Adaptive Linkage Crossover

Ayed A. Salman
Department of ECE
Kuwait University
Kuwait
ayed@eng.kuniv.edu.kw

Kishan Mehrotra
Department of EECS, 2-177 CST
Syracuse University
Syracuse, NY 13244-4100, USA
kishan@ecs.syr.edu

Chilukuri K. Mohan
Department of EECS, 2-177 CST
Syracuse University
Syracuse, NY 13244-4100, USA
mohan@ecs.syr.edu

Abstract

Problem-specific knowledge is often implemented in search algorithms using heuristics to determine which search paths are to be explored at any given instant. As in other search methods, utilizing this knowledge will more quickly lead a genetic algorithm (GA) towards better results. In many problems, crucial knowledge is not found in individual components, but in the interrelations between those components. For such problems, we develop an interrelation (linkage) based crossover operator that has the advantage of liberating GAs from the constraints imposed by the fixed representations generally chosen for problems. The strength of linkages between components of a chromosomal structure can be explicitly represented in a linkage matrix and used in the reproduction step to generate new individuals. For some problems, such a linkage matrix is known *a priori* from the nature of the problem. In other cases, the linkage matrix may be learned by successive minor adaptations during the execution of the evolutionary algorithm. This paper demonstrates the success of such an approach for several problems.

Keywords

Genetic algorithms, crossover operators, deception, linkage probabilities, adaptation.

1 Introduction

What is the role of crossover in a GA? Current explanations mention maintaining diversity in the population while preserving good building blocks. What constitute good building blocks? The Schema Theorem (Holland, 1975) and the Building Block Hypothesis (Goldberg, 1989) imply that successive generations contain an exponentially increasing number of instances of short, low-order, high (sampled) fitness schema, assuming the use of 1-point crossover (1PTX) and fitness-proportionate reproduction selection. However, these results do not say anything about schema whose order or defining length is nontrivially large. If a problem is such that the first and last components of individuals need to evolve together, such linkages are so likely to be disrupted that an operator such as 1PTX becomes useless at maintaining such linkages. As in the case of any other (general-purpose) operator, there are as many problems for which 1PTX works well as for which 1PTX does not, paraphrasing the *No Free Lunch Theorems* (Wolpert and Macready, 1997).

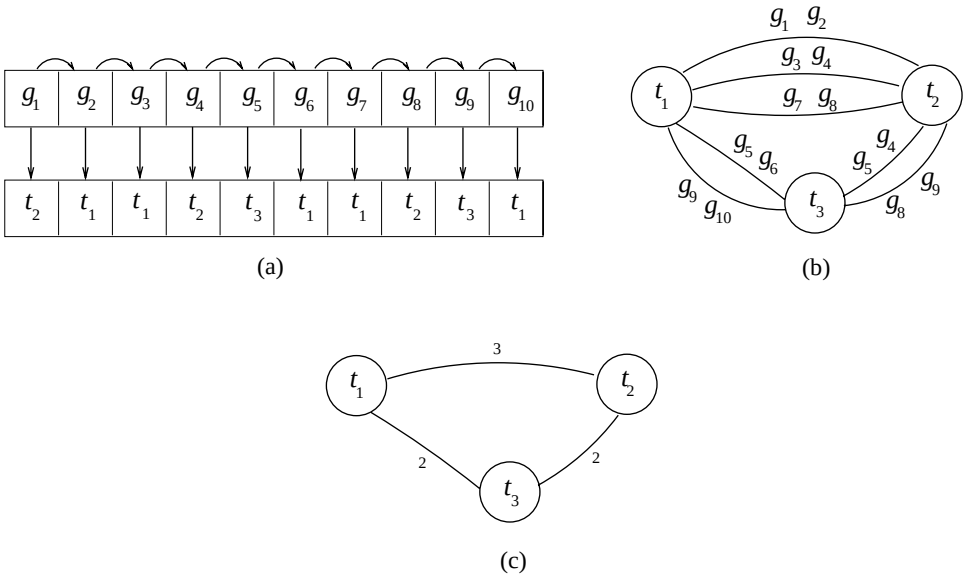


Figure 1: (a) Simple gene-level linkages; pleiotropy and polygeny trait representation, (b) Trait-level linkages; adjacencies in gene level representation, (c) Abstraction of linkage strengths between traits.

Some deceptive problems can be solved using linear transformations that change representation (Liepins and Vose, 1991) and hence linkage structure. Harik and Goldberg (1997) have formulated a "Linkage friendly" crossover operator very similar to the two-point crossover operator. However, discovering the linkages of distant components is difficult. By contrast, we develop operators that work well by exploiting known linkages between the components of chromosomes.

The incorporation of *pleiotropy* (one gene affects multiple traits) and *polygeny* (many chromosomal components are responsible for a single trait) remains largely unexplored by researchers in evolutionary computation, although these are believed to exist in "all systems with complex behavior" (Atmar, 1992). An abstract interpretation of pleiotropy, ploygeny, and linkages is shown in Figure 1. Is there any advantage to such complicated many-to-many (gene-to-trait) coding mechanisms? We hypothesize that such mechanisms do serve an additional purpose equivalent to encoding "linkage probabilities" that determine the likelihood with which different traits are inherited from the same parent during crossover.

Distributed representations carry the advantages of fault tolerance and robustness, which partially explains the method behind the madness of pleiotropy and polygeny in biological evolution. In addition, we argue that these mechanisms indirectly code for elaborate representations of linkages between traits, transcending the limitations of the linear sequencing of genes. If we view nature's algorithm in terms of traits rather than genes, what emerges is not a simple string (linear sequence) in which each element (trait) is connected to its immediate neighbors. Instead, we find a complex of multiple connections between different traits implemented by the connections between multiple genes (for each trait) within the linear chromosomal structure.

By analogy with nature, a GA with a "one-gene-per-trait" (or a "one-sequence-of-genes-per trait") representation must not rely merely on a sequential arrangement of the genes with implied strong linkages among only neighboring genes. In order to be effective, a GA must instead allow for the representations of multiple connections between traits. We suggest that this be accomplished via "linkage probabilities": each individual in the population is a collection of traits whose "sequencing" is irrelevant. In keeping with this understanding, we may use the term "individual" instead of the traditional "chromosome" in reference to the members of the population. The population consists of a number of individuals as well as a representation of the species-specific linkages between traits. Simplicity of implementation and computational considerations dictate the use of a two-dimensional array of first-order linkage values, which is adequate for this purpose. If we use a "one-gene-per-trait" encoding, then such linkage probabilities must be explicitly stated and used by operators of the GA. We propose a crossover operator (LinX) that uses linkage information. We also propose linkage adaptation algorithms that prove to be successful in learning the appropriate linkage matrix during application of the genetic algorithm.

In order to show the effectiveness of our operators, we compared them with the well-known one-point, two-point, and uniform crossover operators. Our test set was composed of four problems. Three of the problems were theoretically constructed to be GA-deceptive, yet have a well-known linkage structure: *Goldberg's order-three problem* (Goldberg et al., 1989), *bipolar order-six problem* (Goldberg et al., 1992), and *Mühlenbein's order-five problem* (Pelikan and Mühlenbein, 1998). The fourth is the well-known *graph bipartitioning problem*, whose exact linkage structure is not known a priori. Experiments show the superiority of linkage crossover over other operators.

Related work is briefly summarized in Section 2. The proposed framework is described in Section 3. Section 4 defines a class of crossover operators using this approach and the relation of these to crossover operators traditionally used in GAs. Section 5 elucidates our algorithms. Section 6 describes benchmark problems used in the experiments. Section 7 presents experimental results that support our new approach, followed by concluding remarks in Section 8. An abbreviated, early version of this paper appears in Salman et al. (1998). Additional experimental results are contained in Salman (1999).

2 Related Work

Problem-specific knowledge can be utilized by genetic search in several ways, one of which addresses the "move-generation" step: how can we define genetic operators that utilize knowledge specific to each class of problems? Maini et al. (1994a) have addressed two ways of doing this: incorporating allele-specific biases and utilizing information about the history of genetic search. Another simple approach is found in the 'particle swarm optimization' system (see Kennedy and Eberhardt (1995)). This paper explores a third, crucial approach, the development of operators that exploit linkages between components of problem representations.

Holland (1975) proposed the "inversion" operator to overcome the linkage problem in GAs. Inversion reverses a substring of the chromosome, allowing for the possibility that genes that should have tight linkage are brought closer together. Unfortunately, this idea has been shown to be ineffective and much slower than the population convergence rate (Goldberg and Lingle, 1985).

Schaffer and Morishima (1987) encoded crossover "flags" within the chromosome

representation that serve as crossover points. A flag also evolves with GA execution, moving from one position to another. They reported superior performance when compared to the standard GA, However, this solution does not address the problem of nonlinear linkages and is very sensitive to parameter values such as the number of flags in practical problems with unknown building block size.

Another serious attempt to solve the linkage problem was by Goldberg et al. (1989) who introduced the Messy Genetic Algorithm (mGA). This algorithm works differently than the simple GA. The messy GA uses an order-based encoding, where a gene is represented as a pair of values: allele and locus. It consists of a *primordial* phase followed by a *juxtapositional* phase. In the primordial phase, building blocks are identified, under the pressure of selection, from a large pool of building blocks of different lengths. In the juxtapositional phase, the building blocks are recombined using a "cut-and-splice" operator. This operator mimics the action of one-point crossover on tightly linked building blocks. Deb (1991) empirically showed that a messy genetic algorithm is capable of solving a variety of deceptive problems in $O((|A|.n)^k)$ function evaluations, where $|A|$ is the cardinality of the alphabet A, n is the problem size, and k is the order of deception.

The large number of function evaluations required is a major problem for messy GAs. Several improvements were later made on the messy GA in order to speed up the primordial phase (Goldberg et al., 1993); the complexity of the resulting Fast Messy GA is subquadratic in problem size. Nevertheless, the Fast Messy GA is highly sensitive to numerous exogenous parameters for which no practical selection methodology is known.

Kargupta (1996) and Bandyopadhyay et al. (1998) used a messy genetic algorithm to learn the linkages between different genes. A list of genes is swapped from the first parent, which acts as a donor, to the second parent, a receiver. The resulting change in fitness is observed. If the fitness increases, this *linkage list* is rewarded in the form of increasing the probability, called *goodness*, of this list to be selected more often; otherwise, it is punished. Their algorithm, called Genes Expression Messy Genetic Algorithm (GEMGA), was tested on several large deceptive problems. They reported observing, empirically, a linear relation between the number of function evaluations and problem size. GEMGA assumes that the population size is large enough to contain at least one instance of each optimal building block. Similar to our approach, a conditional probability matrix is computed, although its entries are thresholded using the maximum value in each row (minus a programmer-defined constant ϵ), since the collection of genes is presumed to be partitioned into nonoverlapping subsets. However, this approach does not capture the possibility of linkage connections of arbitrary strengths. A more general linkage framework, such as the one we propose, would allow for the possibility that gene linkages may be of arbitrary strengths, e.g., in a variation of Goldberg's order-3 problem (described in Section 6.1) wherein the fitness of a chromosome is the sum of the results of applying an order-3 subfunction to overlapping substrings of the chromosome as in

$$f(x_1 x_2 \ldots x_n) = \sum_{i=1}^{n-2} f_3(x_i x_{i+1} x_{i+2})$$

where f_3 may be chosen to be a deceptive function. If linkages are presumed to be transitive, as in the GEMGA approach, this problem would appear to be of order n, requiring very large population sizes. By contrast, our approach distinguishes between strong linkages (between x_i and x_{i+1}), intermediate linkages (between x_i and x_{i+2}), and weak linkages (between x_i and x_{i+2+j} for $j > 0$).

Kemenade (1998) introduced a three-phase genetic algorithm called Building Blocks Filtering GA (BBF-GA), which is very similar to the messy GA. In the first phase, Kemenade evolved several rapidly converging GA subpopulations to produce several high fitness solutions. In the second phase, those solutions were filtered to extract good building blocks. In the last phase, he masked the building blocks to prevent disruption during recombination. Tests of his algorithm on a trap deceptive problem and on an NK-landscape problem showed that it was superior to other algorithms with respect to the number of optimal building blocks discovered, as well as the average fitness for the best solutions obtained. Nevertheless, Kemenade's algorithm requires a large amount of computation to execute its three phases.

Corcoran (1994) developed three techniques for preserving good building blocks during recombination. He theorized that the more disruptive the recombination, the more suitable it is for order-based problems; but, on the other hand, the less preservative it will be for the problem's building blocks. This motivated him to introduce three techniques, *reduction*, *masking*, and *bonding*, of which bonding is the most general. In this technique, Corcoran represented the bonding, or linkage, between different genes by a directional vector attached to each chromosome in the population. These vectors direct the genetic algorithm to move genes specified in the bonding vector together during recombination. The bonding technique requires a bonding matrix for every chromosome in the population, which does not scale well with problem size. Also, it does not address problems where the bonding between different genes is not clear in advance.

In work similar to Corcoran's, Smith and Fogarty (1996) realized that bonding good building blocks together to prevent disruption (during recombination) should enhance the performance. In order to achieve this, they attached two flags, "left" and "right," to each gene, which symbolized whether it was linked to its left or right neighbor. They studied the relation between the epistasis of the problem and the structure of the population under their algorithm. They found that the number of genes that tend to form building blocks is proportional to the degree of the epistasis. This technique lacks the ability to link nonadjacent genes.

Gero and Kazakov (1995) used a pattern recognition technique to identify those building blocks that are useful and those that are harmful. These were extracted from extremely fit and unfit chromosomes in the population. Useful genes are then bonded together and protected from destruction by crossover and mutation. On the other hand, harmful genes are forced to undergo a high rate of mutation. This technique was shown to be plausible on several test problems. This technique is sensitive to the chromosomes used to identify the useful and harmful building blocks.

Using an idea drawn from biological genetic noncoding regions, Levenick (1991, 1995) showed that positioning fitness-neutral "introns" or "meta-bits," between different building block components increased the GA's success rate on some test problems. These introns act as potential crossing points during recombination. For introns to be effective, the genetic algorithm should learn where to put them. Levenick developed a mechanism for positioning introns during the execution of the genetic algorithm, hoping that selection pressure would bring the right manifestation of the introns into place. The early results of this approach were unsatisfactory because the population converges faster than the rate at which introns took their proper places. He then used a diversity-enhancing technique, resource sharing, to slow down the convergence rate, showing that the introns eventually settled between the problem's building blocks. This approach suffers from the same problems as the Schaffer

and Morishima approach, namely, only linear linkages can be represented in this way, and only problems with a known number of building blocks can be solved.

Paredis (1995) distinguished between a problem solution and its ordering. In a co-evolutionary genetic algorithm, he evolved two populations, one to obtain the problem solutions and the other to evolve suitable orderings of genes. At each recombination step, two parents from the first population were selected for mating, and an ordering from the second population was selected to be used in mating them. The ordering population was evaluated based on how fit the children of the crossover were, compared to their parents. Paredis tested his algorithm for two different problems: an order-3 deceptive problem and an epistatic problem. His algorithm showed overall superiority to a simple genetic algorithm, but it also exhibited poor performance when the degree of epistasis was high. He concluded that for a maximum degree of epistasis, where the fitness landscape is rugged and uncorrelated, random search would be a better choice than a GA.

Harik (1997) introduced the concept of learning linkage. In Harik's terminology, the linkage between two genes is the probability that those genes escape separation by one-point crossover during recombination. He used an order-based representation where each gene is represented by its locus and an allele, which move together in the chromosome. Using an exchange operator very similar to two-point crossover, he tried to juxtapose tightly linked genes close to each other so that they would have a better chance of escaping disruption by crossover. Harik observed that there is some kind of "race" between selection force and learning linkage, hence he introduced a diversity-enhancing tournament selection mechanism to cope with the slower linkage learning. Harik reported failure for deceptive problems with uniformly weighted building blocks.

Pelikan and Mühlenbein (1998) proposed a Bivariate Marginal Distribution Algorithm (BMDA) that uses pairwise gene dependencies to approximate a bivariate probability distribution of a problem. This distribution was then used to generate new individuals. The BMDA requires a large population size to estimate the probability distribution accurately.

Mühlenbein et al. (1998) proposed the Factorization of Distribution Algorithm (FDA), which uses problem structure to its advantage. The FDA estimates the probability distribution of the interdependencies between a problem's variables by assuming that it can be factorized into the product of different bivariate distributions (i.e., distribution of dependencies between pairs of genes). This estimate is used to generate new strings probabilistically rather than by standard recombination and mutation. The FDA suffers from two shortcomings: it requires a large population size and it must have prior knowledge about the problem structure.

For some problems, the linkage structure may be obvious and can be well understood a priori; many of the approaches discussed above do not use existing linkage information directly when it is available. Also, many of these approaches do not allow for the representation and use of different degrees of linkages, which is important if linkages are to be learned by a process of incremental modification. Finally, some of these techniques assume associative linkage between different genes (e.g., among three genes, if the first is linked to the second and the second is linked to the third, then it is necessary that the third should be linked to the first); such a relation is not always true (e.g., in the NK-landscape problems). In the following sections, we propose an approach that does not suffer from these shortcomings.

3 Framework

This section describes the overall framework establishing the relation between crossover and probabilistic inference. In our notation, $i \leftarrow p_j$ denotes the event that the ith gene in the offspring comes from the jth parent when crossover occurs, π denotes a partial inheritance assignment, and $\pi(i)$ denotes the parent p_1 or p_2 from which an offspring inherits the ith gene. We assume that there is no a priori bias toward either parent, p_1 or p_2, i.e., symmetry is assumed.[1]

Without loss of generality, we assume that parents p_1 and p_2 generate only one offspring.[2] In examining which genes are inherited from which parent, we restrict attention to those genes where the parents differ. Also, let $\{x_1, x_2, \ldots, x_n\}$ denote a permutation of $\{1, \ldots, n\}$, where n is the number of genes (components) at which parents differ. Each x_i is merely a label that identifies a gene; whether the genes are arranged linearly (or in some other manner) is irrelevant to our approach.

Classical crossover operators break linkages among some genes, ignoring potential dependence among them, whereas it would be desirable to use a crossover operator that respects special attractions between sets of alleles. This dependence can be reformulated in terms of problem-specific conditional probabilities. Specifically, the crossover operator should address the probability of inheriting a gene from one parent, given that some other genes have been inherited. In other words, if $x_1, x_2, \ldots, x_i$ are the positions of genes inherited from $\pi(x_1), \pi(x_2), \ldots, \pi(x_i)$, respectively, then what is the probability that the x_{i+1}th element of the offspring is inherited from p_1? We view crossover as accomplishing this probabilistic inference task, where the probability depends on the problem, and is "hard-coded" into biological chromosomal structures via the mechanisms of pleiotropy and polygeny.

Special Cases:

One-Point Crossover

In one-point crossover (1PTX), the structure of linkages is linear, similar to probabilistic inference with a chain structure. The only linkage between the $(i+1)$th gene and the $(i \Leftrightarrow 1)$th gene is through the ith gene. The linkages associated with 1PTX may be described in the following manner:

$$P(i \leftarrow p_1 \mid (i+a) \leftarrow p_1 \;\&\; (i \Leftrightarrow b) \leftarrow p_1) = 1, \text{ where } a > 0, b > 0,$$

and

$$P(i \leftarrow p_1 | (\forall a > 0.\ [(i+a) \leftarrow p_2]) \;\&\; (\forall b > 0.\ [(i \Leftrightarrow b) \leftarrow p_1])) = 0.5.$$

One-point crossover is expected to work well when the linkages in the problem are of a linear nature, e.g., when the desirable building blocks consist of alleles for physically proximate genes.

[1] For genes at which both parents have the same allele, the ith allele of the offspring may be identical to that of parent p_1 even though $i \leftarrow p_2$.

[2] Operators that generate multiple offspring can be viewed as the results of multiple applications of operators that generate single offspring. For instance, one-point crossover applied to p_1, p_2 at position k is defined to produce two offspring but can be viewed as equivalent to two separate applications of an operator that generates only one offspring; the second application is obtained by reversing the order of the parents.

Uniform Crossover

In uniform crossover (UX), no linkages are preserved.

$$P(i \leftarrow p_1 | j_1 \leftarrow \pi(j_1) \& \ldots \& j_k \leftarrow \pi(j_k)) = 0.5, \forall j_\ell \neq i, \forall k < n.$$

Whether 1PTX or uniform crossover works better on a problem depends on whether the problem itself has implicit linkages of the kind preserved by 1PTX.

4 Linkage Crossover

A general class of crossover operators can be formulated using the framework of linkage probabilities. This class is referred to as General Linkage Crossover (GLinX), and is described below. We use the following additional notation:

- $P(x_{i+1} : x_1, \ldots, x_i; \pi)$ denotes the conditional probability that the x_{i+1}th position in the child chromosome comes from parent p_1, given that the x_jth position comes from parent $\pi(j)$ for $j = 1, \cdots, i$, i.e.,

$$P(x_{i+1} : x_1, \ldots, x_i; \pi) = P\Big(x_{i+1} \leftarrow p_1 | x_1 \leftarrow \pi(x_1) \& \ldots \& x_i \leftarrow \pi(x_i)\Big).$$

- For the special case when $x_1, \ldots, x_i$ are all inherited from p_1, the linkage probability $L(x_{i+1} : x_1, \ldots, x_i)$ denotes $P\Big(x_{i+1} \leftarrow p_1 \mid x_1 \leftarrow p_1 \ \& \ \ldots \ \& \ x_i \leftarrow p_1\Big)$.

Computation of Offspring Components Using GLinX

Suppose p_1 and p_2 differ from each other in n locations. Let $x_1, \ldots, x_n$ denote some permutation of these locations.

- Offspring alleles for positions x_1 and x_2 are inherited from p_1 and p_2, respectively.
- Offspring alleles for the remaining $(n \Leftrightarrow 2)$ locations are successively assigned as follows: Suppose i additional locations, $x_3, \ldots, x_{i+2}$, have been assigned alleles (from p_1 or p_2) so far. Then, the $(i+3)$th component of the offspring is inherited from parent p_1 with probability $P(x_{i+3} : x_1, \ldots, x_{i+2}; \pi)$.

EXAMPLE 1: Consider $p_1 = (0,0,1,0,0), p_2 = (0,1,0,1,1)$, differing in the last four positions ($n = 4$). The first position in the offspring is assigned 0, common to both parents. Let $(x_1, x_2, x_3, x_4) = (2,4,3,5)$. The second ($x_1$th) position in the offspring is chosen from parent p_1, and the fourth (x_2th) position in the offspring is chosen from p_2. Next, the third (x_3th) position in the offspring is chosen from p_1 with probability $P(x_3 : x_1, x_2; \pi) = P(x_3 \leftarrow p_1 | x_1 \leftarrow p_1 \& x_2 \leftarrow p_2)$. Suppose it is chosen from p_1. Finally, the fifth (x_4th) position in the offspring is chosen from p_1 with probability $P(x_4 : x_1, x_2, x_3 : \pi) = P(x_4 \leftarrow p_1 | x_1 \leftarrow p_1 \& x_2 \leftarrow p_2 \& x_3 \leftarrow p_1)$. □

Only in the ideal case would probabilities $P(x_{i+1} \leftarrow p_1 | x_1 \leftarrow \pi(x_1) \& \ldots \& x_i \leftarrow \pi(x_i))$ be available for each i. There are far too many joint linkage probabilities to be specified, and these would be impossible to specify even for problems whose nature is relatively well understood. For specific problems, a dependency structure may be available, enabling calculations of such quantities. In practice, these have to be estimated or approximated based

on limited information, a task similar to that of probabilistic reasoning with uncertainty in expert systems while making conditional independence assumptions. For instance, the expert systems literature addresses the estimation of $P(A|B\ \&\ C)$, given only $P(A|B)$ and $P(A|C)$ along with the prior probabilities.

Invoking conditional independence assumptions, we develop a crossover operator that makes use of first-order *pairwise linkages* $L(x_i : x_j) = P(x_i \leftarrow p_k | x_j \leftarrow p_k)$. Information about such linkages is most likely to be available as domain knowledge for practical problems. For instance, in the graph partitioning problem, the connection weight between two nodes suggests a choice for the corresponding linkage probability. Since p_1 and p_2 are arbitrary, conditional symmetry prevails, i.e.,

$$L(x_j : x_i) = P(x_j \leftarrow p_1 | x_i \leftarrow p_1) = P(x_j \leftarrow p_2 | x_i \leftarrow p_2).$$

We assume that the problem description specifies the first order linkage probabilities, $L(i : j)$, for each i, j; no other information is available. Other probabilities such as $P(x_{i+1} : x_1, \ldots, x_i;\ \pi)$, need to be estimated from $L(x_{i+1} : x_1), \ldots, L(x_{i+1} : x_i)$.

This is analogous to the expert system's task of combining the conclusions obtained from multiple sources of uncertain knowledge. For any two events A and B, Bayes rule gives

$$\begin{aligned} P(A|B) &= \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\overline{A})P(\overline{A})} \\ &= \frac{P(B|A)}{P(B|A) + o(\overline{A})P(B|\overline{A})}, \end{aligned} \tag{1}$$

where $o(\overline{A}) = P(\overline{A})/P(A)$ represents the prior odds of occurrence of $\overline{A}$. Applying Equation (1) to the problem of interest gives:

$$P(x_{i+1} : x_1, \ldots, x_i; \pi) = P(x_{i+1} \leftarrow p_1 |\ \cap_{j=1}^{i} x_j \leftarrow \pi(x_j))$$

$$= \frac{P(\cap_{j=1}^{i} x_j \leftarrow \pi(x_j) | (x_{i+1} \leftarrow p_1))}{P(\cap_{j=1}^{i} x_j \leftarrow \pi(x_j) | (x_{i+1} \leftarrow p_1)) + o(x_{i+1} \leftarrow p_2) P(\cap_{j=1}^{i} x_j \leftarrow \pi(x_j) | (x_{i+1} \leftarrow p_2))} \tag{2}$$

It would be reasonable to replace the odds ratio $o(x_{i+1} \leftarrow p_2) = P(x_{i+1} \leftarrow p_2)/P(x_{i+1} \leftarrow p_1)$ with 1; there is no a priori preference that the allele in the $i+1$th position of the offspring should come from one parent (p_1) or the other (p_2). In the rest of the development, we assume that the prior probabilities are the same for inheriting any component from either parent.

Conditional Independence Assumption

Using this assumption, one writes

$$P(\cap_i A_i | C) = \prod_i P(A_i | C)$$

for arbitrary events $(C, A_1, A_2, \ldots)$. In the present context, it is assumed that

$$P(x_j \leftarrow p_1\ \&\ x_k \leftarrow p_1 \mid x_{i+1} \leftarrow p_1) = P(x_j \leftarrow p_1 \mid x_{i+1} \leftarrow p_1) P(x_k \leftarrow p_1 \mid x_{i+1} \leftarrow p_1),$$

and

$$P(x_j \leftarrow p_1 \;\&\; x_k \leftarrow p_1 \mid x_{i+1} \leftarrow p_2) = P(x_j \leftarrow p_1 \mid x_{i+1} \leftarrow p_2)P(x_k \leftarrow p_1 \mid x_{i+1} \leftarrow p_2).$$

Application of conditional independence assumption to Equation (2) gives

$$P(x_{i+1} : x_1, \ldots, x_i; \pi) =$$

$$\frac{\prod_{j=1}^{i} P(x_j \leftarrow \pi(x_j) | x_{i+1} \leftarrow p_1)}{\prod_{j=1}^{i} P(x_j \leftarrow \pi(x_j) | x_{i+1} \leftarrow p_1) + \prod_{j=1}^{i} P(x_j \leftarrow \pi(x_j) | x_{i+1} \leftarrow p_2)} \tag{3}$$

For the purpose of easy evaluation this expression can be further simplified.

As considered earlier, parents p_1 and p_2 differ in genes $\{x_1, \ldots x_n\}$. In an offspring of p_1 and p_2, some of the genes in $\{x_1, x_2, \ldots, x_i\}$ are inherited from parent p_1, and others from p_2. Let $S_{i,1}$ be the set of genes inherited from parent p_1 and $S_{i,2}$ the set of genes inherited from parent p_2. Then Equation (3) can be written as

$$P(x_{i+1} : x_1, \ldots, x_i; \pi) = \frac{h_1}{h_1 + h_2}$$

where

$$\begin{aligned} h_1 &= \prod_{x_j \in S_{i,1}} P(x_j \leftarrow \pi(x_j) | x_{i+1} \leftarrow p_1) \prod_{x_j \in S_{i,2}} P(x_j \leftarrow \pi(x_j) | x_{i+1} \leftarrow p_1) \\ &= \prod_{x_j \in S_{i,1}} L(x_j : x_{i+1}) \prod_{x_j \in S_{i,2}} (1 \Leftrightarrow L(x_j : x_{i+1})) \end{aligned}$$

and

$$\begin{aligned} h_2 &= \prod_{x_j \in S_{i,1}} P(x_j \leftarrow \pi(x_j) | x_{i+1} \leftarrow p_2) \prod_{x_j \in S_{i,2}} P(x_j \leftarrow \pi(x_j) | (x_{i+1} \leftarrow p_2) \\ &= \prod_{x_j \in S_{i,1}} (1 \Leftrightarrow L(x_j : x_{i+1})) \prod_{x_j \in S_{i,2}} L(x_j : x_{i+1}) \end{aligned}$$

Thus, the independence assumption suggests that a joint linkage probability such as $L(x_{i+1} : x_1, x_2, \ldots x_i)$ can be estimated using the pairwise linkage probabilities $L(x_{i+1} : x_1)$, $L(x_{i+1} : x_2), \ldots, L(x_{i+1} : x_i)$. The amount of space taken up by these pairwise linkage probabilities is $O(\text{number of genes per chromosome})^2$, which is reasonable for most problems. For many problems such as those amenable to a hierarchical decomposition, efficient sparse matrix representations can be used to reduce space requirements considerably. Problem-specific information in some problems can also be stated in terms of pairwise linkage probabilities, a local property that examines two components at a time.

The method described above employs pairwise independence to approximate the general linkage probabilities. The LinX crossover operator that uses this methodology is described in Figure 2.

LinX Crossover:

- Copy all common genes from the parents to the offspring. For the remaining genes of the offspring, use the following steps.
- For two randomly chosen genes, select one allele from parent p_1 and the other from p_2.
- Determine the remaining (unallocated) offspring genes iteratively as follows:
  - Let S_1 be the set of genes inherited so far from parent p_1.
  - Let S_2 be the set of genes inherited so far from parent p_2.
  - Randomly select x_{i+1}, a gene whose value in the offspring has not yet been determined. Calculate

$$h_1 = \prod_{x_j \in S_1} L(x_j : x_{i+1}) \prod_{x_j \in S_2} (1 \Leftrightarrow L(x_j : x_{i+1}))$$

    and

$$h_2 = \prod_{x_j \in S_1} (1 \Leftrightarrow L(x_j : x_{i+1})) \prod_{x_j \in S_2} L(x_j : x_{i+1}).$$

  - Randomly generate a number $r \in [0, 1]$ from the uniform distribution and assign

$$o[x_{i+1}] = \begin{cases} p_1[x_{i+1}] & \text{if } r \leq \frac{h_1}{h_1+h_2} \\ p_2[x_{i+1}] & \text{otherwise.} \end{cases}$$

- Adjust the generated offspring to satisfy feasibility conditions, if needed.

Figure 2: A high-level description of LinX crossover.

5 Linkage Adaptation

Few problems are understood well enough that the precise linkage probabilities are known a priori. Indeed, the main reason for "tinkering" with several operators is ignorance of the relationships between different genes. In such cases, the hardest problem is that of learning the linkage probabilities on the fly during the application of the evolutionary algorithm to the problem. This is difficult because of the presence of considerable noise (a nonexistent relationship appears to exist because of false support from the initial population). The following are adaptation procedures are proposed in this paper.

Hebbian Linkage Adaptation (HLA)

The neural networks literature provides one useful paradigm for linkage adaptation: Hebb's rule states that the simultaneous (synchronous) excitation of two neurons results in a strengthening of the connections between them, while asynchronous activation for two neurons will result in a weakening of the connections. Linkage probabilities are analogous to "connection strengths" (weights attached to edges between nodes) in neural networks. This idea is exploited in the HLA algorithm, adapting linkage values during the execution of the GA using the Hebbian linkage adaptation procedure described in Figure 3. The HLA

Hebbian Linkage Adaptation (HLA) Procedure:

Initial step: Initialize each entry $\mathrm{L}[j,k] \in [0.5, 1]$ (and $X[j,k] \in [\Leftrightarrow 1, 1]$) randomly.
Reproduction step: For each offspring generated in this iteration, and for all $j \neq k$:

- Let $\bar{f}$ be the average fitness of the current population.
- Let O be an offspring of p_1 and p_2 produced in the current generation using LinX.
- Let $p_1[j]$ denote the jth allele of p_1 and $p_2[k]$ the kth allele of p_2.
- For each j,k, such that parents p_1 and p_2 differ in the jth and kth positions and both alleles of the offspring come from the same parent, (i.e., $p_1[j] \neq p_2[j], p_1[k] \neq p_2[k]$ and $O[j] = p_i[j]$ and $O[k] = p_i[k]$ for $p_i \in \{p_1, p_2\}$), adapt the linkage matrix as follows:

$$\begin{aligned} \triangle X[j,k] &= \eta \times (\text{fitness}(O) \Leftrightarrow \bar{f}) \\ \mathrm{L}[j,k] &= max(0.5, \frac{X[j,k] \Leftrightarrow \min_\ell(X[j,\ell])}{\max_\ell(X[j,\ell]) \Leftrightarrow \min_\ell(X[j,\ell])}) \end{aligned}$$

Figure 3: Hebbian linkage adaptation procedure.

algorithm uses the fitness of the offspring resulting from a crossover to judge the efficacy of the linkage probabilities used for that crossover step.

"Follow The Fittest" Linkage Adaptation (FTF)

Another method for adaptation uses an idea extracted from the Dynamic Knowledge-based Nonuniform Crossover (DKNUX) operator of Maini et al. (1994b). DKNUX uses the alleles of the current best chromosome (in every generation of the GA) to bias the probabilities used to determine offspring alleles. Using this "Follow The Fittest" (FTF) model, we use such information to adapt the linkage probabilities between different genes in the chromosome. However, our approach is based on linkages between different genes. Figure 4 describes the FTF adaptation procedure that uses this idea.

6 Problems

Two classes of problems were used to test our approach. The first class consists of problems in which the linkage structure is known a priori. In addition to comparing LinX to other operators, this class allows us to test whether linkage adaptation procedures succeed in learning the right linkage values when started from a randomly generated linkage matrix. The second class consists of real-world problems where linkage information is unknown or partially known.

We have experimented with GA-deceptive problems, where most of the competitive schemata are much different from the optimal one, misleading the GA towards the wrong attractors (nonglobal optima). Sections 6.1 – 6.3 define three GA-deceptive problems: the *order-3 deceptive problem* (Goldberg et al., 1989), *order-6 bipolar multi-modal problem* (Goldberg et al., 1992), and *order-5 Mühlenbein's problem* (Mühlenbein et al., 1998).

"Follow The Fittest" Adaptation (FTF) Procedure:

Initial step: Initialize each entry $L[j,k] \in [0.5, 1]$ (and $X[j,k] \in [\Leftrightarrow 1, 1]$) randomly.
Reproduction step:

- Let $Best$ be the current best chromosome of the current generation;
- For each pair of genes (j,k):

$$X[j,k] = X[j,k] + f(Best_{j,k}),$$

  where $Best_{j,k}$ is the individual obtained by perturbing the j^{th} and the k^{th} genes of $Best$.

- For each pair of genes (j,k):

$$L[j,k] \quad = \quad max(0.5, \frac{X[j,k] \Leftrightarrow \min_\ell(X[j,\ell])}{\max_\ell(X[j,\ell]) \Leftrightarrow \min_\ell(X[j,\ell])})$$

Figure 4: "Follow The Fittest" linkage adaptation procedure.

6.1 Goldberg's Order-Three Problem

Goldberg et al. (1989) defined an n-bit concatenation of an order-three deceptive problem that is difficult for standard crossover operators. Two versions of this problem, "Easy" and "Hard" are defined below. The Easy order-3 problem is an n-bit function, where strongly related bits (of each 3-bit subfunction) reside adjacent to each other. In the Hard 30-bit order-3 problem, the three bits of each subfunction are located far away from each other, separated by other bits. Bits of each 3-tuple are located at $i, i+10$, and $i+20$ for $i = 1, 2, \ldots, 10$. The order-3 deceptive problems considered in this paper are defined below; minor variations of the function led to similar results.

Assuming binary coding,

$$f(x) = \sum_{i=1}^{\frac{n}{3}} f_3(s_i),$$

where each s_i is a disjoint 3-bit substring of x, and

$$f_3(s_i) = \begin{cases} 0.9 & if \quad |s_i| = 0 \\ 0.6 & if \quad |s_i| = 1 \\ 0.3 & if \quad |s_i| = 2 \\ 1.0 & if \quad |s_i| = 3 \end{cases}$$

where $|s_i|$ denotes the sum of the bits in the 3-bit substring s_i.

In Easy order-3 problems, bits of the same subfunction are adjacent; an example of a 9-bit Easy order-3 problem illustrates this concept:

$Position$	:	123	456	789	$Fitness$
$Alleles$	:	011	001	101	$f(x) = f_3(011) + f_3(001) + f_3(101) = 1.2$
$substring$	:	s_1	s_2	s_3	

In Hard order-3 problems, the bits are maximally distributed across the individual; e.g., in the 9-bit Hard order-3 problem, the substrings are distributed as shown below:

$Position$	:	123	456	789		$Position$	:	147	258	369	$Fitness$
$Alleles$	:	011	001	101	$\Leftrightarrow$	$Alleles$	:	001	100	111	$f(x) =$
						$substring$	:	s_1	s_2	s_3	$\sum_{i=1}^{3} f_3(s_i) = 2.2$

A standard genetic algorithm (using 1PTX or 2PTX) works better on the Easy order-3 problem than on the Hard version because of the implicit availability of linkage information in the representation in which tightly linked genes are in close proximity to each other. For UX, both problems are equally difficult because UX ignores gene adjacency. It has been shown (Goldberg et al., 1989) that a simple genetic algorithm using any of these crossover operators rarely converges to the global optima for the Hard versions of the problem.

6.2 Bipolar Deceptive Problem

For this deceptive problem, the solution string is constructed by concatenation of many *order-6 bipolar functions* (i.e., discrete functions with two global optima for complementary bit-strings). Points near the global optima are the lowest in fitness, and the function tends to take larger values as we move towards bit-strings with as many zeroes as ones, where local optima are located. Concatenation of five such 6-bit functions yields a problem with 5 million suboptima and 32 global optima (Goldberg et al., 1992), making this problem challenging. By varying the coupling between each function's bits, we can instantiate different versions of this problem. An "Easy order-6 bipolar" problem is defined as having tightly coupled bits close together; i.e., the distance between the first and last bits of the 6 bits is exactly 5; a "Hard order-6 bipolar" problem is such that the six bits of each constituent function are physically separated by greater distances.

More rigorously, consider a string x of size n, where n is a multiple of 6. Consider a substring (s_i) of size 6 and define a function as follows:

$$f_6(s_i) = \begin{cases} 1.0 & if \quad |s_i| = 0 \\ 0.0 & if \quad |s_i| = 1 \\ 0.4 & if \quad |s_i| = 2 \\ 0.8 & if \quad |s_i| = 3 \\ 0.4 & if \quad |s_i| = 4 \\ 0.0 & if \quad |s_i| = 5 \\ 1.0 & if \quad |s_i| = 6 \end{cases}$$

where $|s_i|$ equals the number of ones in the substring; then, the fitness over the string x of size n is defined as $f(x) = \sum_{i=1}^{\frac{n}{6}} f_6(s_i)$. In an Easy version of this problem, the substrings are concatenated next to each other, i.e., $x = s_{1,1}\ s_{1,2} \ldots s_{1,6}\ s_{2,1}\ s_{2,2} \ldots s_{2,6} \ldots s_{\frac{n}{6},6}$. In the Hard version, they are mingled so that two bits of s_1 are separated by distances that are multiples of $\frac{n}{6}$, i.e., $x = s_{1,1}\ s_{2,1} \ldots s_{\frac{n}{6},1}\ s_{1,2}\ s_{2,2} \ldots s_{\frac{n}{6},2} \ldots s_{1,6} \ldots s_{\frac{n}{6},6}$.

6.3 Mühlenbein's Order-5 Problem

Mühlenbein et al. (1998) define an order-5 decomposable problem as follows:
Let x be a chromosome obtained by concatenating substrings $s_1, \ldots, s_{\frac{n}{5}}$, where, for each 5-bit substring s_i,

$$f_5(s_i) = \begin{cases} 4.0 & if \quad s_i = 00000 \\ 3.0 & if \quad s_i = 00001 \\ 2.0 & if \quad s_i = 00011 \\ 1.0 & if \quad s_i = 00111 \\ 3.5 & if \quad s_i = 11111 \\ 0.0 & otherwise. \end{cases}$$

Each order-5 subfunction $f_5(s_i)$ has one local and one global optima ($s_i = 00000$). The fitness of chromosome x is measured as $f(x) = \sum_{i=1}^{\frac{n}{5}} f_5(s_i)$. The search space for this problem contains many local optima, many zero-fitness individuals, and only one global optimum. A small perturbation to a solution may drop it from a high hill into a deep valley. This problem is therefore considered challenging for genetic algorithms.

6.4 Graph Bipartitioning

Many scientific problems are best solved by parallel algorithms. This is often done by partitioning data among processors in a way that minimizes communication cost while maintaining roughly equal computational loads on all processors. The problem of satisfying these constraints is called *graph partitioning*. Many problems can be mapped into graph partitioning problems. Limiting the number of processors to two reduces the problem into a *bipartitioning* problem. Graph bipartitioning is a well-known NP-hard problem. Many researchers have applied GAs to the graph bipartitioning problem (e.g., Maini et al. (1994b)).

In our experiments, a fully connected graph is used, where nodes represent computational loads and edges represent communication costs. We assume that each node represents equal computational load. Communication costs are represented by edge weights drawn from a uniform random distribution in the range [0,1]. Nodes are to be assigned bin labels $\in \{0, 1\}$, used as components of a bit string that describes a candidate solution to the bipartitioning problem. In a feasible solution, each bin must have equally many nodes, i.e., each chromosome in the GA must contain equally many 0s and 1s. Infeasible candidate solutions are randomly patched up when they are generated, to restrict search to feasible solutions. The cost function to be minimized is the sum of the weights of edges between the two bins.

7 Experimental Results

In this section, we address the following questions with respect to the performance of LinX crossover and the linkage adaptation procedures:

- If there are definite known linkages between genes, will LinX outperform other general purpose crossover operators?
- Will the linkage adaptation algorithms succeed in learning the right linkages between genes and perform competitively?

The genetic algorithm used in our experiments has the following characteristics:

- Chromosomes were randomly initialized.
- Roulette wheel selection methodology was used to select parents for mating.

Table 1: A comparison between the HLA and FTF algorithms for the Easy order-3 problems.

Problem size	Pop. size	Max. Gens.	Optimum fitness	Fitness		Func. Eval. Required	
				HLA	FTF	HLA	FTF
21	40	300	7.0	6.94	6.977	16,543	7,475
30	60	600	10.0	9.893	9.947	57,261	27,077
45	90	1000	15.0	14.713	14.873	160,201	89,985

- A repair mechanism was applied to infeasible solutions in the population to make them satisfy the problem constraints (e.g., in graph bipartitioning problems, there must be equally many alleles of each kind).
- Mutation rate was $1/N$, where N is the number of genes in an individual.
- The crossover rate was set to 1.0.
- An elitism mechanism was used with each new generation consisting of the best 10% of the current generation, and the rest came from newly generated offspring. Our experiments with different elitism ratios showed that equally good results were obtained when 5% - 25% of the best (among the current generation) were retained for the next generation.
- In the HLA algorithm, the learning rate (η) for the Hebbian learning was set to 1.

Our current goal is not to find the best values of parameters to solve optimization problems but to compare the effects of different crossover operators on the quality of solutions produced by the GA. Therefore, during the course of our experiments, all other parameters are kept the same except crossover. Experiments with other variations of the GA are described in Section 7.8; more details can be found in Salman (1999).

7.1 Comparing Linkage Adaptation Algorithms

Both the HLA and FTF algorithms succeeded in learning linkage matrices successfully used to solve various problems. In all the experiments used to compare them, the best solutions obtained using the FTF algorithm were better or similar to those obtained using the HLA algorithm.

Tables 1 and 2 depict two sets of results comparing the HLA and FTF algorithms. Similar results were obtained for other problems. For the rest of this paper, all the Adaptive Linkage crossover results were obtained using the FTF algorithm, henceforth referred to as "ALinX". More detailed results with the HLA algorithm are given in Salman et al. (1998).

7.2 Performance on Deceptive Problems

The performance of LinX was superior to that of all other operators, as shown in Tables 3, 4, and 5. The variance of the best solutions found by LinX was either zero or very small, indicating that it reached either the optimal or a near-optimal solution in every trial. LinX required fewer generations than the other operators to reach the final result.

Table 2: A comparison between the HLA and FTF algorithms for graph bipartitioning problems, minimizing the sum of weights of edges between nodes in different bins.

Problem size	Pop. size	Max. Gens.	Communication Cost		Func. Eval. Required	
			HLA	FTF	HLA	FTF
30	50	300	94.318	94.284	42,000	33,270
75	90	4000	634.4	632.9	1,120,000	853,610
100	200	5000	1153	1152	2,800,000	2,073,250

Among the remaining operators (ALinX, 2PTX, 1PTX, and UX), ALinX was the best. Its performance was slightly better than the others for the Easy versions of the problem: in general, all operators exhibited similar behavior for the Easy versions of the problem. For the Hard versions, ALinX showed a clear advantage over 1PTX, 2PTX, and UX. In almost all experiments, ALinX was statistically superior to the traditional operators, as apparent from the *t-test* results (shown in the Tables 3, 4, and 5) that compare ALinX to its nearest competitor. Where no statistical superiority exists, ALinX's results were more skewed than the others, which indicates that it pushes the solution toward the optimal solution more successfully than the others. Further, ALinX was able to learn the underlying linkage structure of these problems to a significant extent. It distinguishes clearly those genes among which there are strong dependencies. In Tables 3, 4, and 5, numbers in parentheses next to the ALinX label indicate the average linkage probabilities of strongly and weakly linked components, respectively (discussed in Section 7.6).

Table 6 depicts the results for the order-3 problems, when the termination criteria were modified, allowing computations to continue until the optimal solution was found. LinX and ALinX show clear superiority in terms of number of function evaluations required to reach the global optima. LinX and ALinX behave similarly regardless of the structure of the linkage; for both Easy and Hard versions, they required roughly the same number of function evaluations, whereas results with other operators vary with the distribution of genes of each 3-bit subfunction. An interesting observation is that 2PTX crossover was superior to UX crossover for the Easy versions of the problem (where the linkage information was embedded implicitly into the representation, such that genes of each building block are adjacent to each other), but 2PTX was outperformed by UX for the Hard versions.

7.3 Population Distribution

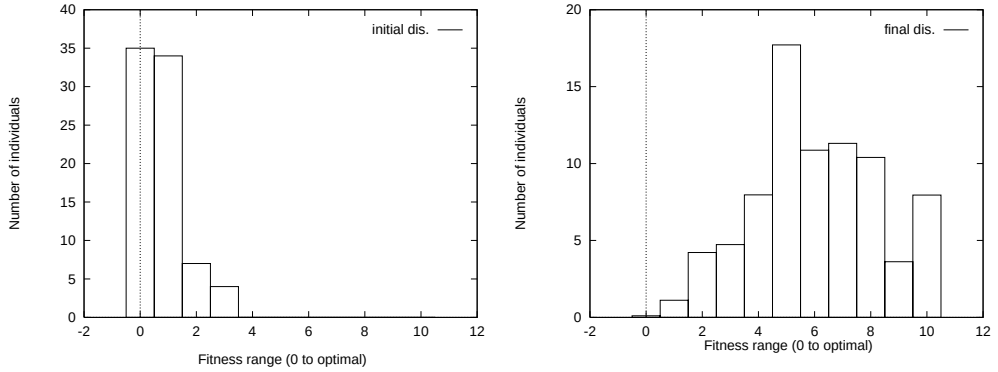
In addition to the above summary statistics, we observed the initial populations and the populations obtained at the end of a number of generations (the terminal population of chromosomes). For Mühlenbein's order-5 problem, this analysis uncovered interesting behavior. Table 7 summarizes the results for ALinX, 2PTX, 1PTX, and UX. Operators were compared according to the best solution found, number of trials for which an optimal solution was found, population average, population variance, and population skewness. We observed the initial populations and populations after 10, 100, and 2000 iterations (the terminal populations of chromosomes). These results were averaged over 60 trials. Clearly, ALinX maintains superiority over 2PTX, 1PTX, and UX. It was better at obtaining best solutions (largest average of best solutions and largest number of global optima found) and better at shifting the whole population towards better areas of search space (best

Table 3: Results for *order-3* problems averaged over 30 trials; *t-test* was performed between ALinX and the next best crossover (other than LinX); fitness of the optimal solution = (number of bits)/3; †: solutions obtained are all optimal.

Operator Type	Pop. Best Fitness	No. of trials in which an optimal was found (max. 30)	# Gen. Required	Variance of the Best Solution	Skewness of the Best Solution
Bits=21, Pop. size=30, Max. Gen.=500, Easy Version					
LinX	6.997	29	182.400	0.000	-4.942
Uniform	6.967	21	390.367	0.003	-1.270
One-Point	6.983	25	276.733	0.001	-1.700
Two-Point	6.950	24	288.133	0.034	-4.623
ALinX(0.975,0.510)	6.983	26	251.800	0.002	-2.644
Bits=21, Pop. size=30, Max. Gen.=500, Hard Version, *t-value*=2.618 with $P \leq 0.006$					
LinX	7.000	30	176.800	0.000	†
Uniform	6.953	20	361.433	0.006	-1.604
One-Point	6.890	9	437.567	0.018	-2.764
Two-Point	6.783	6	446.433	0.156	-4.214
ALinX(0.984,0.510)	6.993	28	254.300	0.001	-3.302
Bits=30, Pop. size=50, Max. Gen.=1000, Easy Version					
LinX	9.997	29	343.400	0.000	-4.942
Uniform	9.920	13	868.033	0.009	-1.513
One-Point	9.973	23	690.633	0.003	-1.684
Two-Point	9.983	25	604.200	0.001	-1.700
ALinX(0.980,0.509)	9.983	25	527.300	0.001	-1.700
Bits=30, Pop. size=50, Max. Gen.=1000, Hard Version, *t-value*=4.49 with $P \leq 0.0005$					
LinX	9.987	29	399.467	0.005	-4.942
Uniform	9.880	11	873.900	0.017	-1.107
One-Point	9.837	3	997.233	0.012	-1.031
Two-Point	9.843	9	910.967	0.032	-1.747
ALinX(0.980,0.512)	9.990	27	602.867	0.001	-2.534
Bits=60, Pop. size=80, Max. Gen.=4000, Easy Version					
LinX	19.993	28	1911.333	0.001	-3.302
Uniform	19.603	0	4000.000	0.038	-0.238
One-Point	19.793	4	3928.800	0.026	-1.172
Two-Point	19.950	19	3033.567	0.006	-1.493
ALinX(0.966,0.523)	19.820	9	3610.400	0.038	-0.879
Bits=60, Pop. size=80, Max. Gen.=4000, Hard Version; *t-value*=6.149 with $P \leq 0.0005$					
LinX	19.990	27	1969.467	0.001	-2.534
Uniform	19.573	1	3991.167	0.039	0.203
One-Point	19.240	0	4000.000	0.062	0.167
Two-Point	19.350	0	4000.000	0.045	0.124
ALinX(0.970,0.522)	19.857	10	3575.100	0.025	-1.435

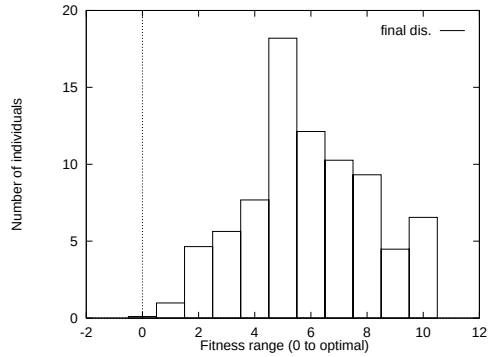
population average, less variance, and more skewed). All populations were shifted from positively skewed fitness distribution in the beginning to near-symmetrical distributions at the terminal stage. Compared to the others, the ALinX distribution was the most symmetrical (near zero skewness).

Figure 5 depicts the initial population fitness distribution and the final evolved population fitness distribution for each operator after 4000 generations. Obviously, the initial randomly selected population does not contain any optimal solution. The LinX operator leads to the maximum number of optimal solutions, followed by ALinX. The remaining three lead to approximately equal numbers of optimal solutions, fewer than observed in ALinX. Both LinX and ALinX lead to fewer local optima near the global optimum than the other operators as shown by the dip in the next-to-last fitness range in Figure 5.

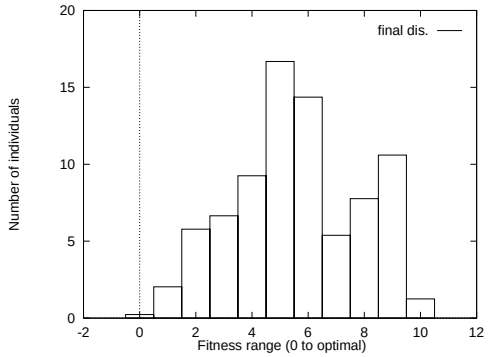


Initial: ALL

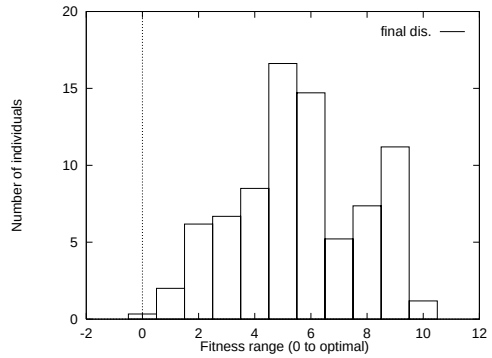
Final: LinX



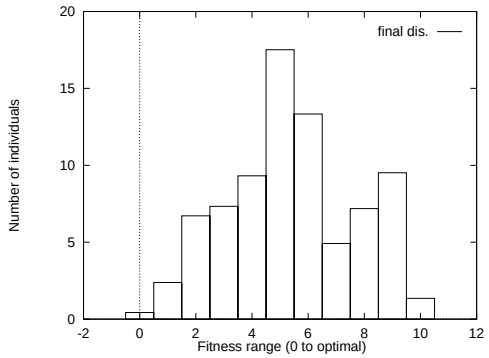
Final: ALinX



Final: 2PTX



Final: 1PTX



Final: UX

Figure 5: Fitness histogram for *Mühlenbein's* order-5 problem, 35 bits, population size = 80, maximum number of generations = 4000.

Table 4: Results for *bipolar* problems averaged over 30 trials; *t-test* was performed between ALinX and the next best crossover (other than LinX); optimal solution fitness = (number of bits)/6; †: solutions obtained are all optimal.

Operator Type	Pop. Best Fitness	No. of trials in which an optimum was found (max 30)	# Gen. Required	Variance of the Best Solution	Skewness of the Best Solution
Bits=30, Pop. size=50, Max. Gen.=500, Easy Version					
LinX	5.000	30	138.233	0.000	†
Uniform	4.967	25	320.633	0.006	-1.700
One-Point	4.973	26	241.333	0.005	-2.050
Two-Point	4.993	29	197.567	0.001	-4.942
ALinX(0.904,0.500)	4.993	29	185.34	0.001	-4.942
Bits=30, Pop. size=50, Max. Gen.=500, Hard Version; *t-value*=2.57 with $P \leq 0.025$					
LinX	4.993	29	157.533	0.001	-4.942
Uniform	4.953	24	346.000	0.010	-1.943
One-Point	4.847	11	420.067	0.018	-0.292
Two-Point	4.893	15	403.367	0.013	-0.411
ALinX(0.904,0.500)	5.00	30	178.231	0.0	†
Bits=60, Pop. size=80, Max. Gen.=2000, Easy Version; *t-value*=1.644					
LinX	10.000	30	744.833	0.000	†
Uniform	9.580	1	1972.200	0.062	-0.550
One-Point	9.780	8	1832.267	0.029	-0.153
Two-Point	9.900	18	1652.967	0.021	-1.535
ALinX(0.902,0.500)	10.00	30	1037.402	000	†
Bits=60, Pop. size=80, Max. Gen.=2000, Hard Version; *t-value*=8.23 with $P \leq 0.0005$					
LinX	10.000	30	705.400	0.000	†
Uniform	9.587	4	1938.233	0.072	-0.218
One-Point	9.320	0	2000.000	0.062	-0.271
Two-Point	9.540	2	1967.833	0.111	-0.708
ALinX(0.900,0.500)	9.993	29	1256.981	0.001	-4.94
Bits=90, Pop. size=200, Max. Gen.=10000, Easy Version					
LinX	15.000	30	3743.633	0.000	†
Uniform	13.920	0	10000.000	0.291	-0.483
One-Point	14.787	8	9563.134	0.036	-1.061
Two-Point	14.947	24	6262.367	0.016	-2.794
ALinX(0.898,0.500)	14.927	24	7185.231	0.04	-3.46
Bits=90, Pop. size=200, Max. Gen.=10000, Hard Version; *t-value*=8.78 with $P \leq 0.0005$					
LinX	14.927	29	3742.033	0.161	-4.942
Uniform	13.860	0	10000.000	0.271	-0.557
One-Point	13.847	1	9939.066	0.258	-0.533
Two-Point	14.167	1	9869.566	0.240	-1.091
ALinX(0.897,0.500)	14.967	26	6483.360	0.009	-2.644

7.4 Graph Bipartitioning: Results

In this example, the linkage behavior is not predetermined but must be learned. Several data sets were considered. For ALinX, the linkage matrix was initialized randomly, whereas for LinX the linkage matrix was constructed using the following heuristic:

> The larger the cost of communication between two nodes, the greater the need to put them into the same bin.

To minimize the cost of communication across the two bins, we attempt to maximize the sum of edge costs between nodes belonging to the same bin. We hypothesized that a linkage matrix reflecting this property would lead the algorithm in the right direction. To implement this heuristic, the linkage matrix entries (for LinX) were made proportional

Table 5: Results for *Mühlenbein's* order-5 problems averaged over 30 trials; *t-test* was done between ALinX and the next best crossover; optimal solution fitness = 4×(number of bits/5).

Operator Type	Pop. Best Fitness	No. of trials in which an optimum was found (max 30)	# Gen. Required	Variance of the Best Solution	Skewness of the Best Solution
Bits=20, Pop. size=30, Max. Gen.=500, Easy Version					
LinX	15.700	17	280.600	0.148	-0.771
Uniform	15.500	8	375.000	0.138	0.000
One-Point	15.683	17	276.700	0.181	-1.060
Two-Point	15.667	13	325.767	0.109	-0.436
ALinX(851,0.543)	15.683	16	305.06	0.164	-1.093
Bits=20, Pop. size=30, Max. Gen.=500, Hard Version; *t-value*=0.707					
LinX	15.783	18	263.000	0.081	-0.796
Uniform	15.567	13	330.833	0.202	-0.525
One-Point	15.383	7	413.600	0.219	-0.280
Two-Point	15.450	8	373.800	0.213	-0.575
ALinX(0.865,0.562)	15.650	18	343.4	0.209	-0.594
Bits=35, Pop. size=60, Max. Gen.=4000, Easy Version; *t-value*=1.6 with $P \leq 0.1$					
LinX	27.800	19	2176.533	0.079	-0.940
Uniform	27.433	10	2688.467	0.306	-0.782
One-Point	27.500	9	3379.900	0.190	-0.605
Two-Point	27.667	15	2384.567	0.144	-0.595
ALinX(0.846,0.551)	27.800	18	2378.398	0.062	-0.388
Bits=35, Pop. size=60, Max. Gen.=4000, Hard Version; *t-value*=2.6 with $P \leq 0.002$					
LinX	27.667	15	2581.867	0.144	-0.595
Uniform	27.300	8	2997.100	0.355	-0.761
One-Point	27.100	4	3596.767	0.283	-0.053
Two-Point	27.100	2	3736.433	0.283	-0.885
ALinX(0.860,0.546)	27.650	15	2853.178	0.158	-0.551
Bits=45, Pop. size=70, Max. Gen.=5000, Easy Version; *t-value*=2.56 with $P \leq 0.002$					
LinX	35.633	14	3486.533	0.154	-0.473
Uniform	35.000	3	4508.933	0.379	-0.214
One-Point	34.983	2	4842.800	0.491	-0.815
Two-Point	35.083	2	4976.200	0.208	-0.054
ALinX(0.849,0.550)	35.383	7	4135.133	0.201	-0.104
Bits=45, Pop. size=70, Max. Gen.=5000, Hard Version; *t-value*=1.5 with $P \leq 0.1$					
LinX	35.617	14	3601.600	0.167	-0.424
Uniform	35.067	4	4344.067	0.375	-0.244
One-Point	34.450	1	4835.433	1.661	-3.186
Two-Point	34.950	1	4884.000	0.230	-0.036
ALinX(0.857,0.547)	35.283	6	4562.2	0.253	-0.274

to the communication costs between the nodes; the higher the communication cost, the higher the linkage, and *vice versa*. Table 8 shows the results for each operator on five graphs containing 25, 35, 45, 60, and 80 nodes with random edge weights between 0 and 1, respectively. Results were obtained by taking averages of the best solutions over ten trials. The relative performance of LinX and ALinX (compared to 1PTX, 2PTX, and UX) appears to improve with problem size.

For the graph-bipartitioning problem, it is difficult to predetermine the right linkage matrix from the graph. Hence it was possible for ALinX to learn a linkage matrix that outperforms the nonadaptive LinX operator that uses a linkage matrix based on a reasonable heuristic. Table 8 shows that ALinX succeeds in learning the appropriate linkage matrix and outperforms other operators, while LinX follows in the second place.

Table 6: The number of function evaluations required by the GA to find the optimal solution for order-3 problems and (in parentheses) the number of trials (≤ 10) in which the global optimum was reached.

# Bits	Pop. Size	Max. # Gen.	Average no. of Function Evaluations $\times 10^4$ (# trials in which an optimum was found) LinX	ALinX	2PTX	1PTX	UX
Easy Order-3							
21	40	1000	0.46 (10)	0.94 (10)	1.10 (10)	0.60 (10)	1.10 (10)
30	60	5000	1.96 (10)	2.48 (10)	2.98 (10)	3.49 (10)	6.57 (10)
45	90	8000	4.96 (10)	22.87 (10)	13.16 (10)	32.67 (10)	38.39 (9)
60	120	10000	13.13 (10)	48.81 (10)	26.48 (10)	63.70 (9)	94.21 (6)
Hard Order-3							
21	40	1000	0.45 (10)	0.78 (10)	2.61 (6)	1.26 (9)	1.09 (10)
30	60	5000	1.59 (10)	2.48 (10)	9.92 (10)	8.21 (10)	4.60 (10)
45	90	8000	7.47 (10)	19.56 (10)	55.71 (5)	61.18 (5)	32.28 (10)
60	120	10000	15.94 (10)	54.53 (10)	108.00 (0)	107.07 (1)	90.81 (5)

Table 7: Results for *Mühlenbein's* order-5 problem (30 bits, population size = 50, Hard version, 60 trials): illustrating the distribution changes in the problem after 10, 100, and 2000 generations, optimal solution fitness = 24.

	Start	ALinX			Uniform		
Generation #	0	10	100	2000	10	100	2000
Best Fitness	10.18	16.77	23.31	23.65	16.49	23.29	23.32
# of successful trials (max. 60)	0	0	10	33	0	11	13
Pop. Average	2.58	7.09	13.32	13.64	6.59	13.15	12.93
Pop. Variance	7.16	19.75	31.86	32.45	18.83	33.32	33.15
Pop. Skewness	0.9	0.41	0.13	0.06	0.45	0.17	0.25
	Start	1PTX			2PTX		
Generation #	0	10	100	2000	10	100	2000
Best Fitness	10.18	15.90	23.19	23.28	16.06	23.12	23.28
# of successful trials (max. 60)	0	0	6	14	0	7	11
Pop. Average	2.58	7.11	13.32	13.21	6.81	13.19	13.33
Pop. Variance	7.16	19.37	35.81	36.06	19.79	35.20	35.21
Pop. Skewness	0.9	0.28	0.10	0.14	0.39	0.11	0.11

7.5 Convergence

Using problem-specific knowledge may, in some cases, lead to rapid convergence to local optima to the detriment of global search. However, we found that LinX and ALinX were surprisingly robust in this respect, as long as simple mutation was also used. Indeed, performance of the algorithm did not vary when an artificial method to overcome premature convergence (random reinitialization) was introduced into the computer program. LinX overcomes the danger of premature convergence usually associated with algorithms using problem-specific knowledge, because it uses this knowledge in a very loose way (only a global linkage matrix is used to direct crossover probabilistically).

Convergence of the linkage adaptation process may also be examined: will elements of the linkage matrix stabilize, remaining almost constant in later iterations of the GA? Figure

Table 8: Graph Bipartitioning Problem, average over 30 trials.

# Nodes	Pop. size	# Gen.	Sum of weights of edges across bins			
			ALinX	LinX	2PTX	UX
25	50	500	64.208	64.508	65.238	65.618
30	70	1000	131.180	130.091	133.937	134.724
45	90	1500	222.858	224.124	230.339	229.706
60	120	2000	402.403	405.897	418.06	420.584
80	160	3000	722.805	725.033	766.335	751.617

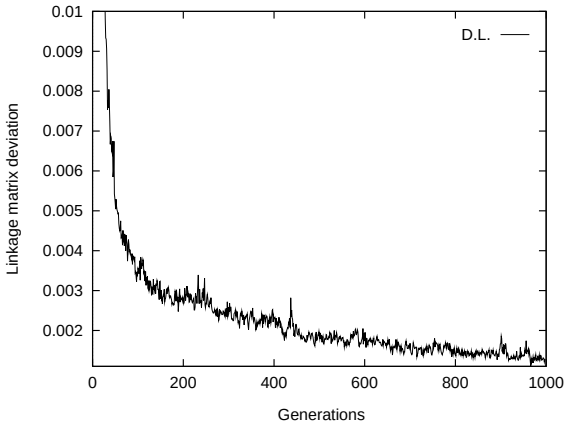


Figure 6: D_L (variation in linkage values) for ALinX for a 60-bit Goldberg's order-3 problem, using population size 70.

6 answers this question in the affirmative using the deviation measure

$$D_L = \frac{1}{N}\sqrt{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(\text{NewL}[i][j] \Leftrightarrow \text{OldL}[i][j])^2}$$

to compare the linkage matrix entries before and after each generation. Similar convergence was also obtained with the HLA adaptation algorithm.

7.6 Interpretability

In this section, we address two questions regarding the interpretability of linkage matrix values:

1. What happens if the linkage probabilities used by LinX are perturbed from the initial optimal setting? Will it really affect the performance of the GA if the optimal probabilities are not chosen? We incrementally perturbed linkage values of one cluster of strongly linked genes, fixing every other value to the ideal case. We observed the best fitness changes over these perturbations. For example, in a 30-bit Goldberg's order-3 problem, we initialized the linkage matrix in LinX to the ideal values; then we perturbed linkage values between the first three genes, which constitute an optimal

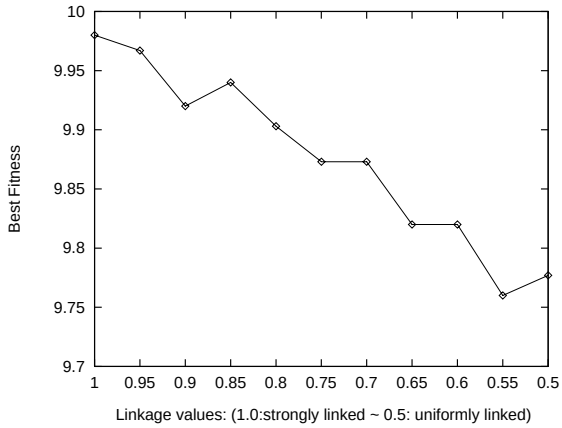


Figure 7: Variation of LinX performance when strong linkage matrix values (=1) are perturbed, for Goldberg's order-3 problem with 30 bits, 50 population size, maximum of 500 generations allowed, and 30 trials.

Table 9: Average linkage values for strongly linked and weakly linked genes after adaptation using ALinX for the Hard order-3 problems (initialized randomly).

# bits	Pop. size	Max # Gens.	Average strongly linked	Average weakly linked
21	30	500	0.984	0.510
30	50	1000	0.980	0.512
60	80	4000	0.970	0.522

building block. Figure 7 shows the effect of these perturbations on the performance of LinX; the performance degrades as we go farther from the ideal values. Simultaneously perturbing linkage values for genes in multiple building blocks yielded worse results. This implies that choosing the right linkage values is important for obtaining good results.

2. Does the adaptive algorithm result in a linkage matrix whose elements are reasonable (compared to the optimal setting) and easy to interpret? To answer this question, we conducted experiments with Goldberg's order-3 deceptive problems for which optimal linkage information is known a priori; the three bits representing each subproblem are tightly linked to each other (linkage probability = 1.0), but not linked to the others (linkage probability = 0.5). Table 9 shows that ALinX is successful in adapting an initially random matrix to obtain the desired linkage values for Hard order-3 problems; similar results were obtained with Easy order-3 and the bipolar problems.

7.7 Degree of Deception

Another set of experiments was conducted to examine the effect of the degree of deception (deceptive to optimal ratio) on the performance of crossover operators. We have tuned the order-3 problem to accommodate different degrees of deception as shown in Table 10. In every case, LinX and ALinX showed superior performance in terms of the number of

Table 10: Order-3 problems with various degrees of deception.

Sub-Strings	Sub-fitness	Sub-String	Sub-fitness
Highly Deceptive		Medium Deceptive	
000	0.95	000	0.9
001 010 100	0.9	001 010 100	0.6
011 101 110	0.05	011 101 110	0.3
111	1.0	111	1.0
Marginally Deceptive		Not Deceptive	
000	0.75	000	0.75
001 010 100	0.6	001 010 100	0.4
011 101 110	0.5	011 101 110	0.5
111	1.0	111	1.0
Unimodal			
000	0.0		
001 010 100	1.0		
011 101 110	2.0		
111	3.0		

optima obtained and the number of generations required to reach the optima, as shown in Figure 8. The performance of LinX was almost constant for different degrees of deception, whereas other operators yielded better results for lower degrees of deception. For unimodal problems (such as the counting ones problem), all operators have similar performance.

7.8 Varying the Parameters

Are the apparent good results obtained by LinX and ALinX the consequence of an inherent bias in experimental parameters? To answer this question, we varied parameter values to explore the soundness of conclusions from the experiments reported above. Three sets of experiments were conducted:

1. The performance of our algorithms was compared to others while varying the mutation rate. This is crucial to eliminate the doubt that the new algorithms work only under a specific mutation rate and also serves the purpose of identifying mutation rates for which LinX and ALinX work best. Figure 9 shows that all the crossover operators vary similarly with mutation. It also shows that LinX and ALinX are superior to the others (2PTX and UX) in the regions where they should perform best. These experiments were conducted on a Hard 30-bit, Goldberg's order-3 problem, with population size 60, executed for 600 generations at most.

2. Fitness proportionate selection is considered to be a poor choice by some researchers. To eliminate the possibility of any bias introduced by this strategy, we conducted a set of experiments using a tournament selection instead. Figure 10 depicts results for different crossover operators using different tournament sizes. Again, a 30-bit, Hard order-3 problem was used. As shown in Figure 10, the performance of LinX and ALinX was superior to the others for all tournament sizes. The figure also indicates that all operators seem to do better with small tournament sizes than with large ones.

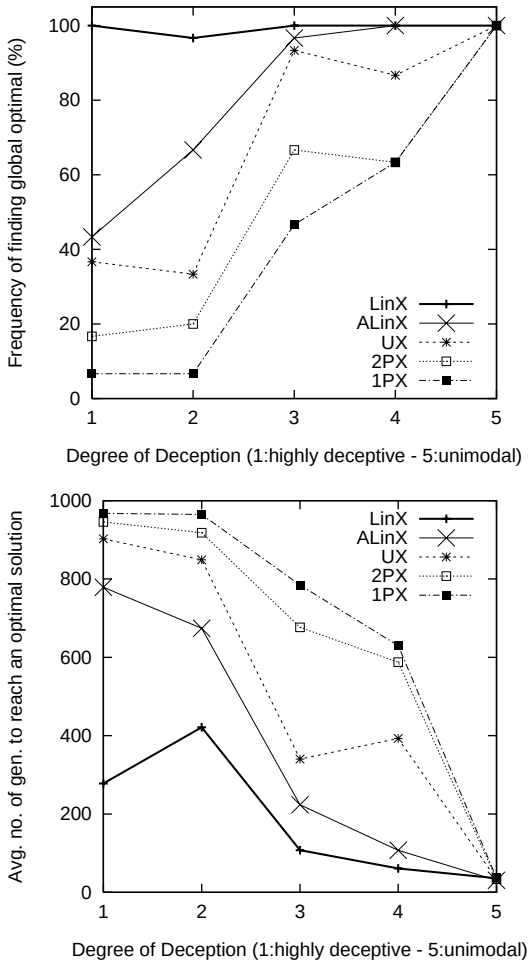


Figure 8: Effect of degree of deception on the frequency of obtaining global optima, and the number of generations required to reach a global optima: Hard order-3 problem with 30 bits, 50 population size, and 30 trials.

3. The last set of experiments was conducted to eliminate the possibility of elitism bias. Elitism degree was varied, testing the GA performance using different crossover operators. Here we used binary tournament selection for a 30-bit Hard order-3 problem. Figure 11 plots the number of trials (out of 10) in which the GA succeeded in finding an instance of the optimum solution against the number of individuals saved from extinction from the previous generation (elitism degree). The figure shows that LinX and ALinX perform better than the other operators for all elitism degrees. Also, this figure indicates that all crossover operators do better with small degrees of elitism.

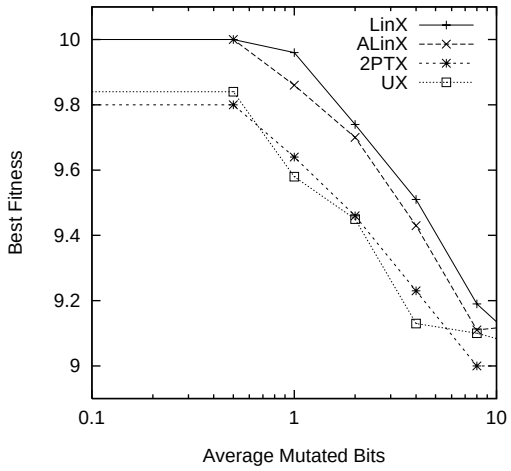


Figure 9: Variation of performance with mutation rate; Goldberg's order-3 problem with 30 bits, 60 population size, 10 trials, optimum fitness equals 10.

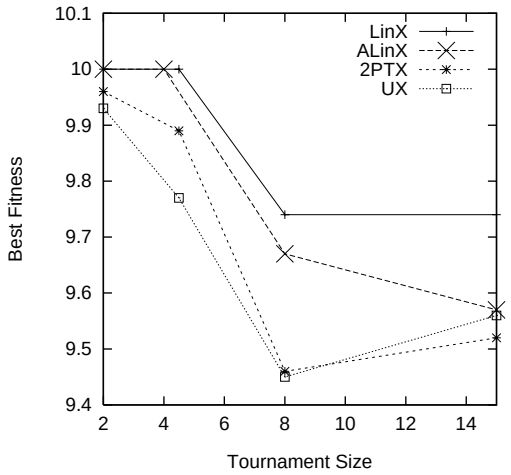


Figure 10: Using tournament selection: variation of performance with different tournament sizes; Goldberg's order-3 problem with 30 bits, 60 population size, 10 trials, optimum fitness equals 10.

8 Conclusion

This research relates the field of probabilistic inference to the application of crossover operators in genetic algorithms. Probabilistic computations have a long history and can be used with considerable advantage in GAs. The framework presented in this paper allows explicit formulation of problem-specific linkages and their subsequent use in crossover. A new class of crossover operators is presented, implemented, and tested. These operators exploit problem-specific linkages among components in a chromosome.

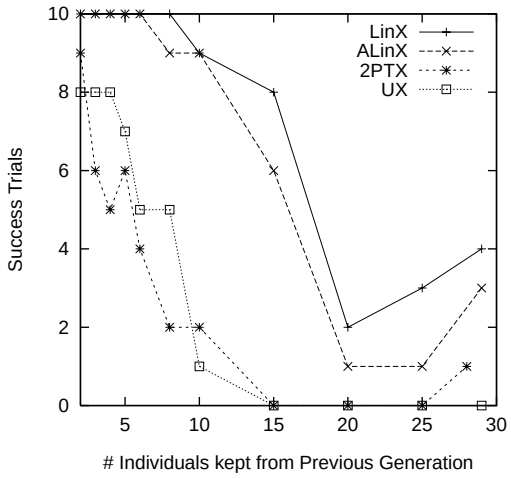


Figure 11: Varying elitism: different elitism degrees versus crossover operator performance; order-3 problem with 30 bits, 60 population size, and 10 trials.

The concept of adapting linkage between genes is shown to be effective and successful, even for problems with only partially known linkage structure. Different methods for adapting linkage probabilities are proposed and tested successfully.

For GA-easy problems with linear representation, where linkage information may be unnecessary or may exist implicitly within the representation, satisfactory results may be obtained using traditional crossover operators to avoid the computational overheads introduced by knowledge-based operators such as LinX. However, LinX does produce better results for such problems, reducing the effect of building block disruption.

For problems with unknown linkage structures, linkage adaptation procedures succeeded in identifying regions where dependencies should be strong. These regions constitute the building blocks that should be evolved together. A potentially promising approach is to stop linkage adaptation after a small number of generations and apply local improvement operators; this approach is currently being explored by the authors to reduce computation time requirements.

References

Atmar, W. (1992). On the rules and nature of simulated evolutionary programming. In Fogel, D. and Atmar, W., editors, *Proceedings of the First Conference of Evolutionary Programming*, pages 17–26, Evolutionary Programming Society, La Jolla, California.

Bandyopadhyay, S., Kargupta, H. and Wang, G. (1998). Revisiting the GEMGA: scalable evolutionary optimization through linkage learning. *Proceedings of the IEEE International Conference of Evolutionary Computation*, pages 603–608, IEEE Press, Piscataway, New Jersey.

Corcoran III, A. L. (1994). *Techniques for reducing the disruption of superior building blocks in genetic algorithms*. Ph.D. thesis, Department of Computer Science, University of Tulsa, Tulsa, Oklahoma.

Deb, K. (1991). *Binary and Floating point Function Optimization using Messy Genetic Algorithms*. Doctoral Dissertation, Department of Engineering, University of Alabama, Tuscaloosa, Alabama.

Gero, J. S. and Kazakov, V. A. (1995). Evolving building blocks for genetic algorithms using genetic engineering, *Proceedings of the IEEE International Conference on Evolutionary Computation*, volume 1, pages 340–345, IEEE Press, Piscataway, New Jersey.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts.

Goldberg, D. E. and Lingle, R. (1985). Alleles, loci, and the traveling salesman problem, *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, pages 154–159.

Goldberg, D. E., Korb, B. and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530.

Goldberg, D. E., Deb, K. and Horn, J. (1992). Massive multimodality, deception, and genetic algorithms. *Parallel Problem Solving from Nature*, volume 2, pages 37–46, Elsevier, Amsterdam, The Netherlands.

Goldberg, D., Deb, K., Kargupta, H. and Harik, G. (1993). Rapid Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms. IlliGAL Technical Report No. 93004, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, Illinois.

Harik, G. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan.

Harik, G. R. and Goldberg, D. E. (1997). Learning Linkage. *Foundations of Genetic Algorithms IV*, pages 247–262, Morgan Kaufmann, San Mateo, California.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan.

Kargupta, H. (1996). The gene expression messy genetic algorithm, In *Proceedings of the IEEE International Conference of Evolutionary Computation*, pages 814–819, IEEE Press, Piscataway, New Jersey.

Kemenade, C. V. (1998). Building blocks filtering and mixing. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 505–510, IEEE Press, Piscataway, New Jersey.

Kennedy, J. and Eberhardt, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, IEEE Press, Piscataway, New Jersey.

Levenick, J. R. (1991). Inserting introns improves genetic algorithm success rate: taking a cue from biology. In Belew, R. and Booker, L., editors, *Proceedings of the Fourth International Conference of Genetic Algorithms*, pages 53–60, Morgan Kaufmann, San Mateo, California.

Levenick, J. R. (1995). Metabits: Generic endogenous crossover control. In Eshelman, L., editor, *Proceedings of the Sixth International Conference of Genetic Algorithms*, pages 88–95, Morgan Kaufmann, San Mateo, California.

Liepins, G. E. and Vose, M. D. (1991). Deceptiveness and Genetic Algorithm Dynamics. *Foundations of Genetic Algorithms*, pages 36–50, Morgan Kaufmann, San Mateo, California.

Maini, H. S., Mehrotra, K. G., Mohan, C. K. and Ranka, S. (1994a). Knowledge-Based Nonuniform Crossover. *Complex Systems*, 8:257–293.

Maini, H. S., Mehrotra, K. G., Mohan, C. K. and Ranka, S. (1994b). Genetic Algorithms for graph partitioning and incremental graph partitioning. In *Proceedings of Supercomputing '94*, Association for Computing Machinery, New York, New York.

Mühlenbein, H., Mahnig, T. and Rodriguez A. O. (1998). Schemata, distributions and graphical models in evolutionary optimization. Submitted for publication. Available at `http://set.gmd.de/AS/ga/publi-neu.html`.

Paredis, J. (1995). The symbiotic evolution of solutions and their representations. In Eshelman, L., editor, *Proceedings of the Sixth International Conference of Genetic Algorithms*, pages 359–365, Morgan Kaufmann, San Mateo, California.

Pelikan, M. and Mühlenbein, H. (1998). Marginal distribution in evolutionary algorithms. Available at `http://darwin.chtf.stuba.sk/martin/work.html`.

Salman, A. A. (1999). *Linkage Crossover for Genetic Algorithms.* Ph.D. thesis, Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, New York.

Salman, A. A., Mehrotra, K. G. and Mohan, C. K. (1998). Adaptive linkage crossover. In Carroll, J., Lamont, G., Oppenheim, D., George, K. and Bryant, B., editors, *Proceedings ACM Symposium on Applied Computing (SAC'98)*, pages 338–342, Association for Computing Machinery, New York, New York.

Schaffer, J. D. and Morishima, A. (1987). An Adaptive Crossover Distribution Mechanism for Genetic Algorithms. In Grefenstette, J., editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 36–40, Lawrence Erlbaum, Hillsdale, New Jersey.

Smith, J. and Fogarty, T. G. (1996). Recombination strategy adaptation via evolution of gene linkage. In *Proceedings of the 1996 IEEE International Conference of Evolutionary Computation*, pages 826–831, IEEE Press, Piscataway, New Jersey.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for search. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.