A Genetic Engineering Approach to Genetic Algorithms

John S. Gero john@arch.usyd.edu.au
Key Centre of Design Computing and Cognition, Department of Architectural and Design Science, The University of Sydney, NSW 2006, Australia

Vladimir Kazakov kaz@arch.usyd.edu.au
Key Centre of Design Computing and Cognition, Department of Architectural and Design Science, The University of Sydney, NSW 2006, Australia

Abstract

We present an extension to the standard genetic algorithm (GA), which is based on concepts of genetic engineering. The motivation is to discover useful and harmful genetic materials and then execute an evolutionary process in such a way that the population becomes increasingly composed of useful genetic material and increasingly free of the harmful genetic material. Compared to the standard GA, it provides some computational advantages as well as a tool for automatic generation of hierarchical genetic representations specifically tailored to suit certain classes of problems.

Keywords

Genetic algorithms, genetic engineering, evolutionary operators.

1 Introduction

Genetic algorithms (GAs) are a family of general stochastic search methods, which can be viewed as computational models of Darwinian evolution theory. They use the analogs of evolutionary operators on a population of states in a search space to find those states that optimize a fitness function. The search space consists of character-strings of fixed or variable length (chromosomes or genotypes) composed of the elements of a given alphabet (alleles). The genotype space is mapped onto another (phenotype) search space. The fitness function is defined as a function of a state in the phenotype space.

Since the biological metaphors (genetic representations, neo-Darwinian evolution theory) provide the conceptual basis of GAs, it seems natural to introduce some of the concepts of the most modern branch of biology – genetic engineering – into genetic algorithms.

2 Motivation

The primary motivation of this work is to identify and use any superior genetic material explicitly by means of genetic engineering. It is similar to the practice of genetic engineering in the genetics of natural organisms. In genetic engineering, the genetic engineer classifies the population into one group that possesses a high level of the property of interest or into another group that lacks it. We shall call them the *super* and *sub* groups, respectively. Then the genetic engineer tries to single out the groups of genes (we shall

call them the *superior* complex genes) in the genotypes that are hypothesized to be responsible for the properties of interest. Groups whose presence distinguishes the first group from the second are found using sequence and statistical analysis. The genetic engineer then attempts to produce the next population whose genetic material contains more of these useful complex genes. Further, he may try to prevent the damage of the superior complex genes by the evolutionary operators.

A complementary concept behind this work is to identify harmful or *defective* genetic material responsible for unwanted properties of the phenotypes. Processes that diminish or remove this harmful genetic material are then used. This concept is also inspired by the practice of genetic engineering in the genetics of natural organisms. We will refer to techniques that manipulate superior and defective genetic material methods as *advanced genetic engineering operators*.

3 Modified Genetic Algorithm

The GA is modified to include additional genetic engineering processes. The modified GA includes cycles where new superior and defective complex genes are evolved, and a new population that is richer in superior and poorer in defective complex genes is generated.

Two libraries of the descriptions of currently identified complex genes (superior and defective) are maintained. As the evolution process proceeds, enhanced by the inclusion of the genetic analysis and the advanced genetic engineering operators, new complex genes are identified and added to these libraries. The complex genes that have been incorporated into these libraries earlier are retested against the newly generated populations. This involves checking that they are still superior or defective complex genes for the current population. Those that do not pass this testing are deleted from the libraries.

The suggested modification of the GA models this simple picture of Darwinian evolution enhanced by genetic engineering technology. For each generation, the comparisons of genetic material of the most fit and most unfit subpopulations are carried out. This yields current knowledge about the useful and harmful genetic features. This knowledge is then used to genetically engineer the current population during a pre-reproduction stage.

The modified GA has the following general structure:

1. Initialization of the population (randomly) and two libraries of complex genes (usually two empty ones) – the superior and defective complex genes.

2. (a) Extraction of the super (highly fit) and sub (highly unfit) groups of individuals from the current population.

   (b) Identification of the superior and defective complex genes that distinguish these groups from each other at the genetic level. For example, this could be the most fit 10% and the most unfit 10% of the population.

   (c) Updating the complex genes' libraries by adding the newly evolved genes and eliminating the ones that test negatively.

   (d) Pre-reproduction processing step that includes various direct manipulations of genotypes of the population. The goal here is to produce superior and to get rid of defective complex genes in the genetic pool. The type of the genetic engineering technique employed determines the type of processing.

3. Reproduction.

4. If the stop conditions (for example, the given number of generations has been produced or the population has converged, etc.) are not met, go to step 1.

Note that steps 1 and 2 of the algorithm may be executed after a fixed number of generations at predefined intervals.

A complex gene is defined as an arbitrary characteristic feature of a genetic string. Many different types of complex genes can exist depending on the specific problem and type of encoding used. It could, for example, be a contiguous genetic substring, a subsequence (a sequence obtained from the genotype by deleting a number of genes), a periodic structure in a genotype, etc. In this paper, we consider only two types of complex genes. We shall call the first one a *local* gene. It is simply a building block in the standard GA – a set of fixed positions within the genotype with fixed values. It exists in problems with position-dependant encoding. The term *local* gene emphasizes the fact that this type of complex gene is an entity that exists only in a given position within the genotype.

We shall call the second type of complex gene a *global* complex gene. It is defined as a genetic substring that may reside anywhere within the chromosome and can exist in problems with non-positional encoding. We can treat it as an extension to the problem's alphabet – an additional letter that can be used just as the original letters from the alphabet are used.

Since the genetic engineering GA produces complex genes recursively, it yields a hierarchy of complex genes. The letters from the genetic alphabet are on the lowest (0) level of this hierarchy. Complex genes built using only elementary genetic letters (not less than two) are on the next level (level-1) of hierarchy, and complex genes on the l-level of this hierarchy are made using at least one complex gene from level $l-1$ and without using any genes from level higher than $l-1$. We will characterize complex genes by their position within this hierarchy. The corresponding measure is called the *l-complexity* of the complex gene. It is defined recursively as 0-complexity for any elementary gene, 1-complexity for complex genes composed of elementary genes exclusively, and l-complexity for complex genes built using complex genes of the complexity $l-1$, etc. The l-complexity is a measure of the number of assembling levels that are used to construct the corresponding complex gene.

We consider any gene in the chromosome to be a member of not more than one superior and one defective complex gene. If more than one superior or defective complex gene competes for the same position, then it belongs to the one that evolved later.

A number of algorithms similar in spirit to the genetic engineering modification of the GA have been developed in the area of genetic programming (Angeline, 1994; Koza, 1992; Rosca and Ballard, 1994), as well as in GAs (Corcoran and Wainwright, 1994). In the terms used in this paper, they attempt to determine the superior complex genes and then use them globally as extra letters of the genetic alphabet. Various modifications of the crossover and mutation operators have been proposed that make the disruption of these genes less likely. The genetic engineering extension to GAs proposed here differs from these algorithms in the following ways: it relies on the identification and processing of not only the superior but also the defective complex genes; the identification of the complex genes is executed differently, and a much wider range of evolutionary operations is used, all within a single conceptual framework.

4 Position-dependent Encoding

According to the building-block hypothesis (Goldberg, 1989), the work of the GA is based on the implicit determination and parallel processing of groups of the chromosome's components whose values are fixed and which are called building blocks (schemata). This hypothesis states that GA search is successful if the highly fit, short building blocks are recombined by the algorithm to form more highly fit, higher-order schemata. This leads, in effect, to the replacement of the entire search space with its subspace defined by these building blocks. This monotonic reduction of the effective search space is interpreted as an exploitation of the building blocks. GAs can be viewed as a tool for the generation of a hierarchy of these fit building blocks (and a hierarchy of the corresponding subspaces of the search space). Mutation and crossover that destroy the fit building blocks are viewed as disadvantageous – they disrupt and slow the search process.

The definition of building blocks in the standard GA is a direct consequence of the position-dependent genetic encoding commonly employed in GAs. The semantic of the genotype's interpretation is tied to the gene's position as well as to its content, i.e.,

$$x(n) = f(u_n) = f(u(1), u(2), \ldots, u(n)) \tag{1}$$

where $u_n = \{u(1), \ldots, u(n)\}$ is the chromosome, $N_{\min} \leq n \leq N_{\max}$ is its length (in this section we assume that n is constant), and $x(n)$ is the resulting phenotype. No special assumptions are made about the mapping f. Consequently, in the general case, it is position-dependent – that is, f depends differently on the different components of the genotype $\{u\}$. The mapping $f(u(k))$ is different for different k and thus depends explicitly on assume that the genotype to phenotype mapping f in Equation (1) can be written as a recursive multistep process

$$x(i+1) = \hat{f}(i, x(i), u(i)); \quad x(1) = x_1, \quad i = 1, \ldots, n-1 \tag{2}$$

then if $\hat{f}$ does not depend on i explicitly (that is, $\hat{f} = \hat{f}(x, u)$), the encoding is non-positional dependent.

We will use both the terms building block (schema) and local complex gene interchangeably for the complex genes in problems with position-dependent encoding.

4.1 Identification of Building Blocks

Formally, we define superior building blocks as those that are present at least k_1 times in genotypes from the super group and at most k_2 times in genotypes of the sub group. Their identification can be carried out using a hierarchical clustering technique (Louis et al., 1993). Since these algorithms have quadratic time complexity (Anderberg, 1973), the overall time complexity associated with the identification of the superior and defective building blocks is $O(N \log N)$ (the computational cost of sorting the current population) plus $O(N^2)$ (the cost of hierarchical clustering) and plus $O(N^2 M)$ (the cost of calculating the distance matrix between analyzed points). Here N is the combined size of the super and sub groups and M is the length of the genotype.

4.2 Crossover That Does Not Damage the Superior Building Blocks

Since our first goal is to prevent damage to the superior building blocks during reproduction, the crossover operator is changed in the following manner. If the crossover point happens to be located within the boundaries of the superior building block, then this block is passed (as a whole entity) to one of the offspring, as shown in Figure 1. If the

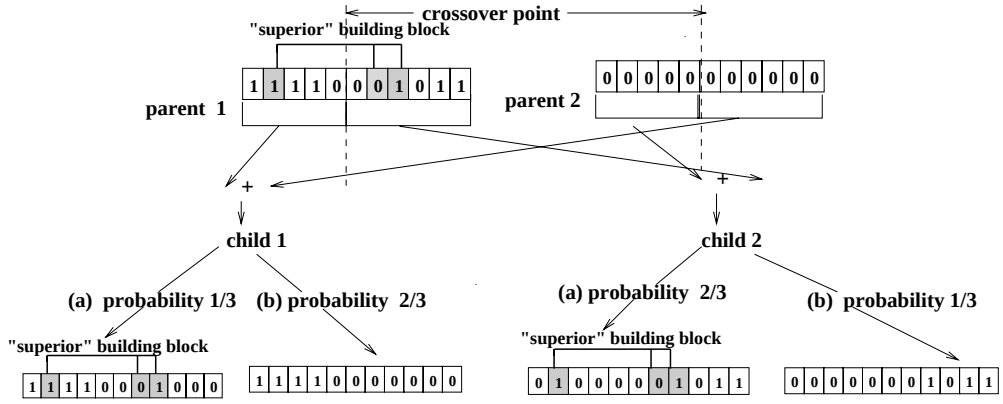

Figure 1: Passing of superior building block from parents to offspring.

parts of the superior building block that would be cut during the standard crossover are also present in the offspring, they override the corresponding genes from the other parent. This is executed randomly with the probability of the offspring inheriting a superior building block during the crossover determined as follows. It is the ratio of the number of components of this block located in the part of parent's genotype that would normally be transferred to the offspring during the standard crossover, to the total number of components of this building block. If two different superior building blocks in two parents compete for the same gene position in the offspring genotype, we select the one whose parent has the higher fitness. Since we also do not want mutation to damage the superior building blocks, the mutation rate of genes that belong to these superior building block, p_m^{sup}, is set lower than the probability of mutation of ordinary genes, p_m, i.e., $p_m^{sup} << p_m$.

4.3 Crossover That Disrupts Defective Building Blocks

Since we also wish to destroy defective building blocks, we arrange the crossover operator to cut them more often than it does ordinary building blocks with similar parameters. A crossover operator can be defined for this that has a probability twice that of cutting across only one such block. It can cut across only one such block with a probability p_{def} times higher than cutting outside of any of the defective building blocks. A boundary occurs between two contiguous genes and is normally the length of a genotype or building block minus 1. Let n be the length of the chromosome, n_1 be the number of the boundaries that reside within the defective building block in one parent only and n_2 be the number of boundaries which are located inside defective building blocks in both parents. The crossover points are chosen with probabilities $\frac{2p_{def}}{n-1-n_1-n_2+p_{def}(n_1+2n_2)}$ for the boundaries that reside completely inside two defective building blocks in both parents. Values of half of this value are chosen, where the boundaries between two genes reside completely within the defective building block in only one parent and $\frac{n-1-n_1-n_2}{n-1-n_1-n_2+p_{def}(n_1+2n_2)}$ for the boundaries outside of the defective building blocks in both parents, as shown in Figure 2. $p_{def} > 1$ is the "weight" coefficient that causes the crossover to cut through the defective building blocks more frequently than through the normal ones. This leads to the more frequent damage of the defective building blocks by the crossover operator compared to the non-defective ones with the same parameters.

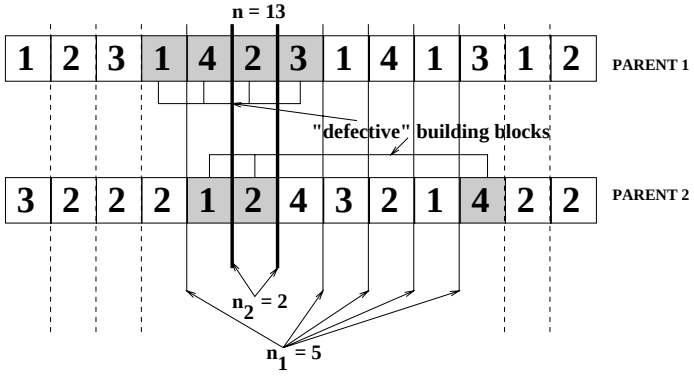

Figure 2: The probability of crossover within the boundaries ($n_2 = 2$) of two defective building blocks (bold vertical lines) $\frac{2p_{def}}{6+9p_{def}}$. This is twice as high as the probability of crossover across only one such block ($n_1 = 5$) (solid lines). This is higher than the probability of crossover through boundaries outside of the defective building blocks (dashed lines) $\frac{1}{6+9p_{def}}$.

4.4 Chemical/Radiation Therapy Against Defective Building Blocks

An analog of chemical/radiation therapy can be constructed by setting the mutation rate of the genes that belong to the defective building blocks p_m^{def} much higher than the mutation rate of the non-defective genes ($p_m^{def} >> p_m$). This leads to the rapid destruction of the unwanted genetic material.

4.5 Gene Surgery – Replacement of the Defective Building Blocks with Superior Ones

The goal in this process is to replace a defective building block with the most similar superior one. The similarity can be measured either by the number of equal valued gene positions or by the number of gene positions that are included (not necessarily with the same values) in both building blocks. Depending on which approach is chosen, $n_{overlap}$ is either the number of chromosome positions that have the same values in both blocks or the number of chromosome positions that are fixed in both building blocks. Then n_{def} is the number of fixed chromosome components in the defective building block (in our simulations we chose the first definition of $n_{overlap}$). The pre-reproduction stage of the modified GA, which models this gene surgery, consists of the replacement of defective building blocks with the matching superior ones with the probability $p_{sur} = \frac{n_{overlap}}{n_{def}}$, as shown in Figure 3.

4.6 Gene Therapy – Random Replacement of the Genetic Material with the Superior Building Blocks

Gene therapy can be modeled as the random replacement of genetic material with superior building blocks. The chromosomes from the current population are picked randomly with probability p_{pick} and the superior building blocks from the super group are chosen randomly with probability $\frac{1}{n_{sup}}$ (n_{sup} is the number of superior building blocks evolved) to overwrite the corresponding parts of the genotypes.

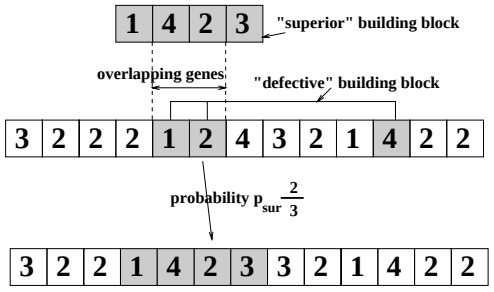

Figure 3: Gene surgery: replacement of defective genes with superior ones.

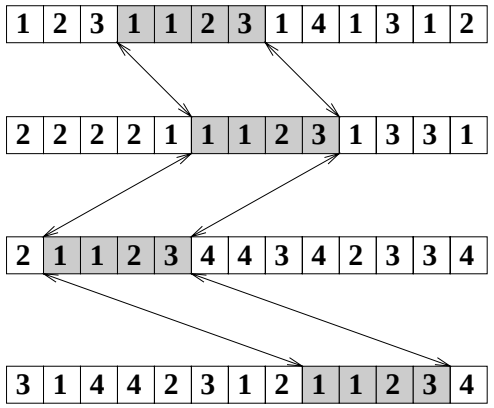

Figure 4: The matching subsequences in a group of genotypes.

5 Non-positional Encoding

The same substring can be responsible for wanted or unwanted properties no matter where it is located within the genotype, as shown in Figure 4. The sequence analysis techniques of genetic engineering and of speech processing (Sankoff and Kruskal, 1983; Collins and Coulson, 1987; Schuler et al., 1991; Needleman and Wunsch, 1970; Karlin et al., 1990), which detect similarities or dissimilarities between groups of sequences, are used to identify these substrings. Their fitness contributions (through the phenotype) are independent of where they reside within the genotypes. These substrings are new genetic primitives that form a specialized genetic alphabet characteristic for the organisms with given features. They correspond to the stable components in terms of complex system theory (Simon, 1973). There is related work on floating building blocks (Wu and Lindsay, 1995).

If such non-positional dependence exists, then it is possible that, for each class of problem, there is a natural, hierarchical, variable-length alphabet whose member's significant presence in the genetic pool of the population leads to the high level of fitness.

This alphabet is a set of genes of varying complexity, as shown in Figure 5. Similarly, a complementary alphabet of the defective complex genes can exist that can be used to generate a population with a low level of fitness.

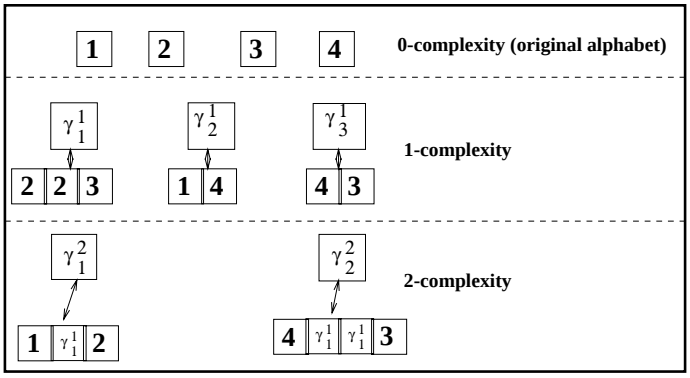

Figure 5: The alphabet of complex genes.

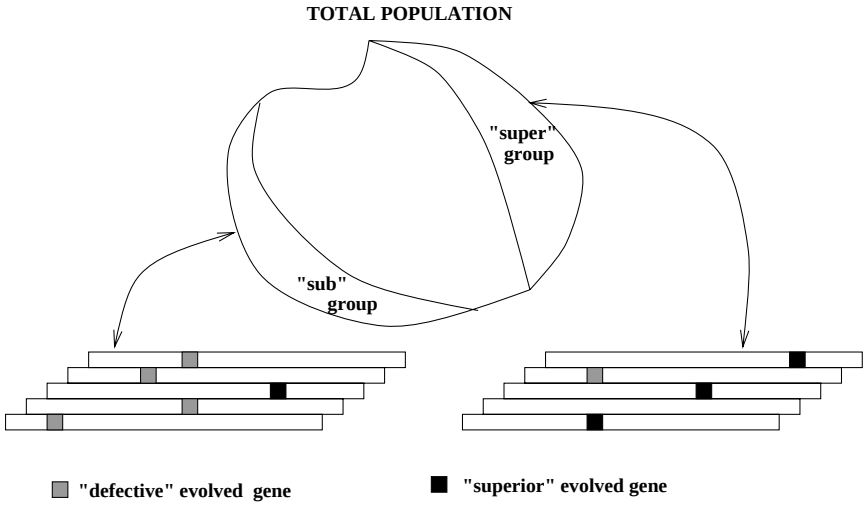

Figure 6: Identification of evolved genes. Here parameters are $k_1 = 3$, k_2=1.

5.1 Identification of Genetic Words

Let us assume that complex genes are identical contiguous pieces of the genotypes that can be located anywhere within the genotype, as shown in Figure 4. We will call these substrings *words* so we can identify them from just any substring of a genotype. We identify superior complex genes by making a list of all words that are used to build at least k_1 genotypes from the super group and at most k_2 genotypes from the sub group (k_1, k_2 are parameters, $k_1 > k_2$), as shown in Figure 6.

Defective complex genes are identified in a similar fashion by making a list of all words that are used in construction of at least k_1 genotypes from the sub group and at most k_2 genotypes from the super group. These lists can be produced using suffix trees for the super and sub groups. A suffix tree is a standard tool in sequence analysis (McCreight, 1976; Weiner, 1973; Aho, 1975; Apostolico, 1985; Crochemore, 1994). Let us give a brief review of suffix trees. It is defined for any sequence $S = \{a_1 \dots a_n\}$ of n symbols as a tree with $n+1$ leafs and at most n interior vertices. Each edge is labeled with some subword of the sequence or word, and each leaf is labeled with a unique position in this

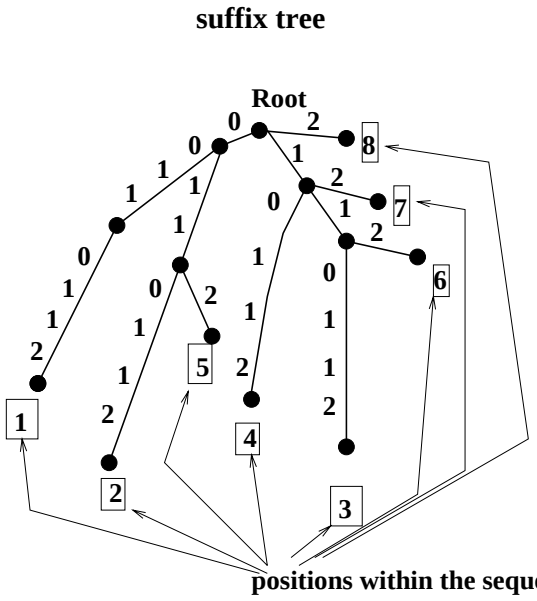

Figure 7: Suffix tree for the genotype $S = \{00110112\}$.

word. Concatenation of the edges' labels along the path from the root to the leaf labeled j spells the subword $S_j = \{a_j \dots a_n\}$ from the jth position till the end (suffix), as shown in Figure 7.

For example, let $S = \{00110112\}$, then the suffixes are $S_1 = S$, $S_2 = \{0110112\}$, $S_3 = \{110112\}$, $S_4 = \{10112\}$, $S_5 = \{0112\}$, $S_6 = \{112\}$, $S_7 = \{12\}$, and $S_8 = \{2\}$. The suffix tree built from these 8 suffixes of S is shown in Figure 7. It is easy to check that leaf 3 stores suffix S_3. Concatenation of labels of all the edges along the path from the root to this leaf spells $S_3 = \{110112\}$.

The primary purpose of a suffix tree is to store all the suffixes of a given word. It can also be viewed as a compact collection of all the subwords in a given word. For any subword in this word, there is a unique path from the root of the tree to some node in it that spells the subword (i.e., the concatenation of the labels of the edges along this path spells this subword). The suffix tree can be constructed incrementally in $O(n)$ time and requires $O(n)$ space using the methods described in McCreight (1976) and Weiner (1973).

Genetic analysis begins with the creation of two genetic superstrings (one for sub and one for super groups) by concatenating all the genotypes from the corresponding group separated from each other by distinct markers (symbols that are not used in genotypes), as shown in Figure 8. Then we construct two suffix trees for these superstrings using McCreight's algorithm (McCreight, 1976). The leaves of the resulting trees are divided into bunches. Each bunch corresponds to one of the genotypes from the super or sub group, as shown in Figure 8. Finally, we explore each tree looking for the deepest path that is extensible to not less than k_1 bunches. This means that the corresponding subword spelled by the concatenation of the edges' labels from the root to the corresponding node appears in at least k_1 genotypes. Then we check if this subword exists in a complementary suffix tree. If it does not, then we have found a complex gene. If it does exist, then it is a complex gene only if all the branches that begin in the corre-

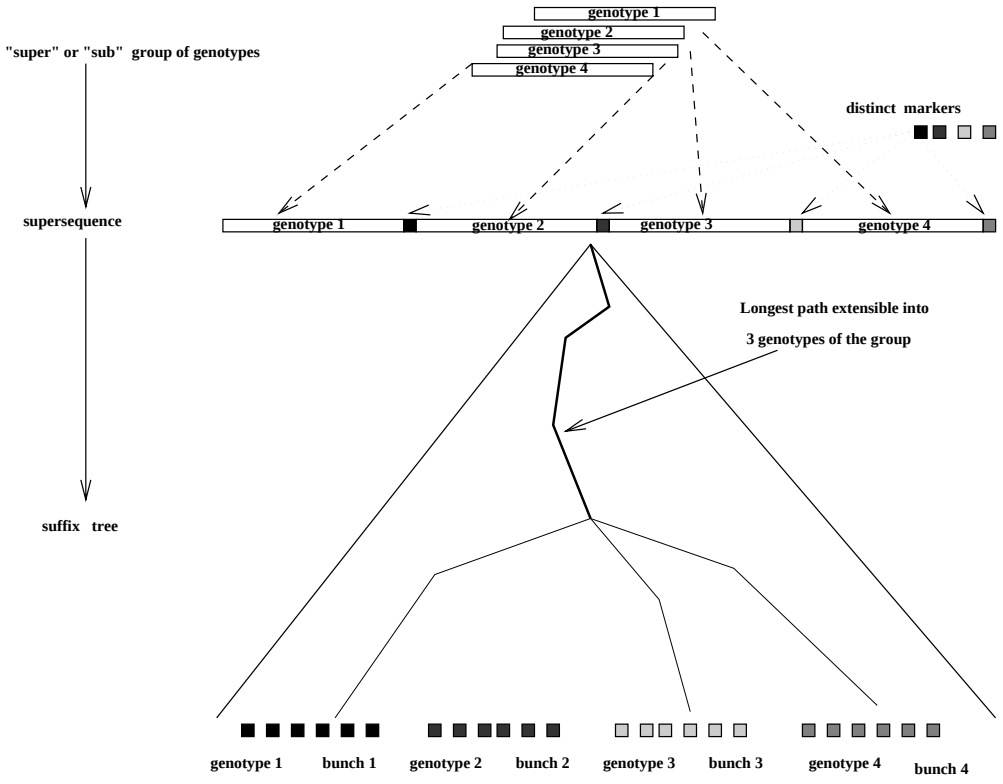

Figure 8: Schematic structure of genetic engineering analysis for finding evolved genes as genetic words.

sponding node end up in not more than k_2 bunches.

The overall complexity of this type of genetic analysis is linear with respect to the combined length of analyzed genotypes from super and sub groups.

Thus, the overall computational complexity of identifying genetic words is $O(N \log N)$ (computational cost of sorting the population) plus $O(NM)$, where N is the combined size of super and sub groups, and M is the length of the genotype.

5.2 Packed Chromosomes

Since the evolved complex genes have variable lengths, it is natural to consider variable-length genotypes. The genotypes that are written in terms of the original alphabet (we shall call them the unpacked genotypes) can be rewritten in a packed form when all the instances of the complex genes are replaced with a single, packed gene, as shown in Figure 9. To make this one-to-one transformation, we first replace all the superior complex genes of 1-complexity, beginning with the first one evolved, then all superior complex genes of 2-complexity, etc. Then the same process is executed with respect to the defective complex genes. All the genotypes are subjected to such packing during the pre-reproduction stage. They are unpacked before the evaluation of the population. All the complex genes that are present in the population's genetic pool can be found using the Boyer-Moore-Galil string search algorithm in $O(2L_{pop})$ time (Crochemore, 1994) (where L_{pop} is the total combined length of the population's genotypes).

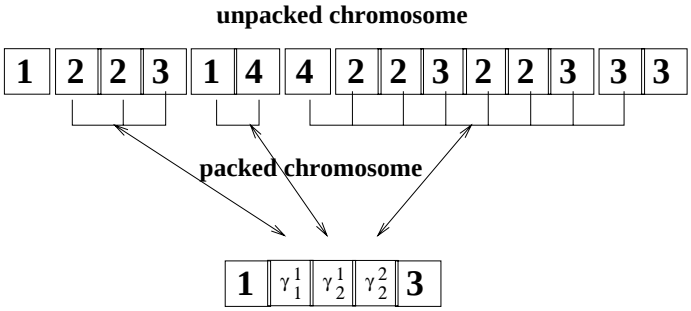

Figure 9: The packed and unpacked genotypes.

We choose to use the standard GA's crossover and mutation operators on the packed chromosomes in our numerical simulations. It is clear that the corresponding operators on the unpacked chromosomes produce different results from the standard ones.

5.3 Crossover

Here, the one-point crossover operator differs from the GA standard crossover in one respect only. In the general case, the same crossover points in two packed genotypes correspond to two different points in the respective unpacked genotypes. Thus, it seems reasonable to crossover two packed parents in two different points (possibly to cross the first one at a random point and the second one at a point chosen at random in some neighborhood of the first one, as shown in Figure 10(a)).

The analog of the two-point crossover used in genetic programming picks and swaps two random continuous pieces in two packed genotypes, as shown in Figure 10(b). These crossover operators do not disrupt the superior complex genes that happen to be present in the population's gene pool.

5.4 Mutation

The mutation operator needs to be changed so that it is less likely to affect the superior complex genes and more likely to affect the defective complex genes than the ordinary genes. It can be carried out in a number of different ways using analogs of radiation / chemical therapy. For example, one can set the probability of mutation of the superior complex genes p_m^{sup} to be lower than the rate of an ordinary gene's mutation p_m, which in turn is lower than the mutation rate of the defective complex genes p_m^{def}. It is possible to treat complex genes either as equal to the elementary ones or as a separate set of entities. In the latter case, it seems reasonable to define the mutation of a superior complex gene as its random deletion or replacement with one of the other superior complex genes of the same complexity. The mutation of the defective complex gene can be defined as its random deletion or unpacking. Here unpacking implies replacing it with the corresponding sequence of the original genes and mutation of its components (macromutation in the unpacked chromosome's space). If complex and elementary genes are considered equal, then one can define a mutation of the elementary gene into a complex one and the complex gene into an elementary one.

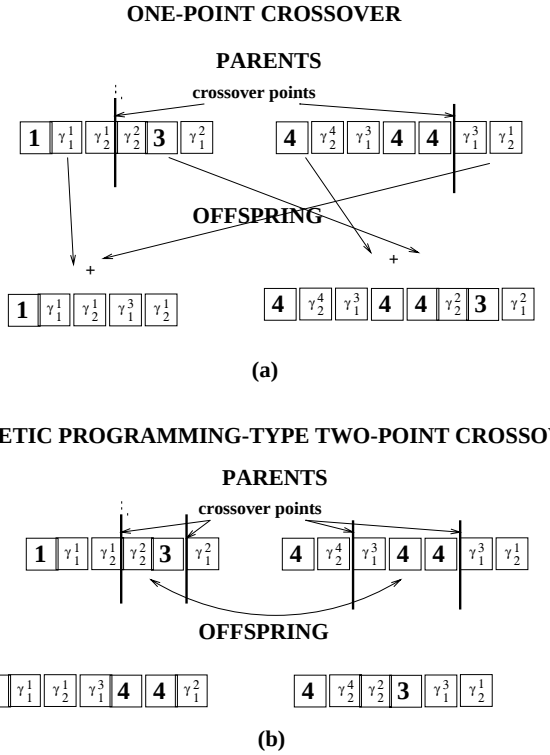

Figure 10: The crossover of two packed chromosomes. (a) genetic algorithm-type crossover. (b) genetic programming-type crossover.

5.5 Gene Surgery and Gene Therapy

Let $n_{overlap}$ be a measure of the similarity of the superior and defective complex genes of the same complexity (for example, the number of their matching components), and n_{def} is the length of the defective complex gene in the unpacked form. Then an analog of gene surgery can be constructed as follows. During the pre-reproduction stage, all the genetic material in packed form is scanned, and all the defective complex genes found are replaced with one of the superior complex genes of the same complexity with a probability p_{sur} proportional to the similarity of these two complex genes, where $p_{sur} = \frac{n_{overlap}}{n_{def}}$, $0 \leq p_{sur} \leq 1$. The analog of the gene therapy operation scans the genetic material and inserts randomly picked superior complex genes in random positions.

6 Experimental Results

6.1 Positional Encoding

The test set of the fixed-length problems with positional encoding consists of the Royal Road problem (Jones, 1995) and the De Jong test suite (F1-F5) (De Jong, 1975).

Royal Road Function This class of problems has a clearly defined hierarchy of building blocks and is an ideal test bed for investigating the building block processing in GAs (Forrest and Mitchell, 1993). A genotype of this function is composed of 2^k non-overlapping continuous regions. The first, of length b, is called the *block*. The second, of

Table 1: Comparative performance of standard and genetic engineering-based GAs on royal road functions. The table gives the mean fitness evaluations taken to find the optimum over 50 runs.

	R_1	R_2	canonical
total bits	64	64	240
Standard GA	61342	70632	4680457
Genetic Engineering GA without advanced genetic operations	43449	50499	2906123
Genetic Engineering GA with radiation therapy	42099	50335	2911871
Genetic Engineering GA with gene surgery	37019	51331	2345127
Genetic Engineering GA with gene therapy	39066	49133	2301214

length g, is called the *gap*. The length of each region is $b+g$. They are labeled from left to right. The fitness depends only on the genes from blocks. The blocks that are composed of 1s only are called *complete*. Each incomplete block contributes the following term into the overall fitness

$$F = \begin{cases} v{\cdot}m, & \text{if} \quad m =< m^*, \\ -v(m^* - m), & \text{if} \quad m > m^*, \end{cases}$$

where m is the number of 1s in this block. Extra reward is given for attaining what Holland called "levels" - the complex building blocks whose fitness is higher than the sum of the fitnesses of its components. At the lowest level, if the genotype contains n_1 complete blocks, then the term $u_1^* + u_1{\cdot}n_1$ is added to its fitness function. If the genotype contains n_2 pairs of complete blocks with the labels $(2i, 2i+1)$, then, again, $u_2^* + u_2{\cdot}n_2$ is added to the fitness. Then the set of complete quadruples is rewarded in the same fashion, etc. Here, m^*, v, u_p^*, and u_p, $p = 1, \ldots, k+1$ are parameters.

We use three functions from this class: R_1 and R_2 functions (Forrest and Mitchell, 1993) and the canonical function with default parameters (Jones, 1995). The R_1 fitness function contains only contributions from the complete blocks of the lowest level. The contributions from the incomplete blocks as well as from the combinations of the complete blocks are omitted. The corresponding parameters are $k = 3$, $b = 8$, $g = 0$, $m^* = 0$, $v^* = 0$, $v = 0$, $u_1^* = 0$, $u_1 = 8$, $u_p^* = 0$, $u_p = 0$, and $p = 2, 3, 4$. The R_2 function differs from the R_1 function in one respect only – it rewards the pairs, quadruples, and octuples of complete blocks. The following parameters are different: $u_2 = 16$, $u_3 = 32$, and $u_4 = 0$. The canonical Royal Road Function has the following parameters: $k = 4$, $b = 8$, $g = 7$, $m^* = 4$, $v = 0.02$, $u_p^* = 1.0$, and $u_p = 0.3$.

We used the same generational GA with one-point crossover and σ-scaling as was used in Forrest and Mitchell (1993) with the same parameters: crossover probability $p_c = 0.7$ and mutation probability $p_m = 0.005$. In the experiments with R_1 and R_2, the population size was 128. In the experiments with the Royal Road Function, the population size was 500. The results of computations are presented in Table 1. For the advanced genetic engineering operations (used on pre-reproduction stage), we used the following parameters: $p_m^{def} = 0.1$, $p_m^{sup} = 0.0005$, $p_{sur} = 0.7$, and $p_{pick} = 0.1$.

The performance comparisons of the standard and genetic engineering-based GAs

Table 2: Comparative performance of standard and genetic engineering-based GAs on the function from the De Jong test suite. The table gives the mean fitness evaluations taken to find the optimum over 30 runs.

	$F1$	$F2$	$F3$	$F4$	$F5$
Total bits	30	24	30	240	34
Standard GA	57945	39634	706	15023	4517
Genetic Engineering GA without advanced genetic operations	37689	28892	497	9612	2814
Genetic Engineering GA with radiation therapy	34123	27181	503	7515	2717
Genetic Engineering GA with gene surgery	28751	23111	421	6901	2111
Genetic Engineering GA with gene therapy	27113	24037	401	6176	2231

on the De Jong test suite are presented in Table 2. The population size was 50.

The results show that the genetic engineering-based GAs perform significantly better than the standard GA (saving of $30-50\%$ of function evaluations). The basic genetic engineering extension of GA without advanced genetic engineering operations provides savings of $30-37\%$, with the genetic surgery and genetic therapy each providing additional computational savings of about 10%. The total computational cost of one cycle of this genetic engineering analysis was about 30% of the computational cost of producing one generation for the functions from the De Jong test suite and about 60% for the royal road functions. The average additional computational cost per generation for the basic genetic engineering GA is 10% for the functions from the De Jong test suite and 15% for royal road functions. For a GA with advanced genetic engineering operations, these additional costs are correspondingly 17% and 21%. Thus the net effective computational savings were between 12% and 20%.

Radiation therapy does not provide any additional advantages on these test problems. The likely cause of this is the absence of defective" building blocks in these problems.

6.2 Non-positional Encoding

Our test suite of the variable-length GA problems with non-positional encoding consists of two problems: a beam cross-section design problem from structural engineering and the non-positional modification of the royal road function.

6.2.1 Beam Cross-section Design Problem

In this problem, the beam cross-section is generated using a simple shape grammar over a uniform planar grid (Gero and Kazakov, 1996). This generation is a recursive process that places elementary cells with different weights on the grid (the feasible weights are $1,\ldots,7$). During each step of this process, a new cell is placed in one of the eight neighboring positions of the last elementary cell that has been added to the current cross-section, as shown in Figure 11.

If the grid position where the next cell is to be located is already taken, then all the corresponding cells already placed on the grid are shifted. The process is initiated when

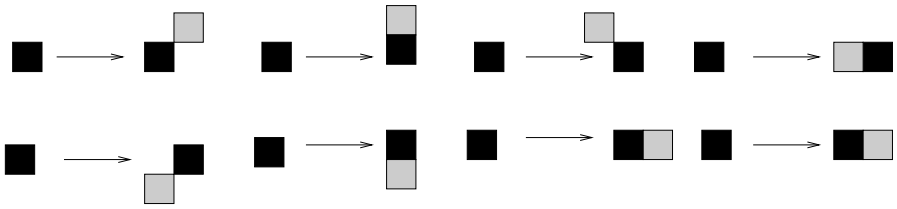

Figure 11: The genotype-phenotype mapping used for the beam design problem.

the first elementary cell is placed in the middle of the empty grid. This process completely defines the genotype to phenotype mapping. The first gene of the genotype defines the weight of the first cell, its $((3 + 2 * (i - 3))$th genes define the weight of the ith elementary cell ($i = 2, \ldots, 19$) added to the beam cross-section. The values of genes belong to the discrete range of values $1, \ldots, 5$ that directly encode weights of corresponding cells. The $(4 + 2 * (i - 3))$th genes ($i = 2, \ldots, 19$) in the genotype define the relative position (with respect to the position of the $i - 1$ cell) of the ith elementary cell added to the beam cross-section. The values of these genes are $0, \ldots, 8$. They correspond to the 8 possible locations of the ith cell with respect to the $(i - 1)$th, as shown in Figure 11. Only the first 20 cells are placed onto the 2-D grid. Thus, although the problem has variable-length genotypes with unrestricted lengths, only 39 genes are expressed. Only the first 39 genes determine the phenotypes. Different geometric properties could be used as fitness functions. The particular fitness function F used in these experiments was:

$$F = \{\textit{moment of inertia}\,;\,\textit{total area of holes}\,;\,\textit{number of holes connected to outside space}\} \longrightarrow \max .$$

In order to handle the multiobjective optimization, we use a simple Pareto-based ranking procedure. We take some positive number $U > 0$ and set a counter $k = 1$. Then we find all points in the current population that belong to its Pareto-set and assign the fitness value for all of them equal to $\frac{U}{k}$. We then set $k = k + 1$. Then we consider the rest of the population without the points just found, single out the Pareto-set for this reduced population, and set the fitness values for the members of this set equal to $\frac{U}{k}$. Set $k = k + 1$, and the cycle continues until all the points are ranked. The simplest version of a pattern recognition algorithm was implemented that allows for the search and identification of double and triple element complex genes only. The following parameters have been used: population size is 200, crossover probability $p_{cros} = 0.8$, and mutation probability $p_m = 0.01$. The mutation operator treats the complex and elementary genes equally, that is, the probability of mutation of any gene (elementary or superior complex one) into any other such genes is the same. The results shown correspond to the average over 30 different initial seeds. The only exception are the results in Figure 12 that come from a single run, but this run was typical.

Genetic Engineering-based GA Without Advanced Genetic Engineering Operations First we present the results of the computations for the genetic engineering extension of a GA that uses only modified crossover and mutation operators. The typical relationship of the fraction of the complex genes in the total pool of genes (in all the populations' genotypes) against generation number is shown in Figure 12. It demonstrates how the superior complex genes evolve and take over the population. It turns out that the mapping of the evolution process onto the abstract space of complex genes pro-

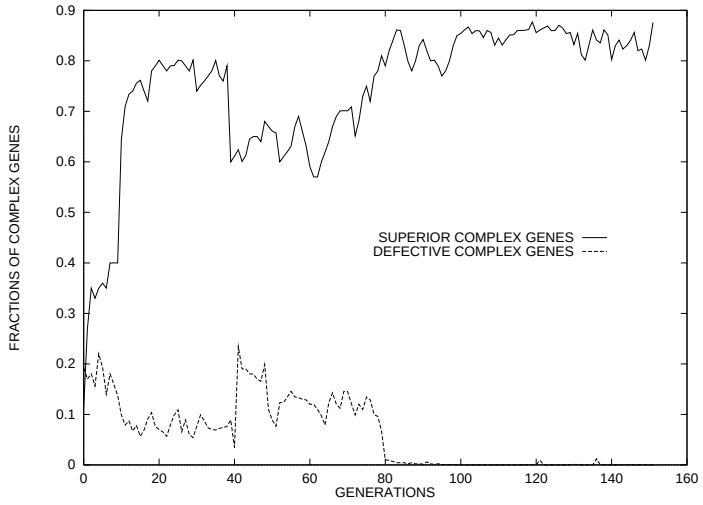

Figure 12: Beam cross-section design problem. Basic genetic engineering extension of GA. The fractions of the superior and defective complex genes in the total pool of genes vs. generation number.

duces a rather complex picture, as shown in Figure 13. The population is redistributed between the different complexities of the genes with some non-trivial underlying processes of the interaction of the subpopulations defined by the complex genes of different complexities.

On average, it takes around 100 generations for the genetic engineering-based GA to find the solution of this problem. The standard GA with the same parameters takes around 400 generations.

Advanced Genetic Engineering Operations The results for the algorithms that use different genetic engineering techniques are shown in Figures 14 to 16. The following parameters have been used: for the radiation/chemical therapy, $p_m^{sup} = 0.0001$ and $p_m^{def} = 0.7$; and $p_{pick} = 0.9$ for the gene surgery.

At first glance, the plots look very similar. They can be divided roughly into four stages. During the first stage, the concentration of the complex genes rapidly increases. During the second stage, it remains at that level for a number of generations. Then at the next stage, it drops and stays at that lower level for approximately the same number of generations, finally jumps to almost the saturation level, and stabilizes there. Possibly this points to some fundamental feature of this problem, independent of the particular type of genetic engineering technique used.

The only plot without the second major feature – the fall in the level of complex gene usage – is Figure 16. The likely cause of this is the relatively crude manner of forcing the superior complex genes into the genetic pool employed by this technique.

6.2.2 Non-positional Royal Road Type Function

This problem has a variable length genotype. Its fitness functions contain the penalty term $30 * (N_{genes} - 64)$ for more than 64 genes in genotype. Each instance of a complex gene $a = \{111\}$ in the genotype contributes 0.1 to the fitness. Adding 2.2 to the fitness rewards each presence of the other complex gene $c = \{\{a\}01\{a\}\} = \{111010111\}$. Since

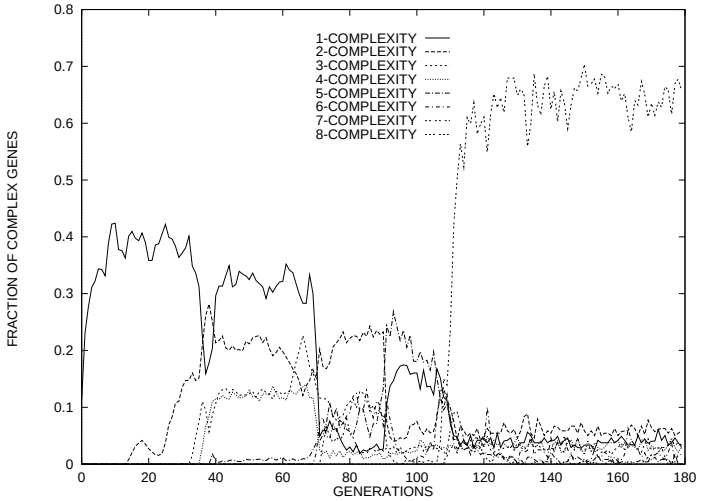

Figure 13: Beam cross-section design problem. Basic genetic engineering extension of GA. The fraction of superior complex genes with different complexities in the total pool of the superior complex genes used to assemble the population vs. generation number.

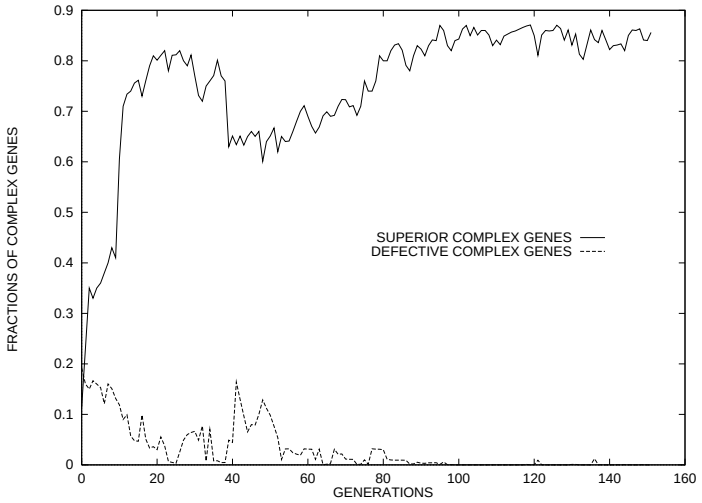

Figure 14: Beam design problem. Use of radiation/chemical therapy. The fraction of the superior and defective complex genes vs. generation number.

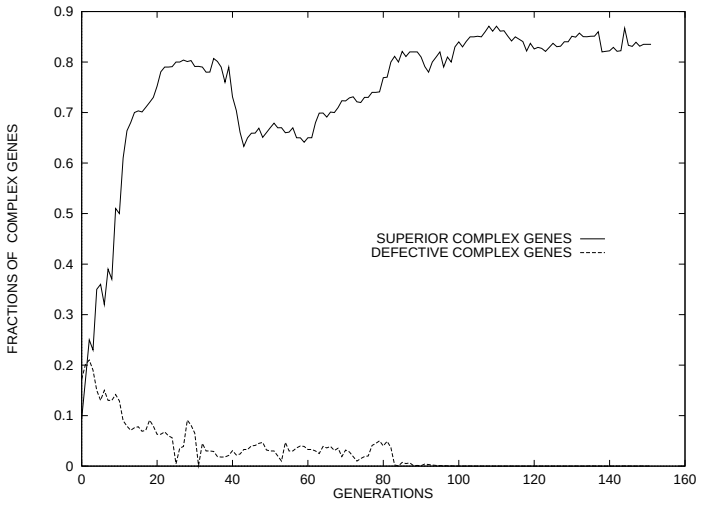

Figure 15: Beam design problem. Use of genetic surgery. The fraction of the superior and defective complex genes vs. generation number.

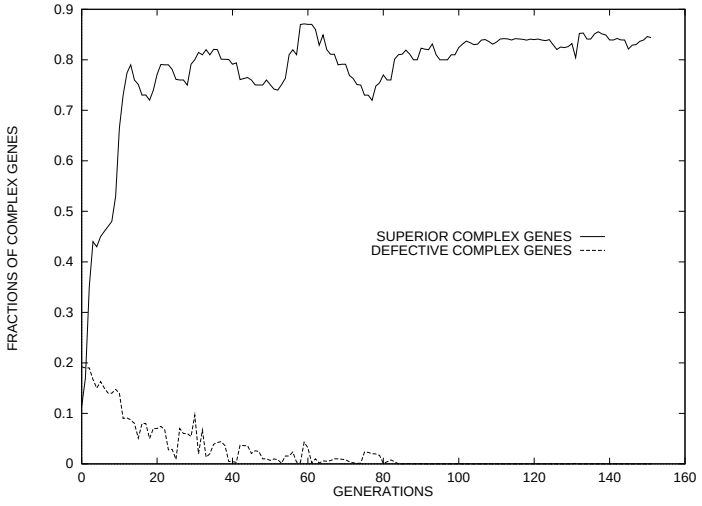

Figure 16: Beam design problem. Use of genetic therapy. The fraction of the superior and defective complex genes vs. generation number.

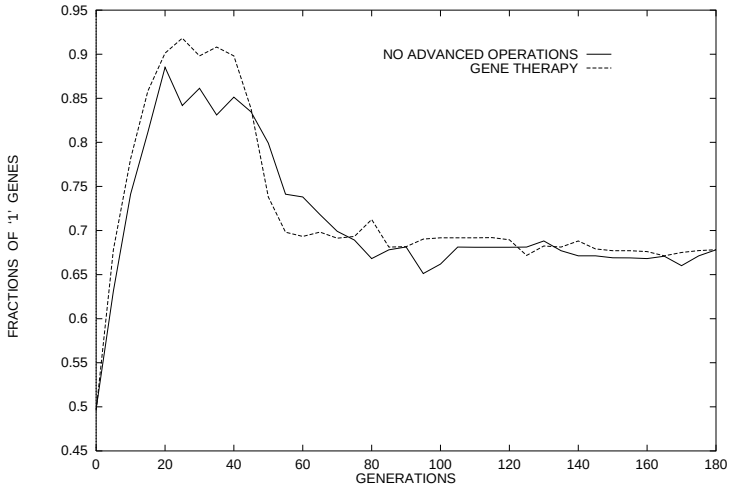

Figure 17: Non-positional royal road function. The fraction of the "1" genes in the genetic pool of elementary genes vs. generation number.

the same part of genotype can be filled with either 2-level complex genes $\{c\}$ or three 1-level complex genes a, these genes compete in the genetic pool. We describe the evolutionary process by plotting the fractions of the elementary genes 1 in the current genetic pool of elementary genes against generation number. When the algorithm tries to improve the population using mostly the complex gene of 1-complexity a, it attempts to saturate the genetic pool of the population's elementary genes with 1s. It tries to do so using the complex gene c, then the concentration of 1s in the genetic pool dips down to approximately $\frac{21}{64}$. Computations have been done using the same version of genetic algorithm with the same parameters as were used for the beam cross-section design problem. The results averaged over 30 runs are shown in Figure 17. Since the problem does not have defective complex genes, the plots for the radiation therapy and gene surgery are very similar to the plot for the genetic engineering-based GA without advanced genetic operations. It takes about 600 generations for the standard GA to find the solution, compared to about 80 for the genetic engineering GA. Such significant computational savings are not surprising since this problem was specially designed to be easy for the genetic engineering-based GA. It possesses the "shiftable" regularity that need not be discovered a number of times such as when the standard GA is used.

7 Discussion

The standard GA employs an indirect, "weak" control of the population. Since all the points in the search space are equal with respect to the rate of reproduction, the response of the population to the current fitness landscape is delayed. The genetic engineering modification of the GA can be interpreted as the introduction of additional control of the population dynamics, which makes it more sensitive to the current fitness landscape. Essentially, this control is just the assignment of different "rates of survival" to different areas in the control hyperspace. This is still a relatively weak control action that affects the population only indirectly. Nevertheless, it provides an additional way of shifting the population into the area of interest in the search space, or doing this more efficiently than is done by the standard GA. In an analogous sense, it is similar to the penalty func-

tion technique, where the search is directed into particular areas of the search space by assigning different weights to the different parts of the search space.

The identification of the complex gene hierarchy is a more complicated problem than the identification of the maximal fitness point. The determination of the hierarchy of the superior and defective complex genes is equivalent to the solution of the inverse problem. The extra processing required to extract and purify the additional information from the data supplied by the standard GA can be computationally expensive. Nevertheless, it is still worth doing if the genetic mapping is sufficiently complex and especially when the information obtained can be reused later to solve similar problems. In order to develop these complex gene hierarchies, the modification of the GA employs multiple solutions of the NP-hard sequential analysis problem of finding the characteristic substrings in the groups of strings. Unlike the complexity of the genetic algorithm, the complexity of this auxiliary problem does not depend on the genotype-phenotype encoding or on the fitness function. It is completely defined by the length of the genotype and the type of genetic encoding (positional or non-positional). Since this auxiliary problem has been studied extensively in genetic engineering, speech recognition, and computer science, a number of highly efficient approximation algorithms have been developed for it.

The genetic coding can be interpreted as a language that describes the objects of interest. The evolved set of the superior complex genes can be considered as a new task-specific representation that is more conducive to solving the problem. The computational process of the genetic engineering extension of the standard GA can be viewed as an iterative procedure that generates increasingly specialized representations and tests them for their ability to solve the problem. The major advantage of GAs is their ability to find improved solutions for optimization problems with poorly understood spaces. The evolved set of complex genes represents the knowledge about these spaces that was acquired by the algorithm.

Genetic algorithms are knowledge-lean processes; they make no use of any domain knowledge in their execution. This gives them robustness and breadth of applicability not found in many other search processes. This advantage is, at the same time, a disadvantage in that often there is considerable knowledge available about the structure of the problem that is not utilized. The process of evolving complex genes can be seen as a process of acquiring and making available increasing amounts of knowledge about the domain under consideration. In this sense, they provide an additional "memory" about the structure of the problem. The effect of this is to gradually turn a knowledge-lean process into a knowledge-rich one without the need to handcraft the acquired knowledge. There are many different evolution paths to this knowledge. This additional knowledge provides benefits to the genetic algorithm. In each class of problem, the knowledge that is represented by and encoded in the evolved complex genes has a meaning within this class; a meaning that is not accessible through the standard GA. This process has the potential to allow the user of such a system to understand the meaning of the complex genes within his or her domain.

The effect of adding genetic engineering to genetic algorithms is to change the evolution path to a solution. The process of complex gene evolution changes the probabilities associated with the fitness landscape by making certain subspaces within the entire search space increasingly likely to be searched. These subspaces are more likely to be of interest than others. The genetic engineering extension of GAs tries to discover the beneficial and harmful genetic material using the analysis of the genetic material available and then exploit this knowledge in order to short-cut the trajectories of the standard GA.

Acknowledgments

This work is directly supported by a grant from Australian Research Council; computing resources are provided through the Key Centre of Design Computing and Cognition.

References

Aho, A., Hopcroft, J., and Ullman, J. (1975). *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts.

Anderberg, M. (1973). *Cluster Analysis for Applications*. Academic Press, New York, New York.

Angeline, P. J. (1994). Genetic programming and emergent intelligence. In Kinnear, K., editor, *Advances in Genetic Programming*, pages 75–98, MIT Press, Cambridge, Massachusetts.

Apostolico, A. (1985). The myriad virtues of subword trees. In Apostolico, A. and Galil, Z., editors, *Combinatorial Algorithms on Words*, pages 85–96, NATO Advanced Study Institute, Series F: Computer and Systems Sciences, Volume 12, Springer-Verlag, Berlin.

Collins, J. F. and Coulson, A. F. E. (1987). Molecular sequence comparison and alignment. In *Nucleic Acid and Protein Sequence Analysis: A Practical Approach*, pages 323–358, IRL Press, Washington DC.

Corcoran, A. L. and Wainwright, R. L. (1994). Chromosome reduction in Genetic algorithms. Technical Report UTULSA-MCS-94-1, University of Tulsa, Tulsa, Oklahoma.

Crochemore, M. (1994). *Text Algorithms*. Oxford University Press, New York, New York.

Forrest, S. and Mitchell, M. (1993). Relative building-block fitness and building block hypothesis. In Whitley, D., editor, *Foundations of Genetic Algorithms*, Volume 2, pages 109–126, Morgan Kaufmann, San Mateo, California.

Gero, J. S. and Kazakov, V. (1996). Evolving building blocks for design using genetic engineering: A formal approach. In Gero, J. S., editor, *Advances in Formal Design Methods for CAD*, pages 31–50, Chapman and Hall, London, England.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. MIT Press, Cambridge, Massachusetts.

Jones, T. C. (1995). A description of Holland's Royal Road Function. *Evolutionary Computation*, 2(4):411–417.

De Jong, K. A. (1975). *An analysis of the behavior of one class genetic adaptive systems*. Unpublished doctoral dissertation. University of Michigan, Ann Arbor, Michigan.

Karlin, S., Dembo, A., and Kawabata, T. (1990a). Methods for assessing the statistical significance of molecular sequence features by using general scoring scheme. *Proceedings of the National Academy of Science U.S.A.*, 87:5509–5513.

Karlin, S., Dembo, A., and Kawabata, T. (1990b). Statistical composition of the high-scoring segments from molecular sequences. *Annals of Statistics*, 18:571–581.

Koza, J. R. (1992). *Genetic Programming*. Addison-Wesley, Reading, Massachusetts.

Louis, S. J., McGraw, G., and Wyckoff, R. O. (1993). CBR Assisted Explanation of GA Results. *Journal of Theoretical and Experimental Artificial Intelligence*, 5(1):21–28.

McCreight, E. M. (1976). A space economical suffix tree construction algorithm. *Journal of Association of Computer Machinery*, 232:262–272.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453.

Rosca, J. P. and Ballard, D. H. (1992). Learning by adapting representations in genetic programming. In Cohen, W. and Hirsch, H., editors, *Proceedings of the Eleventh International Conference on Machine Learning*, pages 407–412, Morgan Kaufmann, San Mateo, California.

Sankoff, D. and Kruskal, J. B., editors (1983). *Time Warps, String and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, Massachusetts.

Schuler, G. D., Altschul, S. F., and Lipman, D. J. (1991). A workbench for multiple alignment construction and analysis. *PROTEINS: Structure, Function, and Genetics*, 9:180–190.

Simon, P. J. (1973). The organization of the complex systems. In Pattee, H. H., editor, *Hierarchy Theory: The Challenge of Complex Systems*, pages 109–127, G. Braziller, New York. New York.

Weiner, P. (1973). Linear pattern matching algorithms. In *Proceedings of the IEEE 14th Annual Symposium on Switching and Automata Theory*, pages 1–11, IEEE Press, Piscataway, New Jersey.

Wu, A. S. and Lindsay, R. K. (1995). A comparison of the fixed and floating building block representation in the genetic algorithm. *Evolutionary Computation*, 4(2):169–193.