Genetic Programming and Evolvable Machines, 3, 129–155, 2002

On Appropriate Adaptation Levels for the Learning of Gene Linkage

JIM SMITH james.smith@uwe.ac.uk
Faculty of Computing, Engineering and Mathematical Science,
University of the West of England, Bristol, UK

Abstract. A number of algorithms have been proposed aimed at tackling the problem of learning "Gene Linkage" within the context of genetic optimisation, that is to say, the problem of learning which groups of co-adapted genes should be inherited together during the recombination process. These may be seen within a wider context as a search for appropriate relations which delineate the search space and "guide" heuristic optimisation, or, alternatively, as a part of a comprehensive body of work into Adaptive Evolutionary Algorithms.

In this paper, we consider the learning of Gene Linkage as an emergent property of adaptive recombination operators. This is in contrast to the behaviour observed with fixed recombination strategies in which there is no correspondence between the sets of genes which are inherited together between generations, other than that caused by distributional bias. A discrete mathematical model of Gene Linkage is introduced, and the common families of recombination operators, along with some well known linkage-learning algorithms, are modelled within this framework. This model naturally leads to the specification of a recombination operator that explicitly operates on sets of linked genes.

Variants of that algorithm, are then used to examine one of the important concepts from the study of adaptivity in Evolutionary Algorithms, namely that of the level (population, individual, or component) at which learning takes place. This is an aspect of adaptation which has received considerable attention when applied to mutation operators, but which has been paid little attention in the context of adaptive recombination operators and linkage learning. It is shown that even with the problem restricted to learning adjacent linkage, the population based variants are not capable of correctly identifying building blocks. This is in contrast to component level adaptation which outperforms conventional operators whose bias is ideal for the problems considered.

Keywords: gene linkage, recombination, adaptive, self-adaptation

1. Introduction

This paper considers the learning of "Gene Linkage" within the context of genetic optimisation, that is to say, the problem of learning which groups of co-adapted genes should be inherited together during the recombination process.[1] The transmission of groups of co-adapted genes is believed to be desirable if there is epistasis between genes.[2] Conversely, it has also been recognised that "mixing" is an important factor in the design of efficient GAs [44], i.e. recombination has a role in bringing together partial solutions from different parents, as well as permitting better estimates of the fitnesses of competing schemata. Thus for a problem composed of a number of separable components, the genes in the building blocks coding for the component solutions will ideally exhibit high intra-block linkage but low inter-block linkage.

Since the foundations of the field of Evolutionary Computing, there has been a history of research in this area (see, for example [19] for a good review), and a number of algorithms have been proposed which aim to "learn" gene linkage in the context of an evolutionary system. Kargupta [19] argues that these can be seen as a search for relations which "properly delineate" the search space, and for good classes within those relations, within his *SEARCH* framework [18]. However an alternative perspective is that these linkage learning algorithms are effectively learning a simultaneous re-ordering of representations, and "crossover masks." As such, these algorithms form part of a larger body of work into "Adaptive" Evolutionary Algorithms which adapt their representations, operators or parameters, according to some learning mechanism, in an attempt to match the algorithm to the problem instance being tackled.

In [1, 40] taxonomies of Adaptive Evolutionary Algorithms are given which divide adaptive algorithms into "population," "individual" or "component" level depending on the granularity of the learning process. It is this issue, i.e., what is the appropriate level at which to learn which genes are co-adapted, which we investigate in this paper, via the use of an adaptive algorithm which can be used at any one of the three levels. This will be done by using a self-adaptive simple algorithm, which is easily adapted to work at any level, and does not rely on statistical assumptions in order to learn linkage, other than a correlation between "good" solutions and "good" strategies.[3]

For the purposes of this paper, this algorithm uses a fixed problem representation ordering, and only considers links between adjacent loci. The test suite chosen is designed to make things "easy" for the linkage learning algorithms, that is to say that where the problems are composed of a number of sub-functions which must be solved, then these functions are coded into continuous blocks rather than being interleaved along the genome. Ultimately, of course, a more general solution is desirable. However considering a restricted set of the linkage learning problem has several advantages. One example of this is that it allows for straightforward examination of the ability of the algorithms to distinguish between genes that should and should not be linked. Furthermore, whilst not able to provide positive evidence that a given scheme will be able to learn linkage under all situations, it can provide useful negative evidence, i.e., if a system cannot adapt in these most favourable of conditions, it will be able to do so in the more general case.

The rest of this paper proceeds as follows:

— Section 2 briefly introduces the topic of linkage learning, and reviews some salient work in this area.
— Section 3 introduces the concept of different levels of adaptation within algorithms, using examples of mutation, and recombination adaptation mechanisms to suggest why it is important 'that the learning takes place at an appropriate level.
— In Section 4 a model of recombination operators will be introduced based on capturing gene linkage via Boolean arrays, and it will be demonstrated that

standard operators can easily be described and generalised within this model.[4] It will be shown that the standard operators do not preserve linkage arrays, rather they are determined anew for each recombination event via the generation of a new random vector. Some well known adaptive linkage-learning algorithms e.g. *Punctuated Crossover* [31], *Gene Expression Messy GA* [17], and *LEGO* [38, 41] which make explicit use of the linkage arrays are also formally modelled within this framework.

— In Section 5 the linkage learning algorithm (*LEGO*) used in this paper is described in more detail, as are the modifications used to provide different levels of adaptation, and the experimental set up used to compare the different learning mechanisms.

— Section 6 presents and discusses empirical results and analyses of the operation of the different operators, before conclusions are drawn in Section 7.

2. Previous work

The Building Block Hypothesis [12] gives an explanation of the operation of GAs which describes evolution as a process of discovering and putting together blocks of co-adapted genes of increasing higher orders. In order to do this, it is necessary for the GA to discriminate between competing schemata on the basis of their estimated fitness. The *Messy GA* [14] was an attempt to explicitly construct an algorithm that worked in this fashion. The use of a representation that allowed variable length strings, and removed the need to manipulate strings in the order of their expression, began a focus on the notion of gene linkage (in this context gene will be taken to mean the combination of a particular allele value in a particular locus).

Munetomo and Goldberg [25] identify three approaches to the identification of linkage groups.

The first of these they refer to as the “Direct Detection of Bias in Probability Distributions,” and is exemplified by what Bosman and Thierens [5] refer to as “Distribution Estimation Algorithms” (DEAs). A good review is given in [27]. In this approach a statistical model of the density distribution of the problem is built based on the current population, and this is used to generate new samples, replacing the traditional recombination and mutation steps of an Evolutionary Algorithm. Examples of this approach include the *Estimation of Distribution Algorithm* [24], *MIMIC* [7], *Univariate Marginal Distribution Algorithm* [23], *Bivariate Marginal Distribution Algorithm* [28], *Depency Trees* [4], *Estimation of Bayesian Network Algorithms* [10] and the *Bayesian Optimisation Algorithm* [26]. Common to all of these approaches is the notion of first identifying a factorisation of the problem into a number of subgroups, such that a given statistical criterion is minimised, based on the current population. This corresponds to learning a linkage model of the problem. Once these models have been derived, conditional probabilities of gene frequencies within the linkage groups are calculated and a new population is generated based on these. The type of structures used to model linkage groups has become more complex with time, with a corresponding increase in the complexity of the problem of identifying “good” models. Although it does not include a method for selecting an appropriate linkage model, the *Factorised Distribution Algorithm* [22] has shown that given a correct linkage model, the calculation and subsequent use of conditional probabilities for offspring generation can lead to highly efficient algorithms, (although some counter examples are shown in [5]).

The other two approaches identified by Munetomo and Goldberg use more traditional recombination and mutation stages, but bias the recombination operator to use linkage information.

In the revised *Gene Expression Messy GA* [17], and the *Linkage Identification by Non-Monotonicity Detection Algorithm* (*LIMD*) [25] first order statistics based on pair-wise perturbation of allele values are used to identify the blocks of linked genes that they manipulate. These two algorithms used similar methods, but differ in that the GEMGA linkage blocks are identified for each string in the population, whereas this is done on a population-wide basis in Munetomo's work. Similar statistics are used in a number of other schemes e.g. van Kemenade's *Building Block Filtering* [45].

The third approach identified does not calculate statistics on the gene interactions based on perturbations, but rather adapts linkage groups explicitly or implicitly via the adaptation of recombination operators. The *Linkage Learning Genetic Algorithm* [16], which attempts to co-evolve the positions and values of genes using a representation which consider loci as points on a circle, with the (real-valued) distance between points representing their linkage.

The *GLinX* algorithm [30] expands on this idea, using a matrix of pairwise linkage probabilities, in tandem with a method (based on an assumption of Conditional Independence) of estimating more complex conditional linkage probabilities. This permits "on-the fly" calculations of probabilities to use for inheriting a gene from either parent, given the pattern of inheritance so far during the crossover operation. They present results on a number of problems which show improved performance over conventional operators, if the linkage pattern is known a-priori and can be hard coded into the linkage array. They also present results from an adaptive algorithm, *AlinX*, which uses Hebbian learning to adapt the contents of the pairwise linkage matrix during evolution.

The *LEGO* (to be described later) and *APES* algorithms both explicitly considered the population as being composed as evolving blocks, made up of chains of linked adjacent genes. Improved performance on a number of test problems were found when compared to "standard" recombination operators [37, 42]. Finally due mention should be given to a number of schemes that have been proposed for adaptively learning crossover points and crossover masks e.g. [20, 21, 31, 36, 46] to name but a few from the literature. Typically within a GA, the mutation operator is applied with a low per-gene probability of perturbing the allele value, and so the search for good crossover masks can also be seen as implicitly learning gene linkage.

3. Adaptation in evolutionary algorithms

The efficiency of the Genetic Algorithm can be seen to depend on two factors, namely the maintenance of a suitable working memory (the population), and quality of the match between the probability distribution function (pdf) generated by the genetic operators operating on that population and the landscape being searched. The first of these factors will depend on the choices of population size μ, and selection algorithm. The second will depend on the action of the reproductive operators (recombination and mutation) and the set of associated parameters on the current population.

Naturally, a lot of work has been done on trying to find suitable choices of operators and their parameters which will work over a wide range of problem types. The first major study [6] identified a suite of test functions and proposed a set of parameters which it was hoped would work well across a variety of problem types. However later studies using a *meta-ga* to learn suitable values [15] or using exhaustive testing [32] arrived at different conclusions. Meanwhile theoretical analysis on optimal population sizes [13] started to formalise the (obvious?) point that the value of μ on the basis of which decisions could be reliably made depends on the size of the search space. These considerations, when coupled with interactions with other Evolutionary Algorithm communities who were already using adaptive operators (e.g. the $(1+1)$ [29] and $(\mu\lambda)$ [34, 35] Evolutionary Strategies), has led to an ever increasing interest in the possibilities of developing algorithms which are able to adapt one or more of their operators or parameters in order to match the p.d.f. induced by the algorithm to the fitness landscape. The reader is directed to e.g. [40] for a review of operator and parameter adaptation in Genetic Algorithms.

The terminology of [1, 40] defines three distinct levels at which adaptation can occur in evolutionary algorithms. *Population-level* adaptations make changes which affect the pdf contribution from each member of the current population in the same way. *Individual-level* adaptations make changes which affect the pdf contribution from each member of the population separately, but apply uniformly to each of the components of the individuals' representation. At the finest level of granularity are *Component-level* adaptations, where the pdf contributions from each component of each member may be changed individually.

Within the context of linkage learning, the inductive learning of crossover masks in [36] can be seen as a population level adaptation, since the rules about which crossover masks to use are applied to all crossover events. Similarly the *GLinX*, *ALinx*, and *LIMD* algorithms rely on population based statistics in order to decide from which parents genes should be inherited during crossover.

By contrast, the *GEMGA* algorithm, although it uses population statistics to initialise the linkage sets attached to each individual, only considers the sets attached to the two parents during crossover, and each member of the population can evolve different sets, i.e. this is an individual level adaptation, as are many other approaches e.g. [16, 31, 46]. The *LEGO* and *APES* algorithms work exclusively on the level of blocks of genes, and only consider whole solutions during evaluation: these are component level adaptations.

Within the field of Evolution Strategies, where the emphasis is principally on mutation as a search operator, there has been a historical move from population level adaptation (e.g. the 1:5 success rule) through individual level adaptation (e.g. the (μ, λ) ES) through to component level adaptation (the Covariance Matrix approach), which reflects a view that different search strategies may be appropriate in different parts of the search space, and that, ultimately, it may be necessary to separate the adaptation from the basis of the representation. However, each increasing level of complexity increases the size of the space of possible parameters being searched, with a corresponding increase in the overhead of learning appropriate strategies.

Within the study of adaptive recombination operators, it is not clear what is the most appropriate level for adaptation. Population level strategies often rely on large amounts of "bookkeeping," which by default means that strategies are adapted according to information which is possibly not related to the current state of the population.[5] On the other hand, the move to individual level adaptation, although permitting the simultaneous exploration of a range of different strategies, increases the learning overhead, and one frequently noted problem [16, 31, 38] is that as the population converges, so there might not be sufficient information available to distinguish between the merits of different groups of linked genes. As Goldberg put it *all our attempts to evolve linkage without intervention* (*inversion, crossover, etc.*) *have failed because the alleles want to converge before the linkage gets tight*.[6]

This raises a fundamental question about the true purpose of attempting to learn Gene Linkage. If the aim is to obtain definitive patterns of linkage, which can be used to inform future attempts to optimise the same, or similar problems, then the effects of convergence on the population can be seen to present a huge impediment to achieving this goal via the use of Adaptive Algorithms. This can be seen by considering a two locus problem where the allele patterns $\{00, 11\}$ contribute more highly than $\{01, 10\}$. If the population converges in either position, then any information learned early in the run about the value of tight linkage between those two loci will get lost amongst the subsequent data which shows linkage to be neutral between those two genes.

If, on the other hand, we take a "one-off" view, i.e. that that the purpose of attempting to learn Gene Linkage is solely to aid the process of optimisation during the current run, then to some extent these concerns become immaterial, *as long as the adaptive process does not become dominated by information garnered from past populations*. Of course all learning algorithms rely on generalising from past information to some extent, however this analysis suggests that there may be a benefit to individual or component based approaches, since these by their nature take account of the fact that the individuals to which they are attached are impermanent.

So far this discussion has deliberately concentrated on the level of linkage learning via adaptive recombination operators, as this is the main focus of this paper. As noted above there exists a class of population based algorithms (DEAs) which do not use recombination or mutation operators, but which also learn linkage via the selection of appropriate models for density estimation. It is necessary to consider whether the discussion above on the levels of adaptation has any relevance to these algorithms.

On one hand, the definition of "adaptation levels" relies on the concept of changes in the way that operators are applied to individual points (pairs for crossover, or more generally sets for recombination) in the search space. By contrast, in DEAs once the population has been analysed to select and then fit an appropriate distribution model, it is discarded, i.e., there are no operators to be adapted, and so the distinction between adaptation levels is largely irrelevant for these algorithms.

Nevertheless, DEAs do learn linkage models, and it would appear at first inspection that they do this at the population level, since they rely on probabilistic models based on population statistics. However, the scoring criteria used in the search for a "good" model relates specifically to the contents of the groups (chains, trees or networks) using metrics such as Minimum Description Length, Bayesian Dirichlet metric etc.

Furthermore, as is demonstrated in [5], it is not sufficient for these algorithms merely to identify linkage models in order to perform optimisation, rather they must also correctly and rapidly identify "building blocks," i.e. allele values within the linkage sets. This focussing on particular hyperplanes, rather than merely relations, is the equivalent within an adaptive GA of identifying linkage sets (relations), and then tuning the mutation rate within those sets once the correct hyperplanes have been identified, and suggests that these algorithms are closer in spirit to component level algorithms such as *APES* rather than population level algorithms.

4. A model for gene linkage

In this section a model of gene linkage is developed that permits the description of a number of different recombination operators in terms of the way that they manipulate and preserve linkage between loci within an individual. This model associates with each member of the population a Boolean array A of size $l \times l$[7] which encodes for gene linkage within the problem representation. Recombination is considered in part as a function which acts on these arrays: $R(A) \to A'$, such that the contents of the parents' arrays after the operator is applied determine which genes in the parents will be transmitted together, and hence which schemata will be preserved. When the allele values from loci in a selected parent are copied into the offspring, the corresponding columns of that parent's linkage array are copied into the array of the child, (even though in many cases they will be completely rewritten by the application of the recombination operator in the next iteration).

Conventional recombination operators take a fixed number of parent individuals as their inputs,[8] and the same transformation is applied to the linkage array of each (as will be described in the next section). In this case the offspring will initially have the same linkage array as its parents. A number of adaptive algorithms have been proposed in which the linkage patterns are learned, and for these operators the parents will in general have differing arrays, as will the resultant offspring.

A recombination operator is thus defined by its arity, the function R, and a representation of the random choices made by the operator, such as the crossover points in N point crossover, and the choice of parents in uniform or scanning crossover.

To give a simple example, which will be formalised later, One Point Crossover works by taking two parents and selecting a crossover point at random, which effectively splits each parent into two blocks of linked loci. In both parents all of the array elements corresponding to inter-block links are set *true*, and those corresponding to extra-block links (i.e. links "across" the crossover point) are set *false*. In other words the effect of this operator is to rewrite both arrays completely, according to a randomly chosen number (the crossover point).

In this model, each element A_{ij} of the linkage array takes the value *true* or *false* according to whether there is a link from locus i to locus j. In many cases an unambiguous definition of the recombination operators requires a symmetrical array, but this is not always the case. Two loci are considered to be linked (i.e. remain together under the recombination operator) if and only if both of the relevant array elements are set *true*. For the purposes of this discussion the problem representation will be assumed to be linear, i.e. string-based.

Having defined the linkage array, it is useful to formally define some terms, starting with a Boolean function that tells us whether two loci are linked.

$$\forall\, i, j \in \{1, \dots, l\} \bullet \mathit{Linked}(i, j) = A_{ij} \wedge A_{ji} \tag{1}$$

One property common to all operators is that since each locus is by definition linked to itself, then the principal diagonal of the array is composed entirely of *true* elements, i.e.

$$\forall\, i, j \in \{1, \dots, l\} \mid (i = j) \bullet \mathit{Linked}(i, j) \tag{2}$$

This may be read as *for all pairs, i and j, of loci between* 1 *and l such that* $i = j$ *it holds that i is linked to j*.

Finally we can define the *Linkage Set*, S_i of locus i as the set of loci which are linked to i:

$$S_i = \{j\} \in \{1, \dots, l\} \bullet \mathit{Linked}(i, j) \tag{3}$$

In order to specify a recombination operator, it is also necessary to represent the arity, a, and the random choices made when the operator is applied. This is done via a randomly created vector $\bar{x}$ of size l. The contents of $\bar{x}$, and the way that they are specified, vary from operator to operator as will be seen.

4.1. *Static recombination operators*

4.1.1. n-point crossover. For n-point crossover operators, the elements of $\bar{x}$ are integers which represent the number of crossover points to the left of the corresponding loci. They are monotonically nondecreasing, with the changes in value corresponding to the randomly chosen crossover points. Formally:

$$\begin{aligned}\forall\, i \in \{1, \dots, l\} \bullet ((i < l) \wedge x_i \in \{0, \dots, n\} \\ \wedge ((x_{i+1} - x_i) \in \{0, 1\})) \vee (x_i = n)\end{aligned} \tag{4}$$

Using definition (4) with $n = 1$ (and by necessity arity 2) allows the specification of One Point Crossover (1X) as:

$$\begin{aligned}R_{1X}(A) \rightarrow A' \mid \forall\, i, j \in \{1, \dots, l\} \bullet ((x_i = x_j) \wedge \mathit{Linked}'(i, j)) \\ \vee(\neg A'_{ij} \wedge \neg A'_{ji})\end{aligned} \tag{5}$$

where the function $Linked'()$ is taken to refer to the array after the operator is applied.

This definition states that two loci are linked if there are the same number of crossover points (0 or 1) to the left of them in the representation. If there are not the same number of crossover points, then according to (5) both of the elements in the array which determine the link are set false. This ensures an unambiguous specification of A'.

Setting $n > 1$ in (4), increasing the arity accordingly to $n + 1$, and using (5) to specify R produces Diagonal Crossover (9).

In the case where $n > 1$, but the arity is two (i.e. "traditional" two-point and n-point crossover) it is necessary to take account of the fact that alternate blocks are also linked. The random vector $\bar{x}$ is still defined as per (4), and the linkage of alternate blocks is catered for by noting that this equates to the x_i values yielding the same result $mod(2)$ if they are linked. The generalised two-parent n-point crossover is defined by (4) and:

$$R_{2,nX}(A) \to A' \mid \forall i, j \in \{1, \ldots, l\} \bullet ((mod_2(x_i) = mod_2(x_j)) \wedge Linked'(i, j)) \vee (\neg A'_{ij} \wedge \neg A'_{ji}) \tag{6}$$

Although Diagonal Crossover picks n crossover points at random and then selects the blocks from $n + 1$ parents successively, it would be simple to generalise this to any number of parents, so that if the arity a is less than n more than one block is picked from some parents (e.g. (6) when $a = 2$). If the arity is greater than $n + 1$, then this generalisation is still valid as long as more than one child is created per crossover application. Thus the general form of this class of recombination operators where n crossover points are picked at random and offspring created from a parents is:

$$R_{a,nX}(A) \to A' \mid \forall i, j \in \{1, \ldots, l\} \bullet ((mod_a(x_i) = mod_a(x_j)) \wedge Linked'(i, j)) \vee (\neg A'_{ij} \wedge \neg A'_{ji}) \tag{7}$$

where the vector of crossover points, $\bar{x}$, is defined as per 4.

4.1.2. The Uniform Crossover family of operators. Uniform Crossover can be represented in one of two different ways. The essence is that at every locus a random choice is made as to which parent the gene should be copied from. Although the more usual case has a 50% probability of selecting from either parent, some authors (e.g. [43]) have championed the cause of parameterised uniform crossover, so that genes are copied from the first parent with probability p. The arity of Syswerda's original version, and most implementations, is two.

The first difference between this family of operators and the n-point crossovers above is immediately apparent: in the former n crossover points were chosen at random, and the vector $\bar{x}$ filled in accordingly. For the Uniform Crossover family of operators, a set of l random values must be chosen. A second apparent difference is that in the former the number of crossover points is known, whereas in the second it may vary between 0 and $l-1$. However it should be noted that since the values are chosen at random from $\{1, \dots, l\}$ for n-point, the value n is in fact an upper bound on the number of crossover points, as they may coincide, and also because a crossover point at either extreme of the representation has no effect.

One representation of two parent Uniform Crossover which makes this distinction clear, simply notes that two adjacent loci will remain linked in both parents if the corresponding random numbers are both either less than or greater than p. Using a vector $\bar{r}$ to denote a random vector where each r_i is selected uniformly from $[0, 1]$, the elements of $\bar{x}$ are given by:

$$\begin{aligned} x_1 = 0, \forall i \in \{2, \dots, l\} \bullet ((r_{i-1} < p) \wedge (r_i < p) \wedge (x_i = x_{i-1})) \\ \vee((r_{i-1} \geq p) \wedge (r_i \geq p) \wedge (x_i = x_{i-1})) \vee (x_i = x_{i-1} + 1) \end{aligned} \tag{8}$$

Using (8) to determine the vector $\bar{x}$, the function R is then defined exactly as per (6). It is evident that this model could be used to generate a class of multi-parent operators by taking any values for the arity, a, and using (7) to specify R. This class of operators would have a crossover point between adjacent loci with probability $2p(1-p)$, and hence a mean number of $2p(l-1)(1-p)$ crossovers per genome. However for $a > 2$ this class is no longer isomorphic to Syswerda's original specification.

An alternative representation, which is more easily generalised to models with arity a, such as Scanning Crossover (8), is for the elements of $\bar{x}$ to be drawn from the set $\{0, \dots, a-1\}$ according to some probability distribution. This might be uniform, fitness proportionate, or parameterised as above. The generalized representation becomes:

$$\begin{aligned} R_{UX}(A) \rightarrow A' \mid \forall i, j, \in \{1, \dots, l\} \bullet ((x_i = x_j) \wedge Linked'(i, j)) \\ \vee(\neg A'_{ij} \wedge \neg A'_{ji}) \end{aligned} \tag{9}$$

Note the similarity to (5).

Using a to denote the arity, the elements of x are drawn from $\{0, \dots, a-1\}$ as:

$$x_i = \begin{cases} 0 & r_i \leq p \\ 1 & r_i > p \end{cases} \qquad \text{Parameterized Uniform Crossover} \tag{10}$$

$$x_i = int(r_i.a) \qquad \text{Uniform Scanning} \tag{11}$$

$$x_i = j \bullet \sum_{k=0}^{j-1} fit_k < r_i \sum_{k=0}^{a-1} fit_k \leq \sum_{k=0}^{j} fit_k \qquad \text{Fitness Proportionate Scanning} \tag{12}$$

This definition of $\bar{x}$ for Fitness Proportionate Scanning is an implementation of the "roulette wheel" algorithm, and the variable k is taken to be an index into the set of parents. Other selection methods could easily be implemented with the corresponding changes to (12).

4.1.3. Other static reproduction operators. All of the recombination operators above utilize a new randomly generated vector $\bar{x}$ to transform the parents' linkage arrays each time they are applied. Although the definition of the operator in terms of its effect on the linkage array is constant, the resultant arrays A' may be different for each application. By contrast mutation based search and Bit Simulated Crossover employ strategies that can be modelled as having a completely fixed and identical linkage array for each individual. Algorithms that employ no recombination select an individual from the population and copy it whole. The effect of this "asexual reproduction" is to treat each individual as having a completely linked genome, i.e. the linkage set consists of every locus. This can be modelled as:

$$R_{asexual}(A) \rightarrow A' \mid \forall\, i, j \in \{1, \ldots, l\} \bullet Linked(i, j) \wedge Linked'(i, j) \tag{13}$$

By contrast, the action of Bit Simulated Crossover is to treat each individual as having a completely unlinked genome, i.e., all the entries in the linkage array are false:

$$R_{BSC}(A) \rightarrow A' \mid \forall i, j, \in \{1, \ldots, l\}, i \neq j \bullet (\neg A'_{ij} \wedge \neg A_{ij} \wedge \neg A'_{ji} \wedge \neg A_{ji}) \tag{14}$$

4.2. Adaptive recombination operators

In Section 4.1 a number of recombination operators were described in terms of the generation of a set of random numbers which are used to specify a vector $\bar{x}$ which instantiates one application of the operator. For the n-point and Uniform crossover families the specification of R is such that the linkage arrays in all parents are completely and identically rewritten according to the contents of $\bar{x}$, i.e., no linkage information is accrued over time.

However, it has repeatedly and persuasively been argued (e.g. [14, 16, 44]) that designers of genetic algorithms ignore linkage at their peril. As was mentioned above part of the rationale behind the development of the messy-ga and its variants was to identify and amass building blocks and intrinsic to this was the notion of linkage.

4.2.1. The gene expression messy GA. The messy-ga itself uses a variable length encoding which allows for under and over-specification, coupled with mechanisms such as "competitive templates" for filling in missing parts of a representation, all of which make it hard to analyse in terms of the model above.

The latest variant, the "Gene Expression Messy GA" uses first order statistics to identify linked blocks, and a reproduction operator that works on these linkage sets. The algorithm maintains linkage sets for each gene in much the same way as described above. The recombination operator works by selecting a number of genes from one parent at random. For each member of this "Exchange Set," (provided certain first order statistical criteria are met), the gene and its linkage set are copied into the opposite parent. The problem of "block conflict" (see the next section) is surmounted by not insisting that the linkage sets are coherent. This can be represented by dropping the linkage criterion (1) and allowing each row of the array to be considered separately.

Because the arrays are potentially different in each member of the population, (i.e., this is an *Individual Level* adaptation mechanism) it is no longer sufficient to consider a single array A. Instead A will refer to the "donor" parent, B to the recipient and C to the offspring. Ignoring the statistical criteria, and denoting the Exchange set as X, the rows in the linkage array of the offspring C (and the corresponding genes) are given by:

$$\begin{aligned}&\forall i \in \{1, \ldots, l\} \bullet (i \in X) \wedge \forall j \in \{1, \ldots, l\} \bullet (C_{ij} = A_{ij})\\ &\quad \vee ((i \notin X) \wedge \forall j \in \{1, \ldots, l\} \bullet (C_{ij} = B_{ij}))\end{aligned} \tag{15}$$

Unfortunately, the inconsistencies introduced also provide scope for spurious linkage correlations to propagate. Although this algorithm is claimed to guarantee convergence to the optimal solution for problems where the building blocks are of a known order k, there are two problems. Firstly, as shown in [44], the population size required to guarantee convergence to the optimal solution scales exponentially with k. Secondly, and more importantly for practical implementations, k is, in general, unknown. This has the unfortunate effect that if k is underestimated, and there is deception of order $\approx k$ in the problem, then the algorithm is guaranteed to converge to a sub-optimal result.

Another variant of this approach, which explicitly attempts to learn values from the range $[0, 1]$ to enter into the linkage array is seen in (16). Again, this uses a dynamic encoding where genes have the form (position, value), and a cut and splice crossover operator. The model used allows some calculations to be made about the bias of the operator in terms of the linkage distribution, provided certain assumptions about the proportion of optimal building blocks are made. Unfortunately when run in the context of a "real" GA, i.e. with selection, the algorithm did not display any appreciable learning of appropriate linkage. This was attributed to the effects of convergence. Essentially it was found that using the crossover operator alone as a means of exploring the space of linkages was too slow, since once selection has converged the problem encoding, mixing or linking genes has no effect. This arises partly because the algorithm was trying to learn real valued linkages, rather than restricting itself to a Boolean space.

4.2.2. Punctuated crossover. In the Punctuated Crossover mechanism [31], self adaptation is used to govern the evolution of the linkage array. The problem representation is augmented by the addition of a binary flag between each pair of adjacent loci, which encodes for a crossover point. Successive genes (and crossover bits) are copied from the first parent into the offspring until a crossover point is encountered in either parent. At this point genes start to be copied from the other parent, *regardless* of whether it coded for crossover at that point. This continues until a full child is created, and a second child is created as its complement. This is equivalent to associating with each individual a tridiagonal linkage array, of the form:

$$A_{ij} = \begin{cases} 0 & |i-j| > 1 \\ 1 & i = j, j+1 \\ 0, 1 & i = j-1 \end{cases} \tag{16}$$

The values are subject to mutation at the same rate as the problem bits, which corresponds to the generation of the random vector and the transforming of the arrays.

In order to define the linkage array, C, of the first offspring, it is useful to keep a running count of the number of times crossover has occurred so far during transcription. This will be denoted by n, and for brevity the parents' linkage arrays will be subscripted by this value. If the parents are A and B, then $A_{n,ij} = A_{ij}$ if $mod_2(n) = 0$, and B_{ij} if $mod_2(n) = 1$. The genome and linkage array inherited by the first offspring are given by:

$$C_{ij} = \begin{cases} A_{0,\,ij} & (i = 1) \wedge (j = i+1) \\ A_{n,\,ij} & (i > 1) \wedge (j = i+1) \wedge A_{n,\,ij} \wedge A_{n+1,\,ij} \\ A_{n+1,\,ij} & (i > 1) \wedge (j = i+1) \wedge (\neg A_{n,\,ij} \vee \neg A_{n+1,\,ij}) \\ 1 & i = j, j+1 \\ 0 & |i-j| > 1 \end{cases} \tag{17}$$

The first line of this equation gives the start conditions. The fourth and fifth lines, in conjunction with the clauses $j = i+1$ ensure that the linkage array of the offspring obeys the tridiagonal condition (16). The second line states that genes and the linkage array elements continue to be inherited from the current parent (n) as long as the linkage flag is set in both that and the complementary ($n+1$) parent, and the third line states that crossover occurs if either parent has the corresponding entry of their linkage array set to *False*.

Unfortunately this implementation did not preserve much linkage information from either parent. It was demonstrated in the original paper that it was possible for two parents with "medium" numbers of crossover points to create two offspring, such that the linkage array attached to one child contained all the crossover points from both parents.

4.2.3. Linkage evolving genetic operator (LEGO). It was shown in the previous section that both the N-point and Uniform families of Crossover Operators can be defined in terms of linkage arrays, with the randomness in each instantiation captured by a random vector $\bar{x}$. Similarly other adaptive operators can be defined in terms of linkage arrays, with the random choice of parents and mutation taking the place of $\bar{x}$. It follows that one way of specifying an adaptive recombination operator, which can adjust its arity and linkage arrays to the problem in hand, is to encode the linkage array within the genome of each individual and to use self adaptation to govern the growth of different linkage sets, and hence recombination strategies. A benefit of using a separate binary linkage array for each individual is that it greatly reduces the size of the search problem compared to using real values (i.e. a probabilistic array). The benefits of evaluating different strategies, which accrue naturally from a probabilistic array in a population level approach, are achieved via the use of multiple (μ) binary arrays.

Based on this approach, *LEGO* [38, 41, 42] was developed as a means of testing the viability of this approach. In brief, the operator works by attaching a linkage array to each member of the population, which partitions the loci in that member into a number of linkage sets. The initial implementation of *LEGO* only permitted linkage to occur between adjacent loci, i.e. the evolved linkage sets consist of chains of linked genes. This has an obvious potential weakness in that it cannot capture tight linkage between two non-adjacent genes unless the entire set of genes between them is also linked, i.e., it places a heavy emphasis on the chosen representation, unlike some of the schemes described above. The rationale for this was that as has already been noted that learning linkage is a second order effect, and that there is often a problem with allele convergence preventing the algorithm from finding good "blocks." By only considering adjacent linkage, the size of the problem space is reduced from $O(l^2)$ to $O(l)$. Furthermore, when the practical implementation is considered, problems of conflict resolution are immediately found with the use of non-adjacent linkage, which require an arbitration process via which would be decided the parts of the accrued linkage information to be ignored during recombination.

In practice, analysis of the evolved linkage on a number of different problem types [39] showed that when the "true" linkage pattern was not adjacent, then highly linked chromosomes tended to evolve, i.e. the system evolved to avoid disrupting groups of co-adapted genes, at the expenses of mixing. Note that possible ways of avoiding this problem include re-ordering prior to chain linkage (as is done in the MIMIC algorithm) or removing the restriction to adjacent linkage by proving a means of block conflict resolution, but these will not be considered here.

The operator is not restricted to selecting from only two parents, so the genome can be considered as being comprised of a number of distinct linkage sets, or blocks. Unlike the case for two-parent recombination, alternate blocks on the same parent are not linked. The linkage set of a locus i can now be redefined to reflect the constraints imposed:

$$\begin{aligned} S(i) = \{j \in \{1, \ldots, l\} \mid & (j = i) \\ & \vee ((j < i) \wedge \forall k \in \{j, \ldots, i-1\} \bullet \mathit{Linked}(k, k+1)) \\ & \vee ((i < j) \wedge \forall k \in \{i, \ldots, j-1\} \bullet \mathit{Linked}(k, k+1))\} \end{aligned} \tag{18}$$

When an offspring is created, blocks are chosen sequentially from left to right. In the first locus a parent (denoted X_1) is chosen according to the pdf p_r defined by the action of selection operator on the entire population.[9] This block is then copied whole, that is to say that every gene from the linkage set of gene $a_{1,1}$, is copied into the new individual.

If there are $s_1 = \mid S11 \mid$ elements in this first set, then, by definition of the linkage set, in both the offspring and a_1, $\neg \mathit{Linked}(s1, s1+1)$. Providing that $s_1 < l$ there is now a new competition to select the next block.

Because the whole purpose of the operator is to preserve information carried in the linkage array, the selection of partial blocks is forbidden. Thus the selection of the next parent is restricted to those with eligible blocks; i.e. the next block selected must have a left hand edge abutting the right-most edge of the partially completed offspring. This requires a dynamic calculation of p_r, to yield a modified pdf p'_r given by the action of the selection operator on the restricted set of permissible parents. The pdf for parental selection for the next block will look like:

$$p''_r(a^t_i) = \begin{cases} 0 & Linked(a^t_{i,s1}, a^t_{i,s1+1}) \\ p'_r(a^t_i) & \neg Linked(a^t_{i,s1}, a^t_{i,s1+1}) \end{cases} \tag{19}$$

Having chosen a parent X_2 the block $S_2(s1+1)$ is copied into the new individual and the process repeats until the offspring is fully specified.[10]

The Lego recombination operator has the form $R : (I \times I \times I)^{\mu} \rightarrow I$. Using O_i to denote the ith offspring produced and $O_{i,j}$ to denote the jth locus of that offspring, the full definition of the operator is given by:

$$O_{i,j} = \begin{cases} X_{1,j} & j = 1 \\ X_{k,j} & Linked(X_{k,j-1}, X_{k,j}) \; 1 \leq j \leq l \\ X_{k+1,j} & \neg Linked(X_{k,j-1}, X_{k,j}) \end{cases} \tag{20}$$

where the parents X_k, X_{k+1} are selected from the population as required (i.e. whenever the linkage criterion is not met) using the modified pdf $p''_r(a^t_k)$. Since the linkage arrays L_i and R_i of the population define the recombination strategy, and are themselves potentially changed during both recombination and mutation, the transition function for the recombination function is given by $\Gamma_R(A_i)$. R is defined by (20) and the effect of mutation, M, on the linkage is given by:

$$\begin{aligned} M(A) \rightarrow A' \mid \forall i, j \in \{1, \ldots, l\} \bullet ((\mid i - j \mid = 1) \wedge (x_{ij} \leq p) \\ \wedge (A'_{ij} \neq A_{ij})) \vee (A'_{ij} = A_{ij}) \end{aligned} \tag{21}$$

where p is the mutation rate and x_{ij} is a random number sampled afresh from $[0, 1]$.

In [42], the effects of iterated recombination and mutation are examined. It is shown that for an infinite population, in the absence of any selection pressure with respect to linkage, the system will rapidly evolve to a steady state. This steady state will retain all four combinations of links between two adjacent genes, and will thus retain the ability to adapt to changing circumstances. The proportion of genes linked at the steady state is entirely independent of the starting conditions, being solely a function of the mutation rate, p, applied:

$$mean\ linkage\ rate = \left(\frac{\sqrt{2 + p(1-p)} - \sqrt{p(1-p)}}{2} \right)^2 \tag{22}$$

Thus any deviation from this pattern indicates a selection pressure for or against linkage.

5. Description of linkage evolving operators used

5.1. Representation

The *LEGO* operator is implemented by adding two Boolean arrays to the representation of each individual, denoting whether each locus is linked to its neighbour on the left and right respectively.[11] For the sake of clarity these arrays have length l, although in practice only $l-1$ of the positions are used since the problem representations are not toroidal. This gives an extended representation for the population as:

$$P^t = (a_1^t, \dots, a_\mu^t) \in (I \times I \times I)^\mu \tag{23}$$

Denoting the three arrays coding for the problem, left linkage and right linkage as α, L and R respectively, an individual comprises the 3-tuple $a_i^t = (\alpha_i^t, L_i^t, R_i^t)$. This will be referred to as the genome of the individual, and a single gene at locus j consists of the relevant entries from all three arrays: $a_{i,j} = (\alpha_{i,j}, L_{i,j}, R_{i,j})$. The linkage array $A(i)$ for the ith member of the population is given by:

$$A_{i,jk} = \begin{cases} 0 & \mid j-k \mid > 1 \\ 1 & j = k \\ R_{i,j} & j = k-1 \\ L_{i,j} & j = k+1 \end{cases} \tag{24}$$

The linkage criterion, (1), for two genes j and k, (where without loss of generality $j < k$) becomes:

$$\mathit{Linked}(j,k) \equiv (j = k-1) \wedge L_k \wedge R_j$$

Initially the Boolean values in the linkage arrays are set using a biased random choice of value so as to create a range of different block sizes in the initial population, representing a variety of recombination strategies. It was found that using a probability of 90% for setting each linkage value to *True* gave a good mix of block sizes in the population. In order to implement the mutation scheme, there is a small probability of changing the value of a link bit via the application of bit-flipping mutation at a low rate ($1/l$).

This scheme was implemented in a steady-state GA, using tournament selection for blocks, and with the new offspring replacing the oldest member of the population if fitter. It was found in [42] that a steady state model gave improved performance and clearer adaptation patterns, this ties in with the well known need for increased selection pressure in self-adaptive algorithm (see e.g. [2, 39]) and also the findings of Eiben et al. [9] that multiparental recombination operators worked better in a steady state setting.

The effects of population level adaptation are examined by the use of three different schemes. In each case the whole population is used to generate the linkage information which governs which genes are inherited together, but this is decoupled from the actual choice of which parent the allele values at each locus are inherited from. The first two cases have arity 2—that is to say that the linkage arrays effectively form a crossover mask for two-parent n-point crossover. Because the component scheme can potentially mix genes from l parents (the totally unlinked case with $\mu \geq l$), there is potential for faster mixing of building blocks, so a third scheme, also of indeterminate arity, was implemented.

The first scheme (Pop. v.1) uses a probabilistic interpretation of the linkage arrays in the evolving population. These are combined to form a vector $\bar{y}$ where each parent X contributes a value to y_i if $\neg Linked(X_{i-1}, X_i)$. These values are weighted using linear ranking over the whole population, with fittest parent weighted at twice the mean. This is done to provide a (roughly) equivalent selection pressure at the level of linkage to that of the component level. A random vector $\bar{x}$ is generated, and compared to $\bar{y}$, to create a crossover mask as per Parameterised Uniform Crossover. This is used to create an offspring from two parents, and inherited as the linkage array of the offspring. Each of the parents is selected via deterministic binary tournaments, creating the same selection pressure at the allele level.

In the second version (Pop. v.2), the linkage information in the whole population is used to select a number of blocks as per the basic *LEGO*. Linkage information is inherited in this fashion, however the actual allele values within the blocks are inherited from two parents selected by tournaments as above. In other words, the alleles in the first block are inherited from X_1, the second from X_2, then the third from X_1 and so on until the offspring is completely specified. This has the effect of separating the process of learning linkage arrays from that of learning allele values, thus moving from component to population level adaptation.

As noted above, the third variant of population level adaptation (Pop. v.3) is designed to separate any effects arising from greater mixing with more than two parents. In this case, linkage blocks are selected as per the component and second population level schemes, but the actual allele values are inherited from a parent chosen by a fresh binary tournament for each block.

5.2. *The test suite*

A set of well studied test problems which displayed a range of epistasis and multimodality were selected. All of the problems selected had known maximum values, and the algorithms were compared on the basis of the time taken to find the optimum.

It should be recognised that there is no single commonly accepted method of comparison, and that different results might be obtained if a different performance criterion were used. Preliminary results show that performance improvements can be obtained for the adaptive algorithms by tuning the link mutation rate, but that the algorithms are all fairly robust to this setting, and the relative order of the results for the four versions remains the same. Similarly the performance of the algorithms was not extensively tuned with respect to selection pressure and population size. The purpose of this work is to examine the relative behaviour of the three different linkage learning algorithms, rather than to produce "fast" linkage learning algorithms compared to the static algorithms tested, and as noted, the optimal set of parameters for any algorithms will tend to depend on the performance metric chosen. However it should be noted that the three sets of results following all show the same pattern, across a range of population sizes.

The test problems were as follows:

1. One-Max: This was run with a genome of length sixty four, and is defined as:

$$f(a) = \sum_{i=0}^{l} a_i \quad a_i \in \{0, 1\} \tag{25}$$

   This is a simple problem, which has no epistasis or deception. Since it is separable, it would be expected that there would be no selective advantage towards Gene Linkage on this problem.
2. Royal Road: This is a sixty four bit problem effectively identical (given the performance measure) to the R1 function (11), and similar to Eshelman and Schaffer's "plateau" functions [33]. It is comprised of eight contiguous blocks of eight bits, each of which scores 8 if all of the bits are set to one, i.e:

$$f(a) = \sum_{i=0}^{7} \prod_{j=i*8}^{i*8+7} a_j \tag{26}$$

   Although there is no deception in this problem there is an amount of epistasis. This problem was designed to be "ga-easy" but Forrest and Mitchell reported that it had been solved faster by Random Mutation Hill-Climbing. Schaffer and Eshelman reported for their plateau function that *"The crossover gene takes over and the population performance is about the same as with mutation alone but this performance level was reached much sooner."*
3. Matching Bits: This is a twenty bit multi-optima problem where each pair of adjacent bits of the same parity contribute 1 point to the score:

$$f(a) = 19 - \sum_{i=0}^{18} |a_i - a_{i+1}| \tag{27}$$

   There are thus two maxima at $(0, 0 \dots 0)$ and $(1, 1, \dots 1)$ which score 19. There are a large number of sub-optimal peaks: for example there are 34 sub-optima scoring 18 at Hamming distances from one of the optima of between two and ten. In addition to being multi-modal, this problem has high local epistasis of order two, and these blocks overlap.
4. "Trap" function: This is composed of eight contiguous subproblems, each being a four bit fully deceptive function given by Deb in [3]. The contribution of each subproblem j depends on the number of bits set to one:

$$f(a) = \sum_{i=0}^{7} \left(\prod_{j=i*4}^{i*4+3} a_j + 0.2 \left(1.0 - \prod_{j=i*4}^{i*4+3} \right) \sum_{j=i*4}^{i*4+3} a_j \right) \tag{28}$$

6. Results

6.1. Comparison with other operators

In addition to comparing the four variants above, experiments were also performed using 1-Point and uniform crossover (both with 100% probability of application). These two operators were chosen to represent and illustrate the extremes of positional bias. In each case the relevant algorithm was run with a population of 100 until the problem's optimum had been reached, or a fixed number (400,000) of evaluations had been done, and the results show the generation in which this occurred, averaged over twenty five independent runs. The same sets of seeds were used for each of the algorithms tested. Experiments were run across a spread of (bit flipping) mutation rates in order to investigate the sensitivity of the various operators to this parameter.

A summary of the results obtained showing the mean and standard deviation (in brackets) of number of evaluations taken at the optimum mutation setting for each operator, is given in Table 1. The figure in the third row for each algorithm is the mutation rate at which this performance was achieved. Figure 1 shows the sensitivity of the operators to mutation rates by plotting mean number of generations to reach the optimum against bit mutation rate for each problem. It should be noted that for each operator the optimal mutation rate is generally different. Equally, for each function, if the operators are ranked according to the performance at a single mutation rate (as is common practice) the rank orders obtained are somewhat dependent on the choice of mutation rate.

Inspection of these results shows that for the linearly separable OneMax, the 1 Point Crossover operator is considerably slower than the rest of the operators to find the optimum, across the spread of mutation rates tested. For the rest of the

Table 1. Generations (std. deviations) to find optimum at optimal mutation rate in % (bold)

Algorithm	One-max	Royal road	Matching ones	Trap
1 Point	2162.28	27693.64	5104.32	9861.32
	(372.0)	(10509.8)	(3653.2)	(6787.9)
	0.6	**1.25**	**7**	**3**
Uniform	1134.8	30005.56	9436.68	306219.56
	(211.8)	(13346.9)	(9999.0)	(125335.1)
	0.4	**0.75**	**6**	**3**
Lego	1261.24	28839.16	7223.76	73375.28
	(223.1)	(13853.6)	(5747.4)	(81273.3)
	0.08	**1.2**	**7**	**3.5**
Pop. v.1	1120.48	31163.04	13745.6	283650.8
	(270.7)	(10832.9)	(12504.9)	(132179.8)
	4	**1.25**	**6**	**3.5**
Pop. v.2	1218.08	33409.72	9825.08	248588.88
	(281.9)	(15812.2)	(10070.9)	(129354.0)
	0.4	**7.5**	**6**	**3.5**
Pop. v.3	996.16	26429.2	7964.76	320093.2
	(207.97)	(12359.36)	(5193.0)	(14026.470)
	0.4	**1**	**7**	**3.5**

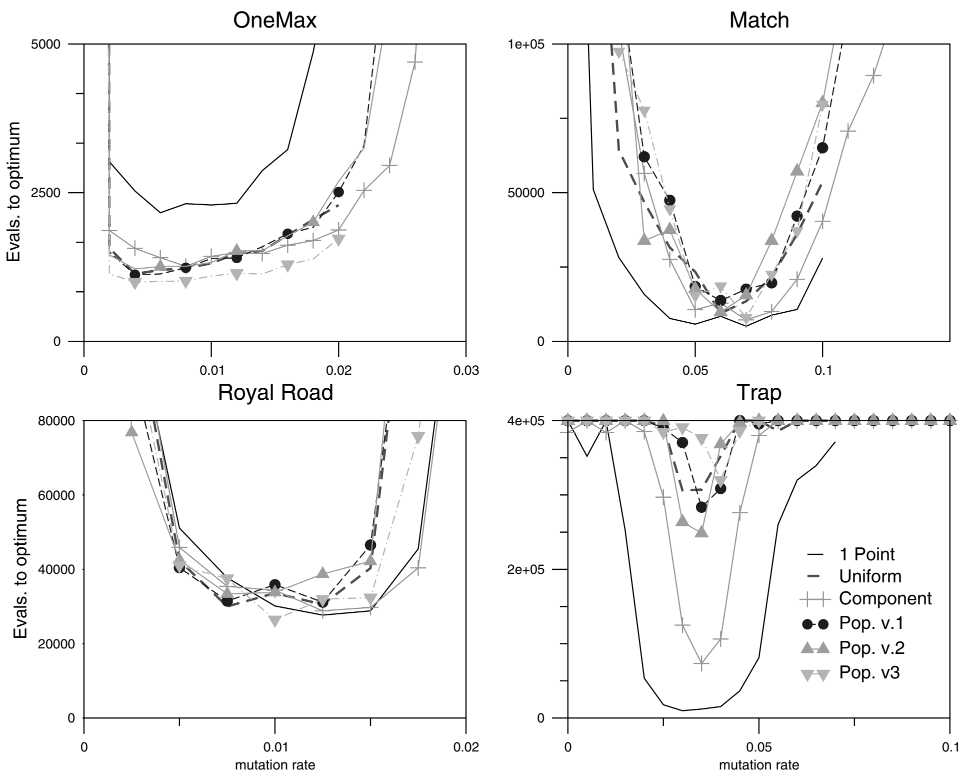

Figure 1. Mean number of iterations to solve problem vs. mutation rate.

problems, all of which display tight linkage which suits the positional bias, the best performance is shown by 1 Point Crossover, albeit with a marginal improvement over the component level, and Pop. v.3 adaptive algorithms on Royal Road and Matching Bits problems. As suggested by the standard deviations, the results for the different algorithms are often not statistically significantly different except in the case of OneMax (One Point performance worse) and the deceptive function, where Uniform Crossover does not solve the problem, and the Component level adaptive algorithm, and 1 Point Crossover are different to each other and faster than the other algorithms.

More pertinently, we can see that although the three population level algorithms give faster performance at their optimal mutation rates for OneMax, the Component level algorithm is more robust with respect to mutation rates, as it is for the rest of the problems, where it gives superior performance. Comparing the three population level algorithms shows that the multiparental population level algorithm is able to sustain higher mutation rates than the two parent versions, but does not perform significantly better. Overall these results suggest that there is a benefit to be had from the component level adaptation, especially on the deceptive problems.

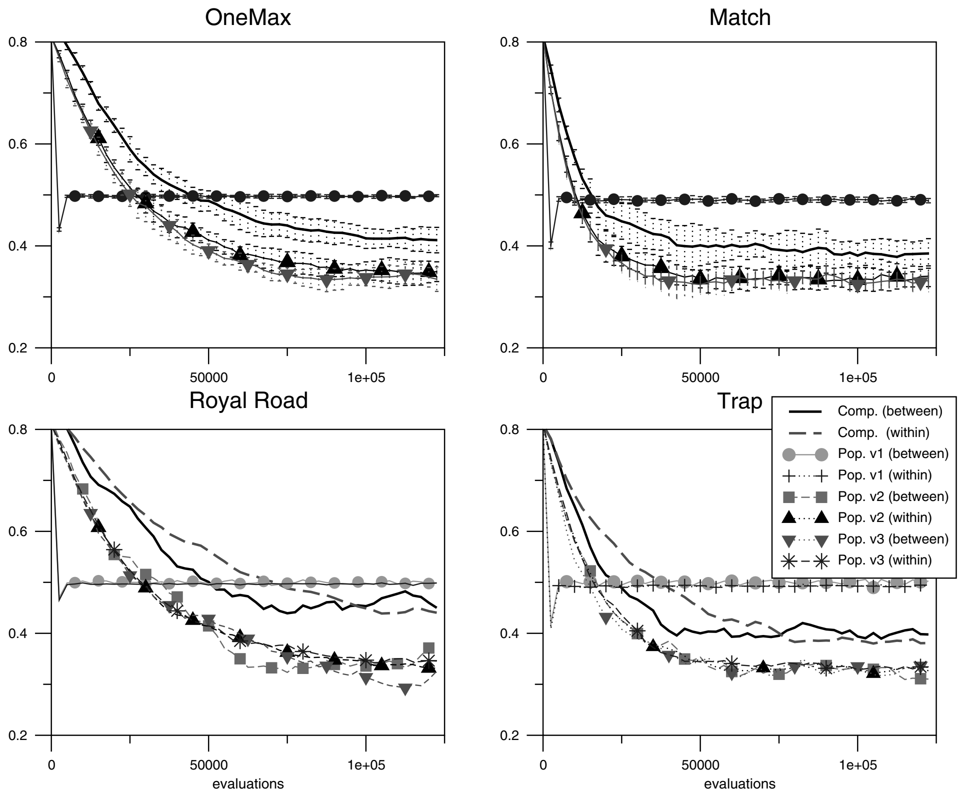

Figure 2. Analysis of evolved linkage as a function of time.

6.2. Analysis of operation—Strategy adaptation

In order to examine the cause of these observations, the mean linkage rate in the population was measured at each locus, as a function of time. Figure 2 shows the first 12500 iterations of the evolution of this linkage averaged over ten runs for each algorithm on each problem using a population size of 250. The mutation rates used were those found to give the best results above for each combination of algorithm and problem.

For the OneMax and Match functions, the figures show the values averaged over the whole string, and the error bars represent one standard deviation. For the Royal Road and Trap problem, the values have been grouped into two classes, representing links within and between sub-problems, and the error bars have been omitted for the sake of clarity. In the following we shall use the term *distinguishable* as shorthand for saying that for significant portions of the evolutionary timescale shown, the distance between two curves is greater than the sum of their standard deviations at that point.

From these figures we can make the following observations:

— The algorithms can be grouped into three classes, namely component level, probabilistic population level (Pop. v.1) and Boolean population level (Pop. v.2 and Pop. v.3). All three classes show different behavior. Where the error bars are not shown, there is still a clear difference between the three algorithms, that is to say that any pair of corresponding curves are distinguishable. However the two population level algorithms which use the normal LEGO method of determining linkage do not show different behavior to each other, despite their different arity.
— The first population level algorithm, which uses a stochastic interpretation of the linkage arrays, very quickly evolves to a steady state of nearly 50% mean linkage on every problem, i.e., there is no evidence of learning. This arises from the averaging effect of the adaptation method used.
— The second and third population level algorithms, which rely heavily on the basic *LEGO* algorithm, but divorce the linkage learning from the allele learning, exhibit very similar patterns of behavior on all the problems, and evolve to the same linkage rate. Both algorithms display brief periods where there is different linkage between and within blocks on the Royal Road problems, but these are brief and there is no significant difference between the behavior between and within blocks on the trap function.
— In contrast, the component level algorithm shows far more distinct patterns of evolving linkage. In particular, it shows a pattern of increasing linkage as the epistasis in the problem increases, as can be easily seen by comparing the curves for the Match and Royal Road problems. The OneMax problem is solved so quickly that the behavior observed is simply an evolutionary drift towards the steady state linkage value. The fact that this occurs slower than the decrease in mean linkage for the Match problem suggests that in the latter case there is actually a selective pressure against linkage on that problem. Moreover, for the Royal Road and Trap problems, the curves for linkage between and within blocks are distinguishable over much of the runs, only converging once the problem is solved and the population converges. This demonstrates the ability of even this unsophisticated algorithm to learn linkage information.

6.3. Scaleability

In order to examine the effect of problem size on the performance of the different variants of the algorithm, a series of experiments was run using increasing numbers of concatenated sub-problems of Deb's 4-bit deceptive function, giving problem lengths of between 20 and 120 bits (increasing in steps of 20). In each case a population size of 1000 was used, and the mutation rate was set to $1/l$. Experiments with smaller population sizes (not reported here for reasons of space) showed that the population based methods required this large population size in order to solve the larger problems reliably within the time allowed for each run. However, the pattern of results for the smaller populations was the same as those reported here.

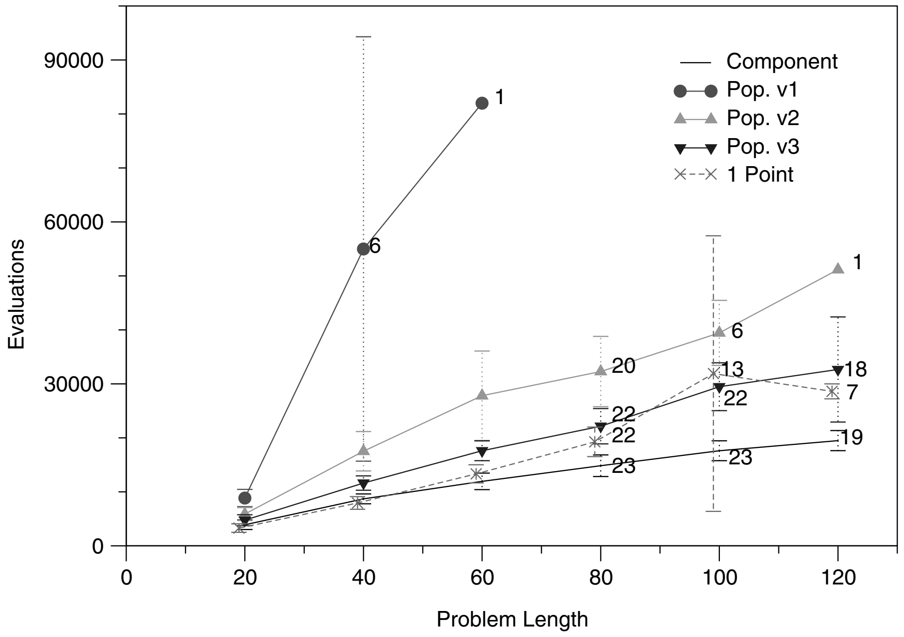

Figure 3. Mean and standard deviations of time taken to solve problems as a function of length. Where not all runs found the global optimum, the number of successful runs (used to calculate statistics) is shown.

Each of the four variants, along with One Point Crossover was given 25 runs of 1 million evaluations on each problem size, and the time taken to find the global optimum noted. Figure 3 shows the mean times taken by the successful runs, with error bars showing one standard deviation. Alongside each point is the number of runs in which the optimum was found (omitted where all runs were successful for the sake of clarity.

From these results we can make the following observations:

— The first population level algorithm displays poor scaleability, and even with these large population sizes is unable to solve problems greater than length 60.
— There is a clear difference between the performance of the second two population level algorithms, with the multi-parent version performing more reliably, and faster, for the longer problems.
— The results for 1-Point crossover show that it is fastest for the smaller problems, as was seen above. However this advantage disappears for longer problems, and it solves the longer problems less frequently than either the third population level variant or the component level algorithm.
— The component level algorithm shows the best performance, both in terms of reliability and speed of locating the global optima for the longer problems. This is despite the fact that the problem representation should favour 1-Point crossover, and demonstrates that the overhead of correctly identifying the building blocks is more than offset by the advantages of preserving and mixing them.

7. Conclusions

The aim of this paper was to offer a perspective on linkage learning from the point of view Adaptive Evolutionary algorithms, and to see if this could give useful insights into the topic of Gene Linkage. From this point of view, it becomes apparent that the notion of *Appropriate Levels of Learning* is one which has definite relevance to the field.[12] Further consideration into the nature of the information used by the learning algorithms suggests that if the purpose of linkage learning is to provide information which can be used offline, then a "learn first, optimise second" such as that taken by the *Messy GA* may be preferable. It also suggests that there may be an inherent weakness in schemes which attempt to learn linkage on-line at a population level, arising from the problem of convergence of allele values, unless mechanisms are specifically in place to combat this.

Distinctions were drawn between linkage learning via adaptation of recombination operators, and the implicit learning of linkage in Distribution Estimation algorithms, although it was noted that even in the latter, it is not possible to separate the process of learning linkage groups from that of learning their contents.

A Boolean definition of gene linkage was introduced, within which the major families of static crossover operators can be modelled and generalised, and it was shown how various Adaptive Linkage Learning algorithms can be easily understood and contrasted using the notion of "Linkage Arrays." It was also shown how consideration of these arrays can lead naturally to the specification of operators which explicitly manipulate them. One such operator, *LEGO* was adapted to provide a means of exploring the effects of component vs. population based linkage learning.

The empirical results, on a test suite designed to be "learning-easy" illustrate that there is a clear difference between the different levels of learning. Not only does the component based scheme provide faster time to discovery of the optimum on all but the simplest problem, it also displays greater robustness to changes in the mutation rate. When the patterns of evolved linkage between and within blocks are examined, there is a noticeable difference between the algorithms. The component level algorithm is able to distinguish the two cases, and evolves linkage patterns which will lead to higher mixing between blocks than inside them. This is reflected in far superior optimisation performance. In contrast, the population level variants do not appear to "learn" the same information, even though two of them used exactly the same method of inheriting linkage arrays, but simply separate this from the allele values inherited.

When the problem sizes were increased these results became more apparent. The difference in results between the algorithms demonstrate that although some of the advantage came from the multi-parental nature of the component level algorithm, this was not as important as the ability to identify, preserve and mix good combinations of allele values. The component level algorithm displayed apparently linear behavior, and outperformed 1-Point Crossover, despite the problem representation being tailored for the high positional bias of the latter. In other words the component based scheme is able to learn and exploit the linkage structure of the problem.

Because the problem representations were explicitly designed to suit the simplified adjacent linkage model it is not reasonable to claim that the success of the simple component level algorithm reported here will work under all conditions, and indeed this was not the point of this paper. However the population level models failed to learn appropriate linkage groups without simultaneously considering the contents of those groups, even in these most favourable of circumstances. This provides empirical evidence to back up the earlier conceptual arguments in favour of trying to learn linkage at an individual or component level rather than at the population level.

Having demonstrated these results using a self-adaptive scheme, the next phase of work will concern extending the mechanism to deal with non-adjacent linkage, and higher order problems, as well as performing comparisons using learning mechanisms which do not rely on self-adaptation, e.g., GEMGA vs. LIMD. Finally, it should be noted that this is an area in which rapid advances are being made (particularly in the related field of Distribution Estimation Algorithms), and these results should be seen as offering a different perspective to be considered in the design of algorithms, rather than offering hard and fast rules.

Notes

1. Simply stated, the degree of linkage between two genes may be considered as the probability that they are kept together during recombination.
2. A chromosome is said to exhibit epistatic interactions if the fitness contribution from one or more genes depends not only on the allele value at that locus, but also on the values at other loci.
3. This assumption that "good" learning strategies will tend to be associated with high quality solutions lies behind a large body of successful research into Self Adaptivity.
4. This model first appeared in [42].
5. Note that this does not refer to DEAs.
6. Personal communication, June, 1996, note these problems have been to some extent mitigated in later versions.
7. Where l is the length of the problem encoding.
8. This is referred to as the *arity* of the operator.
9. In this work binary deterministic tournaments are used, although the use of any operator is possible.
10. It is guaranteed to be possible to complete an offspring since succeeding blocks can be chosen from the same parent.
11. In practice more compact representations are used.
12. It also becomes apparent that a number of operators which adaptively learn Gene Linkage tend not to be recognised as such.

References

1. P. J. Angeline, "Adaptive and self-adaptive evolutionary computations," Computational Intelligence, pp. 152–161, 1995.
2. T. Back, "Self adaptation in genetic algorithms," in Towards a Practice on Autonomous Systems: Proc. First European Conf. Artificial Life, F. Varela and P. Bourgine (eds.), 1992, pp. 263–271.
3. T. Back, D. Fogel, and Z. Michalwicz, Handbook of Evolutionary Computation, Oxford University Press, 1997, vol. 1.

4. S. Baluja and S. Davies, "Using optimal dependency-trees for combinatorial optimisation: Learning the structure of the search space," in Proc. 1997 Internat. Conf. Machine Learning, Fisher (ed.), 1997.
5. P. Bosman and D. Thierens, "Linkage information processing in distribution estimation algorithms," in Proc. Genetic and Evolutionary Computation Conf., W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honovar, M. Jakiela, and R. Smith (eds.), 1999, pp. 60–67.
6. K. De Jong, "An analysis of the behaviour of a class of genetic adaptive systems," Ph.D. thesis, University of Michigan, 1975.
7. J. S. DeBonet, C. Isbell, and P. Viola, "MIMIC: Finding optima by estimating probability densities," Advances in Neural Information Processing Systems, vol. 9, pp. 424–431, 1997.
8. A. E. Eiben, P. Raue, and Z. Ruttkay, "Genetic algorithms with multi-parent recombination," in Proc. Third Conf. Parallel Problem Solving from Nature, Y. Davidor (ed.), 1994, pp. 78–87.
9. A. E. Eiben, C. van Kemenade, and J. Kok, "Orgy in the computer: Multi-parent reproduction in genetic algorithms," in Advances in Artifical Life: Proc. Third European Conf. Artifical Life. F. Moran, A. Moreno, J. Morelo, and P. Chacon (eds.), 1995, pp. 934–945.
10. R. Etxeberria and P. Larranaga, "Global optimisation using Bayesian networks," in Proc. Second Symposium on Artificial Intell. (CIMAF-99), A. O. Rodriguez, M. S. Ortiz, and R. Hermida (eds.), 1999, pp. 332–339.
11. S. Forrest and M. Mitchell, "Relative building block fitness and the building block hypothesis," in Foundations of Genetic Algorithms 2, L. D. Whitley (ed.), 1992, pp. 109–126.
12. D. Goldberg, Genetic Algorithms in Search, Optimisation and Machine Learning, Addison Wesley, 1989.
13. D. E. Goldberg, "Optimal initial population size for binary-coded genetic algorithms," Tcga Report no. 85001, University of Alabama, Tuscaloosa, US, 1985.
14. D. E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: Motivation, analysis, and first results," Complex Systems, vol. 3, no. 5, pp. 493–530, 1989.
15. J. J. Grefenstette, "Optimisation of control parameters for genetic algorithms," Transaction on Systems, Man and Cybernetics, vol. 16, no. 1, pp. 122–128, 1986.
16. G. Harik, and D. Goldberg, "Learning linkage," Technical Report IlliGAL 96006, Illinois Genetic Algorithms Laboratory, University of Illinois.
17. H. Kargupta, "The gene expression messy genetic algorithm," in Proc. Third IEEE International Conf. Evolutionary Computing, 1996, pp. 814–819.
18. H. Kargupta, "SEARCH, computational processes in evolution, and preliminary development of the gene expression messy genetic algorithm," Journal of Complex Systems, vol. 11, no. 4, pp. 233–287, 1999.
19. H. Kargupta and S. Bandyopadhyay, "A perspective on the foundation and evolution of the linkage learning genetic algorithms," J. Computer Methods in Applied Mechanics and Engineering, vol. 2186, pp. 266–294, 2000.
20. J. R. Levenick, "Inserting introns improves genetic algorithm success rate: Taking a cue from biology," in Proc. Fourth Internat. Conf. Genetic Algorithms, R. K. Belew and L. B. Booker (eds.), San Mateo, CA, 1991, pp. 123–127.
21. J. R. Levenick, "Megabits: Generic endogenous crossover control," in Proc. Sixth Internat. Conf. Genetic Algorithms, L. J. Eshelman (ed.), pp. 88–95.
22. Muhlenbein and Mahnig, "Convergence theory and applications of the factorized distribution algorithm," JCIT: J. Computing and Information Technology, vol. 7, 1999.
23. H. Muhlenbein, "The equation for the response to selection and its use for prediction," Evolutionary Computation, vol. 5, no. 3, pp. 303–346, 1998.
24. H. Muhlenbein and G. Paas, "From recombination of genes to the estimation of distributions I. Binary parameters," in Proc. Fourth Conf. Parallel Problem Solving from Nature, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (eds.), 1996, pp. 188–197.
25. M. Munetomo and D. E. Goldberg, "Linkage Identification by Non-monotonicity Detection for Overlapping Functions," Evolutionary Computation, vol. 7, no. 4, pp. 377–398, 1999.
26. M. Pelikan, D. E. Goldberg, and E. Cant-Paz, "BOA: The Bayesian optimization algorithm," in Proc. Genetic and Evolutionary Computation Conf., W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honovar, M. Jakiela, and R. Smith (eds.), 1999a, vol. 1, pp. 525–532.

27. M. Pelikan, D. E. Goldberg, and F. Lobo, "A survey of optimization by building and using probabilistic models," Technical Report, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 1999b.
28. M. Pelikan and H. Muehlenbein, "The bivariate marginal distribution algorithm," in Advances in Soft Computing—Engineering Design and Manufacturing, R. Roy, T. Furuhashi, and P. Chawdhry (eds.), 1999, pp. 521–535.
29. I. Rechenberg, Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Stuttgart: Frommann-Holzboog, 1973.
30. A. Salman, K. Mehrota, and C. Mohan, "Linkage crossover for genetic algorithms," in Proc. Genetic and Evolutionary Computation Conf., W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honovar, M. Jakiela, and R. Smith (eds.), 1999, pp. 564–571.
31. J. Schaffer and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," in Proc. Second Internat. Conf. Genetic Algorithms, J. J. Grefenstette (ed.), 1987, pp. 36–40.
32. J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimisation," in Proc. Third Internat. Conf. Genetic Algorithms, J. D. Schaffer (ed.), 1989, pp. 51–60.
33. J. D. Schaffer and L. J. Eshelman, "On crossover as an evolutionarily viable strategy," in Proc. Fourth Internat. Conf. Genetic Algorithms, R. Belew and L. Booker (eds.), 1991, pp. 61–68.
34. H. P. Schwefel, Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie, Basel/Stuttgart: Birkhaeuser, 1977, vol. 26 of ISR.
35. H. P. Schwefel, Numerical Optimisation of Computer Models, John Wiley and Sons: New York, 1981.
36. M. Sebag and M. Schoenauer, "Controlling crossover through inductive learning," in Proc. Third Conf. Parallel Problem Solving from Nature, Y. Davidor (ed.), 1994, pp. 209–218.
37. J. Smith and T. Fogarty, "Adaptively parameterised evolutionary systems: Self adaptive recombination and mutation in a genetic algorithm," in Proc. Fourth Conf. Parallel Problem Solving from Nature, Voigt, Ebeling, Rechenberg, and Schwefel (eds.), 1996a, pp. 441–450.
38. J. Smith and T. Fogarty, "Recombination strategy adaptation via evolution of gene linkage," in Proc. Third IEEE Internat. Conf. Evolutionary Computing, 1996b, pp. 826–831.
39. J. Smith and T. Fogarty, "Self adaptation of mutation rates in a steady state genetic algorithm," in Proc. Third IEEE Internat. Conf. Evolutionary Computing, 1996c, pp. 318–323.
40. J. Smith and T. Fogarty, "Operator and parameter adaptation in genetic algorithms," Soft Computing, vol. 1, no. 2, pp. 81–87, 1997.
41. J. Smith and T. C. Fogarty, "An adaptive poly-parental recombination strategy," in Evolutionary Computing 2, T. C. Fogarty (ed.), 1995, pp. 48–61.
42. J. E. Smith, "Self adaptation in evolutionary algorithms," Ph.D. thesis, University of the West of England, 1998.
43. W. M. Spears and K. A. De Jong, "On the virtues of parameterized uniform crossover," in Proc. Fourth Internat. Conf. Genetic Algorithms, R. Belew and L. Booker (eds.), 1991, pp. 230–237.
44. D. Thierens and D. Goldberg, "Mixing in genetic algorithms," in Proc. Fifth Internat. Conf. Genetic Algorithms, S. Forrest (ed.), 1993, pp. 38–45.
45. C. van Kemenade, "Explicit filtering of building blocks for genetic algorithms," in Proc. Fourth Conf. Parallel Problem Solving from Nature, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (eds.), 1996, pp. 494–503.
46. A. White and F. Oppacher, "Adaptive crossover using automata," in Proc. Third Conf. Parallel Problem Solving from Nature, Y. Davidor (ed.), 1994, pp. 229–238.