations on several objective functions. Here, we present results on a testbed of functions (and search spaces) proposed by De Jong [4] and extensively used among the genetic algorithms community [2, 5, 8, 7, 11, 12] This testbed includes smooth functions (F_1), nonconvex functions(F_2), discontinuos functions (F_3), stochastic functions (F_4), and multimodal (F_2 and F_5).

We denote by $E[Av(P_i)]$ the expected value of the population fitness at generation i. The population fitness $Av(P_i)$ is simply $Av(P_i) = \sum_{i=1}^{n} F(c_i)/n$, where $P_i = \langle c_1, \ldots, c_n \rangle$ is the i-th population. The results are presented as the ratio of $E[Av(P_i)]$ to the maximum of the objective function because:

- this measures the approximation obtained by the algorithm in relative terms and
- in light of our results, this evaluation of the search heuristic has the desirable property



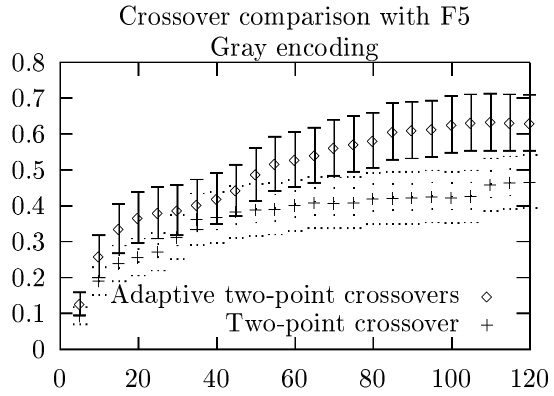


Figure 6: Execution of the GA under gray encoding.

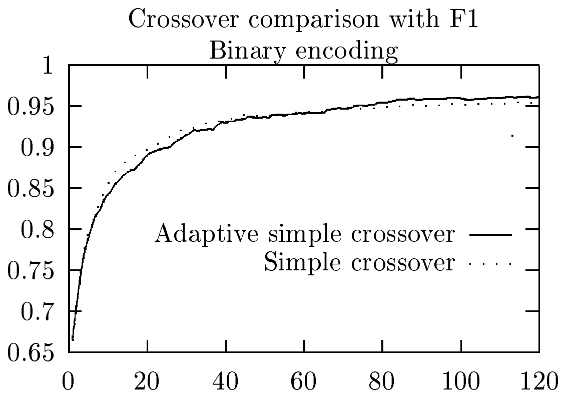


Figure 7: Execution of the GA under binary encoding.

that is invariant to positive scaling [1]. Namely, for a constant $c > 0$,

$$\frac{E[Av(P_i)] \text{ on } cf}{\max_{\mathcal{S}} cf} = \frac{E_{\mathcal{S}}[(cf)^i]}{cE_{\mathcal{S}}[(cf)^{i-1}] \max_{\mathcal{S}} f} = \frac{E_{\mathcal{S}}[f^i]}{E_{\mathcal{S}}[f^{i-1}] \max_{\mathcal{S}} f} = \frac{E[Av(P_i)] \text{ on } f}{\max_{\mathcal{S}} f}.$$

Each combination of parameters was executed 81 times (each execution with a new random seed). In some of the figures presented the curve for other genetic operators has been omitted in an attempt to make the figure clear.

Our first group of results use Gray encoding for representing points in the domain. Gray encoding has been determined as a robust approach for numerical problems [2, 6]. Figure 5 clearly illustrates the more precise solutions obtained by the adaptive version of the genetic operators. The better performance of the adaptive operators occurs across functions $F1$ and F_3. In Figure 6 we present results for F_4 and F_5. In function F_5, the superiority of the adaptive operators is so clear that we have included the illustration of 95% confidence intervals (note that 95% confidence intervals that do not overlap constitutes a test of hypothesis of the

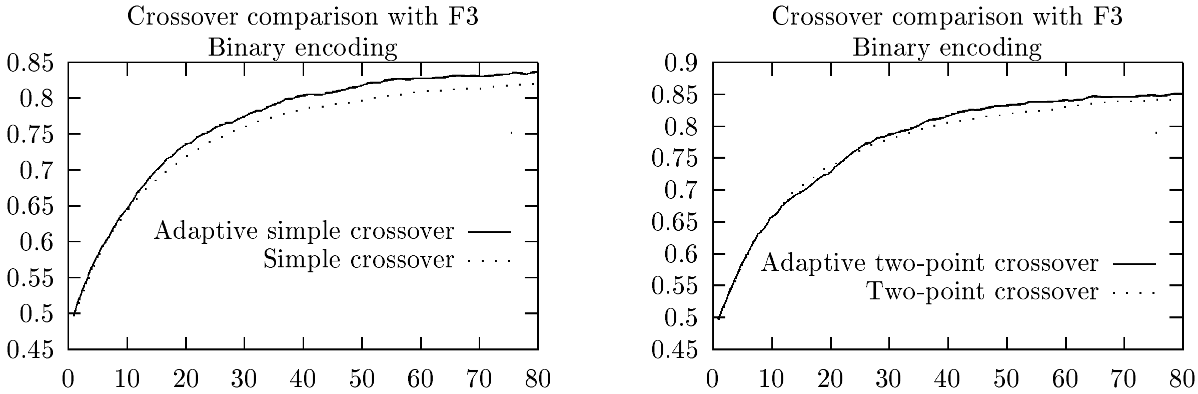


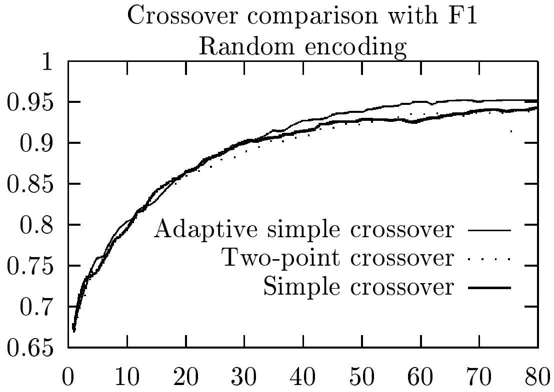
Figure 8: Execution of the GA under binary encoding.

means at least at that level). For function F_4, the adaptive version and the original simple crossover are almost equivalent. Only in latter generations, the adaptive version is slightly less accurate than the original simple crossover. The reader must keep in mind that using the same probability for crossover operators implies that adaptive version will be slightly more disruptive that their ordinary counterparts. This is so because the adaptive version will attempt a crossover in difference space as a way to remain explorative. Function F_4 is such that remaining explorative after several generation has little payoff. However, the reader must keep in mind that adaptive versions require less computational time because of less function evaluations and less futile crossovers. For F_2 both types of operators seem equivalent in terms of precision of $E[A(P_i)]$, however, the same argument applies here in favor of adaptive versions with respect to computational resources.

Using a simple binary encoding for this test bed reveals other behaviors of the adaptive version of genetic operators. Figure 7 illustrates how initially a simple crossover operator may have and apparently higher precision with respect to the optimum. However, this translates into a higher convergence rate to local suboptima and a reduced exploration in favor of exploitation. Because the adaptive version maintains a synergy for exploration, it eventually outperform the original version.

Figure 8 provides two more illustrations of the higher performance of the adaptive versions proposed here with respect to their original counterparts. First, for the simple crossover case and second for the two-point crossover case.

Finally, to demonstrate that the adaptive versions obtain a balance between exploration and exploitation that allows them robust performance in less structured problems we have performed the following experiment. The objective functions remain the same; however, the genetic chromosome is applied a random permutation that moves each gene to a random position within the chromosome. This permutation remains completely unknown to the genetic algorithm and results in objective functions with a domain where groups of interacting chromosomes are not physically related to their position in the chromosome. Methods of adaptive search must contain, at least implicitly, the ability to detect and improve upon



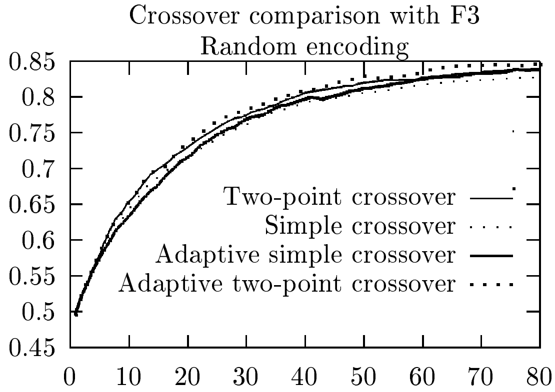


Figure 9: Execution of the GA under encoding masked by random permutation.

non-linearities of related genes in the function optimization process when no information is available through the representation. The adaptive crossover operators demonstrate that they remain robust searchers when information (which could be of value by the representation) is simply not available.

Figure 9 and Figure 10 illustrate the results when the correspondence between a position in the gene and its influence in the objective function is random. In general, the adaptive version are more effective than the original counterparts. The reader must note that in Figure 10, uniform crossover shows the behavior of being a slow starter; however ending with the best solutions towards the end. This not only confirms the uniform crossover is at the other end of the spectrum with low positional bias and high distributional bias. It demonstrates that the adaptive versions are strike a balance in between uniform crossover and the simple, two-point and multi-point crossover operators that usually have high positional bias and low distributional bias. In the experiments with Gray encoding, uniform crossover is outperformed by other operators because of its much higher disruption rate (and more emphasis on exploration). In the random encoding, the search space demands the maximum exploration; thus, the observed pattern for the operators.

5 Concluding remarks

Search mechanisms of artificial intelligence combine two elements: representation, which determines the search space, and a search mechanism, which actually explores the space. Genetic algorithms attribute their success to the synergy between the representation and the combination operators on such representation. The genetic algorithm is more effective when its search bias is adaptively adjusted to the structure of the problem (something that must be also discovered) that with a static representation of the search space.

We have revealed new forms of bias in crossover operators and we have proposed a corresponding aptive genetic operator that reduces physical bias. This allows for more efficient and effective genetic search. If there is information about the structure of the

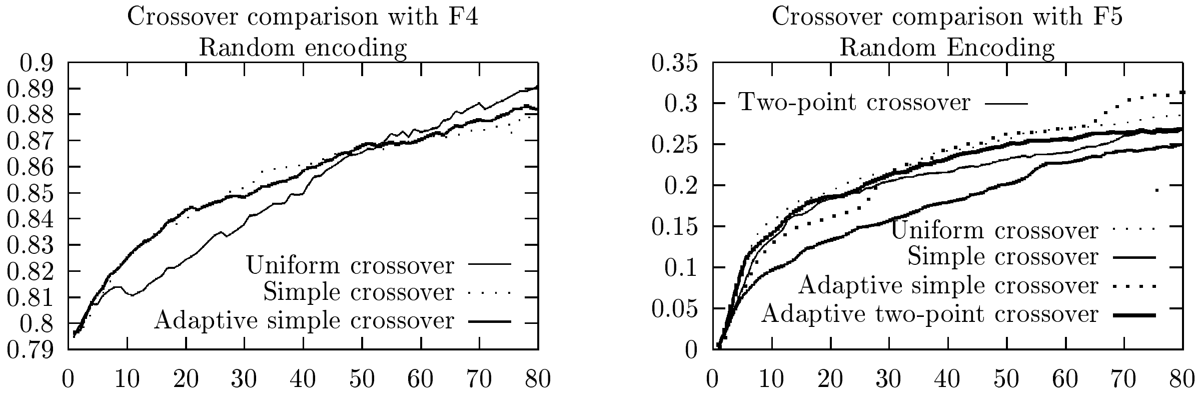


Figure 10: Execution of the GA under encoding masked by random permutation.

search space that can be made explicit with the representation, then such explicit knowledge is of value for the genetic operators and leads to faster and more successful search. However, Genetic Algorithms are usually used in situations when much of the structure of the space is unknown. In such situations the balance between exploration and exploitation should be as independent as possible of physical and distributional biases that may unnecessarily orient the search towards some regions of the space. We have provided here an effective alternative to reduce physical bias in genetic operators like two-point crossover, simple crossover and multipoint crossover. We have also illustrated the diminished physical bias of the proposed operators and their effectiveness in terms of performance per generation. For future study, it remains to illustrate the benefits in computational time of these new operators.

References

[1] D.H. Ackley. An empirical study of bit vector function optimization. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 170–204, Los Altos, Ca., 1987. Morgan Kaufmann Publishers.

[2] R.A. Caruana and J.D. Schaffer. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Proceedings of the Fifth International COnference on Machine Learning*, pages 153–161, Los Altos, CA, June 1988. Morgan Kauffmam.

[3] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[4] K.A. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.

[5] L.J. Eshelman, R.A. Caruana, and J.D. Schaffer. Biases in the crossover landscape. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic*

Algorithms, pages 10–19, San Mateo, CA., 1989. George Mason University, Morgan Kaufmann Publishers.

[6] D.B. Fogel. *Evolutionary Computation — Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, US, 1995.

[7] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Co., Reading, MA, 1989.

[8] J.J. Grefenstette. Optimization control parameters for genetic algorithms. *IEEE Transactions on Systems, Man & Cybernetics*, 16(1):122–128, January-February 1986.

[9] Koza J.R. Introduction to genetic programming. In K.E. Jr. Kinnear, editor, *Advances in Genetic Programming*, pages 21–42, Cambridge, MA, US, 1994. MIT, MIT Press.

[10] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Heidelberg, Berlin, second, extended edition, 1992.

[11] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as a function optimizer. In R. Belew and L. Booker, editors, *Proceedings of the Fourth Int. Conf. on Genetic Algorithms*, pages 271–278, Los Altos, CA., 1991. Morgan Kaufmann Publishers.

[12] G. Syswerda. Uniform crossover in genetic algorithms. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, San Mateo, CA., 1989. George Mason University, Morgan Kaufmann Publishers.