Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms

David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, & Georges Harik
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801

IlliGAL Report No. 93004
February 1993

Illinois Genetic Algorithms Laboratory
Department of General Engineering
University of Illinois at Urbana-Champaign
117 Transportation Building
104 South Mathews Avenue
Urbana, IL 61801

Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms

David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, & Georges Harik
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign

Abstract

Researchers have long sought genetic algorithms (GAs) that can solve difficult search, optimization, and machine learning problems quickly. Despite years of work on simple GAs and their variants it is still unknown how difficult a problem simple GAs can solve, how quickly they can solve it, and with what reliability. More radical design departures than these have been taken, however, and the messy GA (mGA) approach has attempted to solve problems of bounded difficulty quickly and reliably by taking the notion of building-block *linkage* quite seriously. Early efforts were apparently successful in achieving polynomial convergence on some difficult problems, but the initialization bottleneck that required a large initial population was thought to be the primary obstacle to faster mGA performance. This paper replaces the partially enumerative initialization and selective primordial phase of the original messy GA with *probabilistically complete initialization* and a primordial phase that performs *building-block filtering* via selection and random gene deletion. In this way, the fast mGA is able to evaluate the best building blocks from modestly sized populations of longer strings, thereafter cutting down the string length by throwing off the genes of lesser importance. Design calculations are performed for population sizing, selection-deletion timing, and genic thresholding. On problems of bounded difficulty, ranging from 30-bits to 150-bits, the fast mGA finds global optima reliably in a time that both theoretically and empirically grows no more quickly than a subquadratic function of the number of decision variables. The paper outlines the key remaining challenges and suggests extension of the technique to other-than-binary structures.

1 Introduction

Genetic algorithms (GAs) are receiving increased attention in difficult search, optimization, and machine learning applications, but despite this increased interest, genetic algorithms still lack an integrated theory of operation that predicts how difficult a problem GAs can solve, how long it takes to solve them, and with what probability and how close to a global solution the GA can be expected to come. On the one hand, it seems odd that such a theory still eludes us. Researchers have spent considerable effort trying to understand GAs, and important bits and pieces of such a theory have been around since the late 1960s and early 1970s. On the other hand, the simplest GAs are nonlinear, stochastic, highly dimensional algorithms operating on problems of infinite variety. Although recognizing these facts of GA life may not reduce our impatience, it may help increase our understanding of why the lock to fast selectorecombinative processing has not yet been completely picked.

Despite the lack of a completely operational theory, progress has been made in obtaining portions of one. Holland's (1975) pioneering theoretical results have led to more recent efforts that concentrate on effective building-block processing through competent supply, growth, exchange, and decision making; these have led to the threshold of a theory of the limits of simple GA processing (Goldberg, Deb, & Clark , 1992). It is unclear how far these ideas can take us, but recent efforts have made it clear that the process is dependent on a better understanding of building-block mixing (Goldberg, Deb, & Thierens, 1993). Although many have suggested that uniform crosses and other kinds of linkage-independent crossover operators can be adjusted to ensure building-block growth, the issue ultimately hinges on whether those surviving building blocks can actually be exchanged in a timely and stable fashion.

For several years another tack has been taken toward the design of well-grounded GAs. Work on so-called messy genetic algorithms (mGAs) started in 1988 and was first published somewhat later (Goldberg, Korb, &

Deb, 1989). That work took Holland's calls for *tight linkage* (Holland, 1975) quite seriously and attempted to get the linkage right prior to subsequent genetic processing. Those efforts were fairly successful, apparently achieving global solutions in polynomial times in a sense similar to that of the *probably almost correct* (PAC) algorithms of computational learning theory (Deb, 1991; Goldberg, Deb, & Korb, 1990, 1991). In some sense these efforts were both good news and bad news. The good news was that hard problems could be solved in polynomial time, and this offered promise that perhaps all problems of bounded difficulty could be solved as quickly. The bad news was that there appeared to be a mismatch between the amount of processing required for different phases of the original mGA. Although the latter phases were shown to require $O(\ell \log \ell)$ function evaluations, where ℓ is the number of boolean decision variables in the problem, the initialization of the mGA required $O(\ell^k)$ function evaluations, where k is an exponent that goes up as the problem becomes increasingly difficult. This mismatch indicated that design efforts had not yet sharpened the mGA pencil sufficiently.

In this paper, we consider one way to overcome this *initialization bottleneck*. In some ways we harken back to the earliest days of messy GA investigation, when strings longer than the building-block length were used to try to get multiple copies of needed substructures in a more modestly sized population. Those original experiments were unsuccessful, but in the current incarnation, we try to use such *probabilistically complete initialization* with a method of *building-block filtering* to obtain well-tested, tight building blocks from random initial populations of modest size. Together these two techniques allow us to obtain a collection of good building blocks that can then be fed to the normal apparatus of the messy GA, permitting successful juxtaposition and the discovery of global optima with high probability with modest computational effort.

We start by briefly reviewing the mechanics of the original messy GA. We continue by describing the mechanics and analysis of probabilistically complete initialization and building-block filtering. The modified GA is then applied to a number of difficult test functions, starting with a previously used order-three function of modest size and going on up to 150-bit problems of order-five deception. Theoretical predictions and empirical results suggest that the procedure may be as good as subquadratic in a probably almost-correct sense. The paper concludes by outlining the investigation necessary to determine whether these results carry over to arbitrary problems of bounded difficulty.

2 A Brief Review of Messy GAs

In this section, we briefly review messy GAs. Readers interested in more detail should consult other sources (Deb, 1991; Deb & Goldberg, 1991; Goldberg, Deb, & Korb, 1990, 1991; Goldberg & Kerzic, 1990; Goldberg, Korb, & Deb, 1989). Specifically, the following topics are reviewed:

1. messy codes;
2. handling over- and underspecification;
3. mGA inner and outer loops;
4. basic mGA theory;
5. time-complexity estimates.

In the remainder of this section, each of these is discussed in more detail.

2.1 Messy codes

Messy GAs are messy because they allow variable-length strings that may be under- or overspecified with respect to the problem being solved. Although many of the theoretical results are for binary strings, messy floating-point GAs have been suggested (Goldberg, Korb, & Deb, 1989) and tried (Deb, 1991), and other optimization, classification, and machine-learning structures have been suggested. That we emphasize bit strings is not a case of being radical bit fiddlers for its own sake; rather we are radical bit fiddlers in the name of understanding all GAs and in the hope of developing well-grounded algorithms that converge on problems of bounded difficulty quickly and reliably.

Messy GAs relax the fixed-locus assumption of most simple GAs. This is accomplished first by defining a messy gene as an ordered pair that identifies the gene by its name and value and then by defining a messy

chromosome as a collection of messy genes. For example, imagine a problem of length $\ell = 3$ and consider the messy chromosome as follows: `((1 0) (2 1) (1 1))`. In this string, the first entry is gene one with value one, the second entry is gene two with value one, and the third entry is gene one with value one. Since both name and value are specified, arbitrary building blocks may achieve tight *linkage* simply by having or rearranging the constituent alleles of the building block in close proximity to one another. Taking linkage theory seriously was one of the primary design guidelines adopted throughout the mGA development process. This was important for two reasons. With tight building blocks, there is a low probability of good ones being destroyed by crossover and other genetic operators. Additionally, the existence of tight linkage makes it easy to *exchange* different building blocks (Goldberg, Deb, & Thierens, 1993), thereby getting the positive recombination of the best substructures that goes into the creation of optima or near-optima.

By defining a messy chromosome as a collection of messy genes, notice that we have not required all genes to be present, nor have we precluded the possibility of multiple, possibly contradictory, genes. Returning to the previous example, `((1 0) (2 1) (1 1))`, notice that gene one is specified in two different places, and notice that the specified values are contradictory. We must have some way of overcoming this problem of *overspecification*, a matter to be discussed in a moment. Moreover, if this is truly a 3-bit problem then the string is *underspecified*, because there is no mention of gene three. Handling over- and underspecification were challenges that were successfully addressed in the original study, and we briefly consider the computations adopted therein.

2.2 Over- and underspecification: Expression and competitive templates

In the original study and all subsequent work, overspecification has been handled through a *gene expression* operator that employs a first-come-first-served rule on a left-to-right scan. For example, the string `((1 0) (2 1) (1 1))` would be expressed as `((1 0) (2 1))`, because the second instance of gene one would be dropped on the left-to-right scan. First-come-first-served expression was adopted instead of alternative voting or averaging rules, because the possibility of a proliferation of low-order, incorrect building blocks that can occur early as a result of population sampling and selection on misleading problems makes these alternative schemes unreliable. It also turns out that first-come-first-served expression sets a lower bound of 0.5 on the possible crossover disruption, which in turn ensures building-block growth when appropriate selection schemes are adopted.

In some problems, underspecification is not a problem because any structure of any size can be interpreted naturally. In parameter optimization problems over some fixed number of parameters all decision variables must be supplied to the objective function to obtain a value. In messy GAs the unspecified bits of a messy chromosome are filled in by a *competitive template*, a string that is locally optimal to the previous level. In a moment we will discuss the performance of level-wise messy GAs, but the idea here is that a locally optimal structure is obtained at one level and substrings that overlay the locally optima structure and obtain function values higher than that of the template must by definition contain building blocks at the next level or higher. In this way, good evaluation can be obtained of partial strings containing building blocks, and with this reliable evaluation in hand the building blocks may be recombined with other reliably evaluated building blocks to form better structures.

2.3 Inner loop, outer loop

The basic inner loop of the original mGA has three phases:

1. initialization;
2. primordial phase;
3. juxtapositional phase.

This inner loop may be repeated at each building-block level, thereby ensuring that the competitive template at the next level is sufficiently optimal to guarantee a good objective function value signal for building blocks at the next level.

Initialization. Initialization in the original work was performed by creating a population with a single copy of all substrings of length k. By doing this, we know that all building blocks of the desired length are present,

and if the genetic processing respects the good building blocks and recombines them, they can be expected to grow and form good solutions. The downside of having all these building blocks present, is that each must be evaluated to determine which are the best. This requires a population size, $n = 2^k \binom{\ell}{k}$, because in a problem of ℓ, there are a total of $\binom{\ell}{k}$ gene combinations of size k, and for each gene combination there are a total of 2^k different allele combinations.

Primordial phase. Having to evaluate each of these substrings is bad enough, but if population after population requires evaluation of this many strings, the mGA becomes a very expensive affair, indeed. In turns out that because the initial evaluation is reliable, it is not performed repeatedly during the so-called *primordial phase*. In this phase, selection alone is run to dope the population with a high proportion of the best building blocks. In the primordial phase, it is also common to adjust the population size to be appropriate for the processing of the recombinative or *juxtapositional phase*

Juxtapositional phase. After the population is rich in copies of the best building blocks, processing proceeds that more closely resembles that of simple GAs. During the juxtapositional phase, selection is used together with a cut-and-splice operator. Although various mutation operators have been defined, mGA tests have never used them. This has been done to put the algorithm to the most stringent test of its ability to exploit initial diversity properly.

The cut-and-splice operator replaces ordinary crossover operators, and the coordination of the operator is discussed elsewhere (Goldberg, Korb, & Deb, 1989); sample code is available (Deb & Goldberg, 1991; Goldberg & Kerzic, 1989). Because the probability of cutting a parent string goes up as the string gets longer, cut and splice have two limiting types of behavior. Early on when strings are short, the chance of a cut is low, but splicing proceeds at normal rates. Therefore, early in the game, cut-and-splice behaves like splice alone, roughly doubling string lengths at each invocation. Later in a run when string lengths are long, parent strings get cut with near certainty, and the combination of cut-and-splice acts something like the one-point crossover operator of simple GAs.

Level-wise processing. The initial tests of the mGA were performed at a particular level, with the worst possible competitive template, but the need for a competitive template that is optimal with respect to the previous level has always recommended the idea of level-wise processing. Starting at level $k = 1$ implies the need for an order-zero optimal template—any random string. If the mGA does its job, the resulting structures should be optimal to order-one and can be used as competitive templates for $k = 2$ processing, and so on. Deciding when to stop is not easy, and unfortunately global solutions do not come with nametags that say "Hello, I am global." Nonetheless, the mGA can be run level by level until a a good-enough solution is obtained or the algorithm may be terminated after some specified number of levels have passed with no improvement.

2.4 Why do mGAs work?

Like all GA theory, mGA theory of operation is something of a patchquilt, but this is not an apology. As the invention of the airplane and other complex systems has shown, one must divide hard design problems into quasi-separate subproblems (Bradshaw & Lienert, 1990; Goldberg, in press) and work relentlessly and relatively independently on each subproblem. This is the approach that has been adopted in mGA work, and the following subproblems have been tackled:

1. Obtain and evaluate tight building blocks.
2. Increase proportions of the best building blocks.
3. Make good decisions among building blocks.
4. Exchange building blocks well.
5. Test against bounding, hard problems so results transfer to a large class of easier problems.

We outline the basic argument, and although the discussion will not yet satisfy hardcore theorem-provers, we believe that the argument is tight and leads to proofs similar to the probably almost-correct (PAC) results of computational learning theory.

We have already seen how mGAs obtain tight building blocks through partial enumeration, and we have seen how they are evaluated by overlaying them on top of a competitive template. Ensuring that proportions of the best building blocks increase is a matter of satisfying an mGA-appropriate schema theorem. During the juxtapositional phase, binary tournament selection is used, meaning that the best strings get something like two copies. First-come-first-served expression ensures that currently expressed building blocks continue to survive and be expressed with a probability of roughly 0.5. Thus, calculating a growth factor, we recognize that good building blocks get something like $2[1 - 0.5] = 1$ times their original numbers.

In order that good decisions be made during the juxtapositional phase requires that decisions be made well in the presence of noise. This question has been addressed elsewhere (Goldberg, Deb, & Clark, 1992), and it turns out that if populations are sized using statistical decision theory, the actual best structures will be preferred on average over the course of a run.

The question of exchanging building blocks properly has received surprisingly little attention. A recent study (Goldberg, Deb, & Thierens, 1993) calculated and verified a *control map* for simple GAs processing easy problems. These results are germane here, because mGAs that use tightly linked building blocks on hard problems are analogous to simple GAs processing alleles on easy problems. The control map shows that typical mGA control values (binary tournament selection with $s = 2$ copies to the best, and splice probability $p_s = 1$) ensures reliably accurate mixing in the mGA.

Together, these results suggest that the mGA should converge to good answers in hard problems as long as we have some idea what hard is. mGAs have general been tested with problems by using sums of fully deceptive subfunctions. Some (Grefenstette, in press; Mitchell & Forrest, 1991) have explicitly asked questions about the relevance of deception and implicitly (sometimes explicitly) cast doubts on studies that use deception in one way or another. A paper devoted to doing something positive is not the place to examine the largely negative discussions of others; however, despite these criticisms, we believe that our methodology has been sound, and that mGA convergence and speed will hold up under other types of problem difficulty. We know now and have known for a long time that there is indeed more to GA hardness than deception. Isolation, misleadingness, building-block crosstalk, noise, and massive multimodality can all play a role in throwing GAs off the track. This would seem to argue that our focus on deception has been misguided, except that deception encompasses two of the trickiest GA roadblocks—isolation and misleadingness—and other of our efforts have tackled the other difficulties in isolation. As was mentioned before, the lesson from the invention of the airplane and other complex systems is that difficult systems must be designed by breaking the big design problem up into little, more-manageable, quasi-independent subproblems. This is what we have done.

The critics criticize this and call for more elegant, highfalutin mathematics and models as the way to go; we are not at all sure how the critics know this, because we have yet been able to find their papers on these subjects. Those of us who have tried more complex analyses know that the tools—things such as difference equations, diffusion models, Markov chains, information theory, and the like—are powerful, but cumbersome, and in design, cumbersome tools can be the kiss of death. Instead we have used our intuition, we have used our imagination, we have used dimensional analysis, we have used careful bounding experiments, and we have flexibly used bailing wire and chewing gum to tie together a series of simple models that are easy to apply and have enough predictive power to guide the design process. This permits us more time to think about integrating mechanism, and it gives us enough information to get the GA's parameters in the right ballpark without the problem-specific manipulation recommended elsewhere (Grefenstette, 1986). The latest fruits of the approach advocated here will be presented in a later section, and we will soon see how this methodology has apparently achieved accurate, subquadratic results on hard problems. Whether this is an important achievement is for others to judge, but we know we would not have gotten to this threshold by prematurely calling for rigor or elegance. Ultimately, our disagreement with these critics is a methodological dispute, and if we were engaged in science—if we were engaged in *description*—perhaps we could longer tolerate flawed methodologies (and ideologies and theologies) to proliferate as they have done in the complex sciences such as economics, psychology, political science, and the like. But since we are engaged in the design or engineering of complex systems—since we are engaged in complex systems *prescription*—it seems to us that competing methodologies and their advocates must be put to the acid test of practice. Therefore, we challenge our critics to show similar results using their methodologies, or we ask them to join us in our patchquilt approach, or we ask them to remain silent for a time while others do the heavy lifting that needs to be done. The moment will come for their theorem proving and elegance, but that time is not now. Now is the time to invent, experiment with, and use so-called "harmful" models like the schema theorem, deception, rough population sizing, and

Table 1: Complexity Estimates for the Original mGA (Goldberg, Deb, & Korb, 1990)

Phase	Serial
Initialization	$O(\ell^k)$
Primordial	0
Juxtapositional	$O(\ell \log \ell)$
Overall mGA	$O(\ell^k)$

mixing control maps to design better GAs. In our view, denigrating such useful models and replacing them with the mysterious vaporware of unspecified and uninvestigated mathematics and models is the truly harmful act here, an act that both misunderstands the difficulty of the design task before us and is ignorant of how other complex systems have been successfuly designed.

2.5 mGA time complexity

One trademark of mGA work has been the explicit concern for algorithm time complexity. The serial time complexity estimate of another study (Goldberg, Deb, & Korb, 1990) is presented in table 1. Looking at the tabulation it is clear that the computation is dominated by the initialization phase. Moreover, the difference between initialization at $O(\ell^k)$ and normal juxtapositional processing of $O(\ell \log \ell)$ is so significant, we have long wondered whether there isn't a way to get the initialization more in line with the requirements of the rest of the algorithm. In the next section, we describe a modification to the initialization and primordial phases that does just that, reducing the computational requirements of the modified mGA to $O(\ell \log \ell)$ overall.

3 Modifications for Fast mGA Processing

As was just pointed out, the original messy GA faces something of an initialization bottleneck, where the price for having all building blocks of a given size present is the need to evaluate a population of $O(\ell^k)$ structures. Having all structures present is certainly one way to go, but in a sense such partial enumeration seems unGA-like, and one colleague has questioned whether mGAs weren't in some sense partially substituting explicit parallelism for implicit parallelism. That criticism is correct, and in the earliest days of mGA investigation we rebelled against the need for partial enumeration. Even back then we tried to run populations with smaller numbers of members and longer strings. In this way, it was hoped, we could get one or more copies of all building blocks probabilistically. Those early efforts were unsuccessful, but the idea stuck with us, and it turns out to be one of the keys to eliminating the initialization bottleneck. In this section we develop an initialization technique called *probabilistically complete initialization* that permits us to create a controlled number of copies of building blocks of specified size.

One of the reasons early efforts at probabilistically-complete initialization failed was because we tried to use the longer strings directly in the mGA. As we later found out in subsequent mGA investigation (Goldberg, Deb, & Korb, 1990), the use of long structures with a large number of errors places too much of a load of parasitic bits on the system, mistakes that cannot easily be corrected once they are tagging along. This observation suggests that keeping long strings long is part of the problem, and the second part of our effort toward fast mGAs is to create a mechanism of *building-block filtering*. In this way, we start with long structures, but alternatively apply selection and random deletion at specified intervals. By alternating between selection and deletion we first pump up the good stuff enough so that when deletion destroys some of the building blocks we need, there still remain a substantial number for subsequent processing. It turns out that the scheduling of these operators is not too tricky, and bounding calculations have been performed to guide the process probabilistically.

In what follows, we first examine design calculations for probabilistically complete initialization and follow up with calculations that guide the sequence of operations involved in building-block filtering. Together, these techniques replace the old initialization and primordial phases of the original mGA. The new procedures present the remainder of the old mGA (the juxtapositional phase) with what it needs to do its search—an

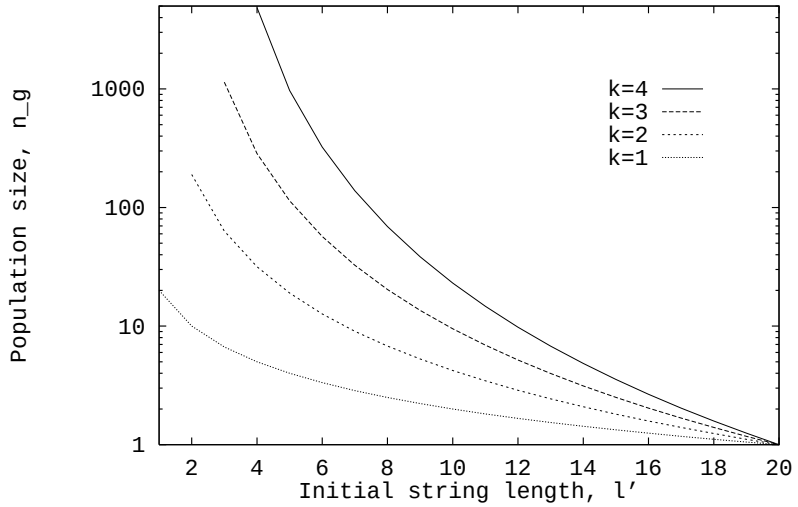

Figure 1: The population size required to have one expected instance of a building block of size k in strings of length ℓ' is plotted against ℓ'. The problem size $\ell = 20$ is assumed. The sizing requirement is about an order of magnitude less with the reduction of building block size by one.

adequate supply of good, clean building blocks—with similar reliability and much lower computational cost.

3.1 Probabilistically complete initialization

There are two decisions important to the initialization of an mGA: selection of the initial string length and selection of the population size. In this subsection, we consider the effect each of these has on the adequacy the processing. We start by examining the effect of string length and continue by considering the effect of population size.

Imagine that the initial strings are of length ℓ' larger than k and smaller than ℓ. The initial population needs to be sized so that all gene and allele combinations of size k are included in the population. We consider the gene-wise and allele-wise calculations separately. To have all gene combinations to order k, we calculate the probability that a random string of size ℓ' contains a gene combination of size k. The number of strings of size ℓ' created with ℓ genes is $\binom{\ell}{\ell'}$. The number of ways a string of size ℓ' contains a gene combination of size k may be calculated by assigning k genes to the string and then choosing the total number ways $\ell' - k$ genes can be created from $\ell - k$ genes. This quantity is $\binom{\ell - k}{\ell' - k}$. Thus, the probability of randomly selecting a gene combination of size k in a string of length ℓ' with a total number of genes ℓ is

$$p(\ell', k, \ell) = \frac{\binom{\ell - k}{\ell' - k}}{\binom{\ell}{\ell'}}. \tag{1}$$

Inverting this computation suggests that in each subpopulation of size $n_g = 1/p(\ell', k, \ell)$ strings of size ℓ' created at random, one string on average will have the desired gene combination of size k. Figure 1 shows the variation of n_g versus ℓ'. The figure shows that the subpopulation size reduces faster than exponentially with the initial string length ℓ'. It can be shown that for large values of ℓ and ℓ', $n_g \approx (\ell/\ell')^k$. For $\ell' \approx \ell$, n_g is a constant and does not depend on ℓ. It is also interesting to note that n_g reduces drastically with the reduction of k.

To include all allele combinations to order k, we have to multiply n_g by an appropriate factor that takes into account all 2^k allelic combinations and the noise faced by building-block evaluations. Elsewhere (Goldberg,

Deb, & Clark, 1991), a population-sizing equation has been developed for simple GAs to account for the building block evaluation noise:

$$n_a = 2c(\alpha)\beta^2(m-1)2^k. \tag{2}$$

Here the population size is calculated in a manner so that the selection error between two competing building blocks is no more than a desired value α. The parameter $c(\alpha)$ is the square of the ordinate of a normal random deviate whose tail has area α, β^2 is the maximum signal-to-ratio, m is the number of subfunctions. In strings of length ($\ell' < \ell$), the building block evaluation noise may be smaller than that in simple GAs, because a smaller number of stray bits are associated in the former case. The overall population size n may now be calculated by multiplying the two factors as follows:

$$\begin{aligned} n &= n_g n_a \\ &= \frac{\binom{\ell}{\ell'}}{\binom{\ell-k}{\ell'-k}} 2c(\alpha)\beta^2(m-1)2^k. \end{aligned} \tag{3}$$

To find a good initial string length ℓ', we should try to minimize the total number of function evaluations, and since the number of generations to convergence is roughly $O(\log n)$, minimizing n will minimize the number of function evaluations. Looking at the population-size estimate, assuming fixed β, the primary factor to consider is n_g. Since n_g reduces faster than exponentially with increased ℓ', the best choices of ℓ' are likely to be very close to ℓ. In our simulations, we have used $\ell' = \ell - k$. With this choice, and assuming problems of fixed and bounded difficulty, n_a is $O(\ell)$ (Goldberg, Deb, & Clark, 1991). Therefore, overall the above population sizing is $O(\ell)$. This is a substantial improvement of the initialization phase from that in the original messy GAs.

3.2 Building-block filtering via selection and deletion

In the initialization procedure described in the previous subsection, strings start off at a length near the problem length: ($\ell' \approx \ell$). The trick to getting the mGA to function properly is to reduce this initial length down to something like the building-block length k. In this subsection, we propose to do this by alternately performing selection and random deletion from strings in the population. The key to getting this to work right is to pump up enough copies of the good building blocks so that even after random deletion eliminates a number of copies there remain one or more copies for subsequent processing. We might view this as a schema theorem for selectodeletive filtering, and here we perform a rough analysis sufficient for design of the filtering technique.

We start by introducing some notation. Imagine a sequence of string-length values, starting with an initial length $\lambda^{(0)} = \ell'$ and continuing with successive lengths $\lambda^{(1)}, \lambda^{(2)}$ and so on until a reduction has occurred to a size near the building-block length k. Define the ith length-reduction ratio as $\rho_i = \lambda^{(i)}/\lambda^{(i-1)}$. At the ith stage, $\lambda^{(i-1)} - \lambda^{(i)}$ genes are randomly deleted from each string. After the reduction, selection is invoked for some number of generations (without new evaluation) to pump the proportion of good building blocks up to a reasonable level. Thereafter another round of deletions is performed and on the process goes.

To make this work we need to make sure that the disruption caused by deletion is less than the pumping action of selection. By analogy to the analysis of probabilistically-complete initialization calculations, to correctly choose a building block of size k from strings of length $\lambda^{(i-1)}$ by picking $\lambda^{(i)}$ genes at random, we need a building-block repetition factor $\gamma = \binom{\lambda^{(i-1)}}{\lambda^{(i)}} / \binom{\lambda^{(i-1)}-k}{\lambda^{(i)}-k}$ strings to have one expected copy remaining of the desired building block. For large values of $\lambda^{(i-1)}$ and $\lambda^{(i)}$ compared to k, γ varies as $(\lambda^{(i-1)}/\lambda^{(i)})^k$. We may choose $\lambda^{(i)}$ so that γ is smaller than the number of duplicates generated by the selection operator. Different strategies can be adopted to reduce the string length from ℓ' down to the order k. We choose to fix γ to a constant value much less than 2^{t_s}, where t_s is the number of selection repetitions per length reduction. This is done because we expect binary tournament selection to roughly double the proportion of best individuals during each invocation. Using the asymptotic relation for $\gamma = (\lambda^{(i-1)}/\lambda^{(i)})^k = \rho_i^{-k}$, we recognize that the assumed fixed γ roughly implies a fixed length-reduction ration $\rho = \rho_i$, for all i, and we calculate the total number of length reductions required to reduce the string length to $O(k)$. Assuming final string length equal to ζk, where $\zeta \geq 1$, the number of length reductions (t_r) required is given by the equation

$$\ell'/\rho^{t_r} = \zeta k. \tag{4}$$

```
Initialize a population of strings of size ℓ′ at random
Repeat
    Successive selections   {no evaluations}
    Choose ℓ″(< ℓ′) genes at random
    Set ℓ′ = ℓ″
    Evaluate all new strings
Until (ℓ′ ≤ ζk)
```

Figure 2: Some pseudo-code shows the modified initialization and primordial phase with repeated selection and deletion.

Simplifying we obtain, $t_r = \log(\ell/\zeta k)/\log\rho$. This suggests that if the $\lambda^{(i)}$ values are chosen to make the deletion so that the length-reduction factor ρ is a constant, t_r varies as $O(\log\ell)$. Since the population size is of $O(\ell)$ and since the evaluation of strings is performed only once after each length reduction, the overall complexity of initialization and primordial phases is expected to be $O(\ell\log\ell)$. Some pseudo-code showing the coordination of selection and deletion is presented in figure 2. It is nice that the constant-ρ length reduction yields $O(\ell\log\ell)$ steps, because this matches the computational requirements of the remainder of the messy GA; however, even if this reduction were done no more quickly than bit by bit, it may be shown that the selectodeletive primordial phase requires only $O(\ell^2)$ time steps.

Note that there is an interesting reversal here compared to the original messy GA. In the original mGA during the original primordial phase short strings required fill-in from the competitive template throughout the primordial phase. In the modified procedure, the competitive template is still required, but early on it plays very little role in string evaluation, and good building-block comparison depends on good statistical decision making alone. As the strings shorten, the competitive template takes on a greater role, but the reduction in the role for the competitive template should make the modified mGA less sensitive to the errors in the competitive template, thereby giving a more robust procedure. Nonetheless, we still recommend that level-wise processing be adopted using the best strings of one level to serve as templates for the next.

4 Thresholding Revisited

To restrict competition between building blocks with little in common, a genic thresholding mechanism has been used (Goldberg, Deb, & Korb, 1990), where tournament selection between two strings is only permitted if they share a greater than expected number of genes in common. In random strings of two different lengths, λ_1, λ_2, the expected number of genes in common is $\lambda_1\lambda_2/\ell$. This procedure has proven not to be adequate in the current application, and we increase the threshold value probabilistically to improve the procedure's effectiveness.

Specifically, we increase the threshold by some multiple of the standard deviation:

$$\theta = \lceil \frac{\lambda_1\lambda_2}{\ell} + c'(\alpha')\sigma \rceil, \tag{5}$$

where σ is the standard deviation of the number of genes two randomly chosen strings of possibly differing lengths have in common, and the parameter $c'(\alpha')$ is simply the ordinate of a one-sided normal distribution with tail probability α. In our simulations, we use a constant value of $c' = 3$. The variance of the hypergeometric distribution may be calculated as follows (Deb, 1991):

$$\sigma^2 = \frac{\lambda_1(\ell-\lambda_1)\lambda_2(\ell-\lambda_2)}{\ell^2(\ell-1)}. \tag{6}$$

As in the original thresholding implementation, a first string is chosen for the tournament, and other strings are checked to see if they meet the threshold requirement. As it is possible that no strings meet the requirement, a limit is placed on the number of candidate strings that can be checked to prevent checking all strings in the population or an infinite loop. In the original implementation this limit was called n_{sh}, and a value $n_{sh} = \ell$ was found to be sufficient. Since the threshold value here is more stringent, a larger n_{sh} value

Table 2: The Order-Three Subfunction.

String	function value	String	function value
000	28	100	14
001	26	101	0
010	22	110	0
011	0	111	30

is required and $n_{sh} = 1/\alpha'$ is used. This calculation is conservative, because the most difficult condition is the random population; after selection, there will be multiple copies of building blocks, and the probability of finding similar strings is higher.

5 Simulation Results

We present computational results using the mGA with modified initialization and primordial phase on various problems of bounded difficulty. Results on the baseline 30-bit, order-three problem (Goldberg, Korb, & Deb, 1989) are presented first, followed by order-five problems of lengths between 50 and 150.

5.1 Base-line results

The 30-bit, order-three deceptive problem was the first problem that was solved using the original mGAs. Since then, a number of researchers have used that function in their studies (Dymek, 1992; Eshelman, 1991; Muhlenbein, 1991, 1992; Whitley, 1991). Ten, order-three deceptive problems are concatenated together to form a 30-bit problem. Each subfunction has two optima—a deceptive optimum (000) and a global optimum (111). With 2 optima and 10 subfunctions the problem has a total of 2^{10} or 1,024 optima of which only one is global. The function values of the order-three subfunction is shown in table 2. The initial string length is assumed to be 27. The population sizing is calculated using equation 3. Calculating the signal-to-noise ratio for each subfunction using table 2, we calculate that the required population size is 3,331. A splice probability of one and a bit-wise cut probability of 0.03 is used. A template with all zeros is used.

Figure 3 shows the population best and average function value versus generation number, where an average of five runs is shown. The primordial phase lasts 28 generations. Strings are reduced in length from 27 to three in three steps, roughly cutting by half each time. After seven generations, the string length is reduced from 27 to 15. At generation 15, the string length is reduced to seven, and in generation 22 the string length is reduced to three. Even though the primordial phase continues for 28 generations, strings are evaluated only four times—generations zero, eight, 15, and 22. At generation 29, the juxtapositional phase begins. At this point, the building blocks are of length three and do not contain any stray bits. It takes only four generations (the minimum number of generations required to form a 30-bit string from three-bit building blocks by splicing alone) to find an instance of the global solution. In five randomly restarted runs, the global optimum was found each time, and the number of function evaluations required to find the global solution was no more than 26,650. The original mGA was able to solve this problem in roughly 40,600 function evaluations, and although we observe some improvement with the new procedure, the leverage of using fast mGAs is not substantial in smaller problems. In the following, we apply the modified mGAs on large-size problems and compare the computational complexity with that of the original mGAs.

5.2 Large-scale optimization

To investigate whether the modified mGA works on large-scale problems, a class of order-five difficult problems has been designed. Ten, fourteen, and thirty size-five subfunctions are concatenated together to form 50, 70, and 150-bit problems. The order-five problem used here is a trap function (Deb & Goldberg, 1992) as shown in figure 4. The figure shows the function values as a function of *unitation*—the number of 1s in the bit string. In such a function, all strings of identical unitation have identical function values. The function is fully deceptive, as can be verified with the conditions developed elsewhere (Deb & Goldberg, 1992).

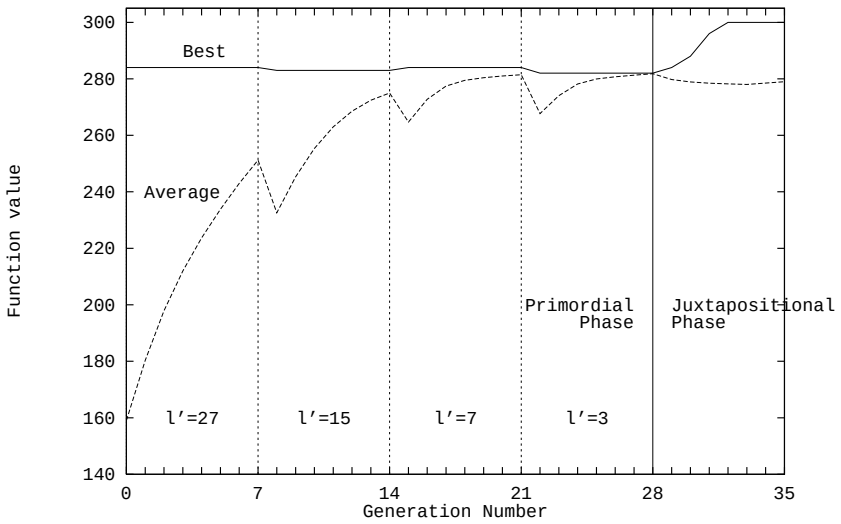

Figure 3: Population best and average function value are plotted versus generation number for the 30-bit, order-three deceptive problem using the modified mGA. An average of five runs is shown. The primordial phase continues till generation 28. The string length reductions during the primordial phase are also shown. At each instance of string length reduction in the primordial phase, all strings are evaluated.

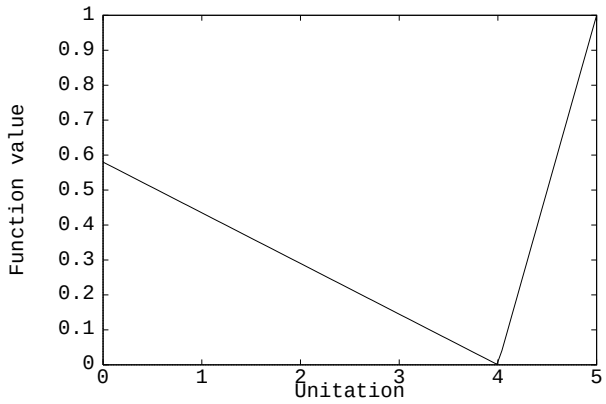

Figure 4: The five-bit subfunction is shown versus unitation (the number of 1s in a string). The subfunction has two attractors, one deceptive attractor at 00000 and one global attractor at 11111.

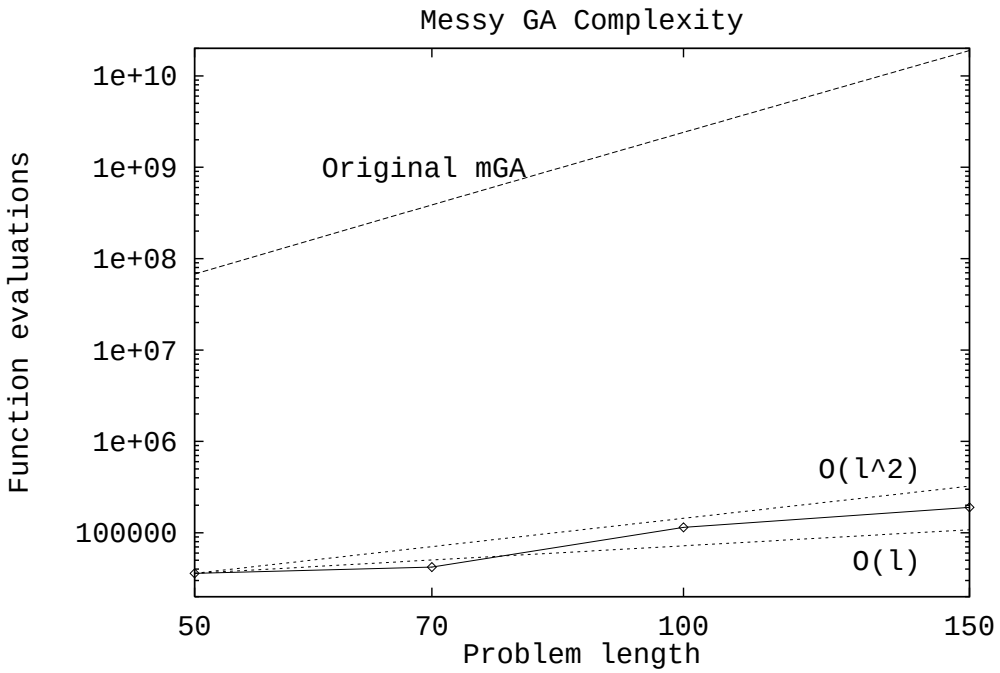

Figure 5: A graph of the number of function evaluations (averaged over five runs) to find an instance of a global solution is plotted versus problem length using the modified messy GAs and is compared to the estimated number of function evaluations required by original messy GAs. The original mGA for the order-five problem is $O(\ell^5)$, but the modified procedure is apparently subquadratic.

The population size in each problem is calculated with equation 3. In each case the initial string length starts at $\ell' = \ell - k$. Thereafter the string lengths are reduced so that $\gamma = 3$. In all simulations, we work at level $k = 5$ and use a worst-case template of all zeros. In a practical problem we should have performed levelwise processing at $k = 1, 2, 3, 4$ and finally 5., and this would have increased the number of function evaluations but not the asymptotic time complexity. According to our analyses, the computational complexity of the modified mGAs should not be more than $O(\ell^2)$ and should be more like $O(\ell \log \ell)$. Figure 5 plots the total number of function evaluations versus problem length. The ordinate is calculated as the total number of function evaluations required to find one instance of the global solution. Fixing $\ell = 50$ as the base point, two lines with $O(\ell^2)$ and $O(\ell)$ are drawn. The experimental points are shown to be no worse than the $O(\ell^2)$ line.

These are large, difficult problems. For example, in the 150-bit problem the search space is of size $2^{150} = 1.43(10^{45})$, and it has a total of 2^{30} or more than a billion optima of which only one is the global attractor. The modified mGA required only $1.9(10^5)$ function evaluations to find a global optimum, which is much less than the number of local optima and far less than the size of the search space. Even had we performed levelwise processing, the fast mGA would have remained subquadratic in asymptotic convergence rate, even though more function evaluations would have been required, and clearly this is to be much preferred to the $O(\ell^5)$ computations of the original scheme and simple hillclimbing (Muhlenbein, 1992).

Even though the modified mGA compares favorably to the original scheme. It should be pointed out that these results were not obtained through parameter fiddling. The original mGA settings together with the design calculations of this paper give good results without parameter adjustment, and these results should scale to other problem because of the sound theoretical basis for these settings.

5.3 An interesting connection

The revised algorithm was designed with speed and accuracy in mind, and in one set of problems it has apparently achieved that goal, but the overall pattern of processing makes an interesting connection with Davidor's observation of another variable-length GA (Davidor, 1989). Specifically, Davidor observed his variable-length GA going from long structures to shorter structures to longer structures, and later (Davidor, 1991) labeled this an *adaptation anomaly*. He also gave examples of other natural systems where such anomalies take place. Our work was not directly inspired by this example, but after we stood back from our design and results, the similarity in pattern became evident, and the long-short-long signature of the fast mGA is absolutely essential to obtaining fast results quickly. This lends support to Davidor's having singled out the anomaly as an interesting phenomenon, and it lends support to any hypothesis that suggests that the anomaly is a necessary part of fast, efficient processing.

6 Key Challenges

These first results using fast mGAs offer more-than-a-little hope that we may be able to solve problems of bounded difficulty quite quickly. The five types of function difficulty that have been identified are

1. isolation;
2. misleadingness (deception)
3. noisiness;
4. crosstalk;
5. massive multimodality.

With these difficulties in mind there appear to be three primary challenges that must be addressed before claims of mGA robustness will stand up:

1. Solve problems of nonuniform scale and size.
2. Attack problems with significant crosstalk and noise.
3. Attack massively multimodal problems.

In the following, each of these is described in some detail.

The test functions used here have simply taken a number of copies of the same deceptive subfunction and added their values. This type of test function adequately tests the mGA's ability to solve either problems of bounded isolation or bounded deception, but in mGAs there is the additional difficulty of separately evaluating separate building blocks. This problem has been addressed elsewhere (Goldberg, Deb, & Korb, 1990), but these issues must be examined with the modified primordial phase as well. The need to adequately evaluate building blocks with lower marginal fitness contribution places the thresholding and population sizing under a stringent test, and we believe that it will be necessary to develop a model of the effect of thresholding on building-block noise to properly account for these effects in the population sizing. The problems associated with mixed-size building-blocks appear less difficult, and connect with previous work on null bits and variable-length substructures (Goldberg, Deb, & Korb, 1990). To solve problems with nonuniform building blocks, the length-reduction phase should be discontinued when the string length is smaller than the maximum size of the building blocks. Null bits may be required to allow the smaller building blocks to be expressed without inhibiting other building blocks. In its present form, when a string reduction is performed, all strings are reduced to a fixed length. Variable-length reduction may be used so that the number of stray bits tagging along with building blocks reduces.

By adding together independent subfunctions, we have temporarily ignored the possibility of significant subfunction crosstalk. A reasonable model of a more difficult problem would be one where misleading or otherwise difficult subfunctions determined the building blocks, but the presence or absence of bit combinations *across* subfunction boundaries causes additional changes in fitness values without changing the underlying structure. Thinking about such functions is facilitated by some Walsh analysis (Goldberg, 1989a, 1989b), but

another approach is simply to add noise of specified variance to the underlying objective function. Experiments with such functions is encouraging, and it appears that appropriate population sizing that accounts for the crosstalk is all that is required to permit fast, accurate solutions. Of course such experiments should also validate the fast mGA's ability to deal with noisy functions robustly.

Massive multimodality challenges GAs in two ways. First, the presence of many local optima provides many opportunities for a population to get stuck locally instead of proceeding to a global or more nearly global solution. Additionally, in multimodal problems with a solution set containing more than one member, it might be desirable to have the population converge stably to all solutions or a representative subset of the solutions. This requires the introduction of mechanisms to allow the formation of multiple, stable species within separate niches or demes. Both of these issues have been addressed in the context of simple GAs under the assumption of tight linkage (Goldberg, Deb, & Horn, 1992). In that study simple GAs without a niching mechanism were able to find one of a set of global solutions reliably as long as the population was properly sized on signal-to-noise grounds. Moreover, with niching, all global solutions were retrieved as long as the global solutions were made salient with respect to the locals through a scaling mechanism. The first observation is likely to carry over to fast mGAs, but it needs to be tested, and experiments are currently underway. Introducing niching techniques should address the second concern, but some means of handling competitive templates is necessary. Various schemes are under consideration, from isolated subpopulations, to subpopulations with migration, to integrated niching schemes. Enough is known about various niching and deming schemes that carrying these over to fast mGAs should be feasible. Doing so is not only important for solving multimodal optimization problems, but it is also critical to those genetics-based machine learning systems in which the population contains multiple structures responsible for covering different parts of a solution.

Solutions to these challenges should permit fast mGAs to solve a very large spectrum of hard problems quickly. Carrying over these techniques from binary codes to permutation codes, floating-point codes, classifier codes, and program codes should be relatively straightforward once the basic technique is consolidated.

7 Conclusions

The holy grail of genetic-algorithm research has been robustness—broad competence and efficiency—because GA users would ultimately like to solve difficult problems quickly with high reliability, without having to twiddle with operators, codes, or GA parameters. The search for this GA chalice has been thwarted by the lack of a fully integrated theory of GA operation, but even without the whole story in hand, the crusade has been much aided by Holland's crucial illumination of the role of building blocks. Current efforts investigating the critical role of building-block mixing should soon yield predictions on the limits of simple GA performance and whether such often-suggested "fixes" as parameterized crossover, elitism, niching, and mating restriction can help simple GAs solve difficult problems quickly.

Early returns are mixed, but as we await these results a different branch of the search-for-robustness tree has borne its second fruit. Specifically, this paper has presented first results on fast messy genetic algorithms that deliver reliable solutions to certain large-scale, difficult problems in what appears to be subquadratic time. This remarkable speedup over the apparent difficulty-dependent polynomial convergence of the original messy GA was achieved by (1) initializing the population probabilistically with long structures instead of enumeratively with short structures and (2) replacing the old selective primordial phase with a filtering procedure that combines alternative invocation of selection and gene deletion. Together these two changes permit a collection of highly fit, reliable building blocks to be found in a population of modest, $O(\ell)$, size. These building blocks can then be processed by a somewhat modified juxtapositional phase that requires the addition of a more stringent genic thresholding operator.

The full demonstration of this capability across a spectrum of problems will be one of the most forceful realizations of implicit parallelism to date, but a number of challenges remain, including the investigation of problems with mixed size and scale, problems with crosstalk and noise, and problems with massive multimodality. Techniques to tackle each of these are in hand or are in development, but among competing alternatives, it appears that mGAs offer some of the best near-term prospects for solving hard problems quickly and reliably, once and for all.

Acknowledgments

We acknowledge the support provided by the US Army under Contract DASG60-90-C-0153 and by the National Science Foundation under Grant ECS-9022007.

References

Bradshaw, G. L., & Lienert, M. (1991). The invention of the airplane. *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, 605–610.

Davidor, Y. (1989). *Genetic algorithms for order dependent processes applied to robot path planning.* Unpublished doctoral dissertation, Imperial College for Science, Technology, and Medicine, London.

Davidor, Y. (1991). An adaptation anomaly of a genetic algorithm. *Proceedings of the International Conference on Simulation of Adaptive Behavior,* 510–517.

Deb, K. (1991). Binary and floating-point function optimization using messy genetic algorithms (Doctoral dissertation, University of Alabama). *Dissertation Abstracts International, 52*(5), 2658B. (Also available as IlliGAL Report No. 91004).

Deb, K., & Goldberg, D. E. (1991). *mGA in C: A messy genetic algorithm in C* (IlliGAL Report No. 91009. Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Deb, K., & Goldberg, D. E. (1992). *Sufficient conditions for deceptive and easy binary functions* (IlliGAL Report No. 91008). Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Deb, K., & Goldberg, D. E. (in press). Analyzing deception in trap functions. *Foundations of Genetic Algorithms.*

Dymek, A. (1992). *An examination of hypercube implementations of genetic algorithms.* (Masters thesis and Report No. AFIT/GCS/ENG/92M-02.) Ohio: Air Force Institute of Technology, Wright-Patterson Air Force Base.

Eshelman, L. J. (1991). The CHC adaptive search algorithm: How to do safe search when engaging in nontraditional genetic recombination. *Foundations of Genetic Algorithms*, 265–283.

Goldberg, D. E. (1989a). Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems, 3*, 129-152.

Goldberg, D. E. (1989b). Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems, 3*, 153-171.

Goldberg, D. E. (in press). Making genetic algorithms fly: A lesson from the Wright Brothers. *Advanced Technology for Developers.*

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems, 6*, 333–362.

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

Goldberg, D. E., Deb, K., & Korb, B. (1990). Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems, 4*, 415–444.

Goldberg, D. E., Deb, K., & Korb, B. (1991). Don't worry, be messy. *Proceedings of the Fourth International Conference in Genetic Algorithms and their Applications*, 24–30.

Goldberg, D. E., Deb, K., & Thierens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society for Instrumentation and Control Engineers, 32*(1), 10–16.

Goldberg, D. E., & Kerzic, T. (1990). *mGA1.0: A common LISP implementation of a messy genetic algorithm* (TCGA Report No. 90004). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

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

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-16*(1), 122-128.

Grefenstette, J. J. (in press). Deception considered harmful. *Foundations of Genetic Algorithms.*

Mitchell, M., & Forrest, S. (1991). *What is deception anyway? And what does it have to do with GAs?* Unpublished manuscript, Los Alamos National Laboratory, Los Alamos, NM.

Muhlenbein, H. (1991). Evolution in time and space—The parallel genetic algorithm. *Foundations of Genetic Algorithms*, 316–337.

Muhlenbein, H. (1992). How genetic algorithms really work I: Mutation and hillclimbing. *Proceedings of Parallel Problem Solving from Nature 2*, 15–26.

Whitley, L. D. (1991). Fundamental principles of deception in genetic search. *Foundations of Genetic Algorithms*, 221-241.