Salzburg Interest Group on Integrated Systems

siGis-017-1998 February 17, 1999

ptGAs – Genetic Algorithms Evolving Non–Coding Segments by means of Promoter/Terminator Sequences

The final version of this article has been published in
Evolutionary Computation 6(4): 361–386
published by *The MIT Press* 1999

Helmut A. Mayer

helmut@cosy.sbg.ac.at

Department of Computer Science
University of Salzburg

siGis
Salzburg Interest Group on Integrated Systems

Correspondence to:
Universität Salzburg
Institut für Computerwissenschaften und Systemanalyse
Jakob–Haringer–Straße 2
A–5020 Salzburg
Austria

ptGAs – Genetic Algorithms Evolving Non–Coding Segments by means of Promoter/Terminator Sequences

Helmut A. Mayer
Department of Computer Science
University of Salzburg
A–5020 Salzburg, Austria
helmut@cosy.sbg.ac.at

Abstract

In this article we present work on chromosome structures for *Genetic Algorithms* (GAs) based on biological principles. Mainly, the influence of non–coding segments on GA behavior and performance is investigated. We compare representations with non–coding sequences at predefined, fixed locations with "junk" code induced by the use of *Promoter/Terminator* sequences (ptGA) defining start and end of a coding sequence, respectively. As one of the advantages of non–coding segments a few researchers have identified the reduction of the disruptive effects of *Crossover*, and we solidify this argument by a formal analysis of crossover disruption probabilities for non–coding segments at fixed locations. The additional use of promoter/terminator sequences not only enables evolution of parameter values, but also allows for adaptation of number, size, and location of genes (problem parameters) on an artificial chromosome. Randomly generated chromosomes of fixed length carry different numbers of promoter/terminator sequences resulting in genes of varying size and location. Evolution of these ptGA chromosomes drives the number of parameters and their values to (sub)optimal solutions. Moreover, the formation of tightly linked building blocks is enhanced by self–organization of gene locations. We also introduce a new, non–disruptive crossover operator emerging from the ptGA gene structure with adaptive crossover rate, location and number of crossover sites. For experimental comparisons of this genetic operator to conventional crossover in GAs, as well as properties of different ptGA chromosome structures, an artificial problem from the literature is utilized. Finally, the potential of ptGA is demonstrated on an NP–complete combinatorial optimization problem.

Keywords: genetic algorithms, chromosome structures, non–coding segments, promoter/terminator sequences, building blocks, spontaneous crossover, exon shuffling.

1 Introduction

Genetic algorithms (GAs) are a class of *Evolutionary Algorithms* (EAs) modeling the principles of evolution in order to solve complex optimization problems in science and engineering. Different solutions to a problem form a *Population* of fixed–length bit strings (*Chromosomes*), whose *Fitness* (quality of solution) determines the probability of passing on its genetic material to the next *Generation*. Random *Mutation* of specific bits of chromosomes alter the solution (*Individual*) so as to explore the search space for possibly better individuals. *Crossover* between two (or more) individuals at one or more randomly chosen sites exchanges parts of genetic material potentially combining *Building Blocks* of fit individuals to a solution closer to the optimum (*Building Block* hypothesis (Holland, 1975; Goldberg, 1989)).

The problem with this intuitively convincing concept is that the GA "knows" nothing about the building blocks' structure, as no part of a conventional GA chromosome is more salient than any other, i.e., the chromosome has no explicit structure. Specifically, essential building blocks have the same probability to be destroyed as less important ones (provided they are of same length). Thus, a few researchers have begun to take a closer look at genetic structures in biological systems, specifically, the genetic information encoded on *DNA* (*D*eoxyribo*N*uclein*A*cid) molecules.

In nature, structure (in terms of information content) is primarily induced by coding and non–coding sequences demarcated by promoter and terminator sequences. Within the coding sequences, genetic material can be categorized as (coding) *Exons* or (non–coding) *Introns* [1]. A striking, natural analog to the building block hypothesis is the *Exon Shuffling* hypothesis (Gilbert, 1978; Doolittle, 1978; Gilbert, 1987). It is hypothesized that exons code for functional sub–units of biological structures, and shuffling of already existing partial solutions (exons) could accelerate evolution towards more complex structures.

We loosely transfer these biological principles to the chromosome structures for GAs and term this extension ptGA, where p stands for promoter and t for terminator. In a strict sense, the GA still has no "knowledge" of this new structure, as it treats all sequences the same. However, the disruptive effects of crossover w. r. t. building blocks are reduced with the altered chromosome structure. Moreover, we make the knowledge of this structure available to a new, non–disruptive crossover operator emerging from the chromosome structure containing non–coding sequences. As a consequence, the ptGA utilizing this kind of crossover only combines building blocks (shuffles exons) without destroying them. Still, mutation is able to destroy building blocks, and to act more aggressively than with conventional GA chromosomes, specifically, when mutating a p/t–sequence.

We could additionally limit mutation to coding sequences, but this would contradict our initial non–biological motivation for the use of non–coding sequences. The starting point of this work was the idea of a GA not only opti-

[1] Please, see Section 2.3 for a more detailed explanation.

mizing problem parameter (hereafter, parameter) values, but also the number of parameters (specifically, the number of hidden neurons in an *Artificial Neural Network*). This can be achieved by mutation of p/t–sequences or non–coding sequences resulting in possible destruction or generation of a coding sequence (parameter), respectively. There are a variety of real world problems the ptGA could be applied to, where the (sub)optimal number of parameters is not known in advance.

The article is organized as follows:

In Section 2 the basic information processing aspects of molecular genetics are summarized with specific emphasis on gene structure. We discuss the gene structures in basic groups of biological organisms and the way they are processed in order to build higher–level biological building blocks. We review theories on the purpose and origin of non–coding segments in the genetic code, and conclude with the presentation of the basic principles transferred from biology to machines, specifically, to the proposed ptGA.

In Section 3 we review GA instances allowing for flexible representation of parameters and the development of the use of non–coding regions on GA chromosomes in the past decade.

In Section 4 we conduct experiments with ptGA chromosomes employing non–coding regions at fixed locations by means of a simple example taken from literature (Levenick, 1991). Moreover, we give the results of a formal analysis of the disruption reducing effects of non–coding sequences at fixed locations w. r. t. crossover.

In Section 5 we utilize p/t–sequences allowing not only for evolution of parameter values, but also for adaptation of number, size, and location of parameters on randomly generated fixed–length strings. The distribution of the number of coding segments and their length on randomly generated ptGA chromosomes is analyzed. We introduce a new, non–disruptive crossover operator emerging from the ptGA gene structure in a self–organizing manner and compare this mechanism to conventional crossover in GAs.

In Section 6 we employ the ptGA for the successful treatment of three instances of an NP–complete combinatorial optimization problem. The results are compared to conventional GA performance described in literature (Khuri et al., 1994).

In Section 7 we conclude this article by summarizing the results, referring to partially existing ptGA extensions, reflecting on additional ptGA features to be adopted, and by identifying possible problem domains for future ptGA applications.

2 Molecular Genetics

We briefly summarize the stages of DNA processing representing the most essential information processing aspects of biological systems at the molecular level. Before delving into details, the two most fundamental types of organisms should be introduced. *Prokaryotes* (viruses, bacteria, and blue–green algae) lack a membrane–bound, structurally discrete nucleus and other subcellular compartments. All other organisms are subsumed under the term *Eukaryotes* consisting of cells with a discrete nucleus. Prokaryotes have nearly no non–coding DNA sequences, whereas eukaryotes have lots of these segments.

The majority of the information required to develop a biological system and keep it alive is encoded in genes which are concatenated on DNA molecules – the codebook of life. Two strands of polynucleotides form an antiparallel double helical structure comprising the DNA. The strands contain four different deoxyribonucleotides (bases) building pairs, namely, *Adenosine* (A), *Thymidine* (T), and *Cytidine* (C), *Guanosine* (G). These pairs are joined through hydrogen bonds across the strands resulting in a complimentary structure of the two DNA strands - a duplex(Adams, 1991). Sequences of these base pairs (bp) (typically in the range of $10^2 - 10^4$ bp) constitute a gene containing the assembly instructions for a specific *Protein*. Proteins are the basic functional biological sub–units enabling and regulating all processes, e.g., the transportation of oxygen in blood vessels is accomplished by the iron–containing protein *Hemoglobin*, and the production of proteins itself is supported by proteins.

In prokaryotes DNA forms a single, circular strand, whereas in eukaryotes the complete genome (all genetic information encoded in DNA) is split into packages called *Chromosomes*. The genes needed for the production of cell–specific proteins are transcribed into *messenger RNA* (mRNA). *Transcription* encompasses copying and processing of a specific DNA sequence. This piece of code is then transferred to a *Ribosome* – the protein factory of a cell – where the genetic blueprint is used to construct a protein.

At a ribosome the information encoded in the mRNA is interpreted in chunks (*Codons*) of three bases (triplets), each of the triplets representing a specific amino acid. As the 20 different amino acids can be encoded in 64 ways (4^3 different triplets out of four nucleotides), various triplets code for identical amino acids. The specific amino acid is carried by *Transfer RNA* (tRNA) which has a complimentary triplet (*Anti–Codon*) enabling the docking of the tRNA to the mRNA codon (triplet). The amino acids carried to the ribosome by tRNA interact and combine under the influence of mechanical and electro–magnetic forces to the encoded protein (protein folding). Termination of *Translation* of mRNA code into proteins is signalled by specific triplets (stop codons) not coding for an amino acid.

The complete genome of the bacterium *Escherichia Coli* (E. Coli) – the workhorse of prokaryotic geneticists – consists of 4.5×10^6 bp, whereas the human genome contains 4.0×10^9 bp which approximately translates to a storage capacity of 953 MB in terms of computer science. This is roughly the amount of data stored on a hard disc of todays personal computers. It is extremely fascinating (at least for a non–biologist) that we carry this huge amount of information in every single cell. Even more interesting is that as little as 9% of the human genome might be coding DNA (Smith, 1989).

2.1 Gene structure and transcription

For brevity and less complexity we will only focus on prokaryotes. E. Coli is a prokaryote with a genome of approximately 5000 genes arranged along a single, circular DNA molecule. Most genes are present as single copies, and the genes for proteins used for a specific biological process are clustered on the DNA simplifying the transcription process, as these genes can be transcribed at once. Prokaryotic genes are *co–linear* which means that the genetic code in a gene on the DNA is identical to the code processed by the ribosome, i.e., there are no intervening, non–coding sequences or *Introns* as in eukaryotes. Transcription is achieved by *RNA Polymerase*, a single, very complex protein consisting of several functional sub–units.

Promoter and *Terminator Regions* on the DNA enable RNA polymerase to detect the beginnings and ends of genes to transcribe. The canonical promoter region in E. Coli is defined by a sequence pattern called *Pribnow Box*, two chains of six nucleotides at positions -35, $TTGACA$) and -10, $TATAAT$ (Figure 1). The position indices refer to the first nucleotide transcribed from DNA to mRNA which by definition is $+1$. As with most nucleotide sequences these promoter sequences are not uniquely defined. By defining a promoter as a combination of position and sequence the redundancy of the start information is increased and the detection of the promoter region becomes more reliable, i.e fault–tolerant.

The encoding of the terminator region is accomplished by two schemes. Either the terminators are certain nucleotide sequences at the end of the gene (*Downstream* [2] is the more correct biological term) causing RNA polymerase to dissociate from the DNA strand, or the termination of transcription is modulated by a protein factor known as *Rho* (ρ) *Factor* (Figure 1). Basically, three types of regulation at the transcription level can be discerned, namely, repression of transcription (negative control), activation of transcription (positive control), and attenuation of transcription (complex translation–transcription interaction).

The gene structure of eukaryotes is similar, but location and structure of promoters and terminators are of greater variety and complexity. The main difference to prokaryotes in terms of genetic code are introns, i.e., non–coding sequences which are found within and separate the parts of a single gene.

2.2 On the role of introns

Although to a small extent introns can be also found in prokaryotes and viruses, they are typical for eukaryotes. This seemingly waste of resources must have some benefits, assuming that evolution tends to favor processes consuming less energy. As intron processing and splicing of the exons requires additional energy (the permanent replication of "junk" (non–coding) DNA even more), there have to be advantages to compensate for these drawbacks. Not only do introns play a role in regulation of gene expression by containing promoter and enhancer sequences, they also enable a variable combination of exons they separate, a

[2] We will also transfer this terminology to ptGA chromosomes.

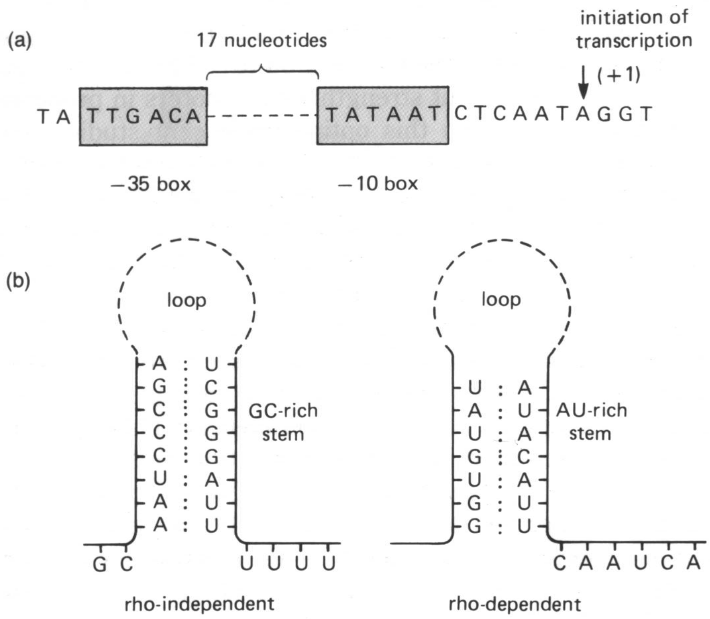

Figure 1: Promoter and terminator sequences in prokaryotes. (a) Consensus sequences for E. Coli promoters, (b) rho–dependent and rho–independent terminator sequences (taken from (Beebee and Burke, 1992) by permission of Oxford University Press).

process called *Exon Shuffling* (Gilbert, 1978; Doolittle, 1978; Gilbert, 1987). If the exons are thought of as gene modules encoding functional or structural subunits of proteins (this does not seem to be true for all eukaryotic genes), special modules could be used in a variety of genes and could be inserted by recombination processes (Figure 2). *Alternative Splicing*, where only subsets of exons are transcribed, is another advantage, since a single gene can encode a variety of proteins. One of the central questions in evolutionary genetics is, whether prokaryotes lost introns (*Introns–Early* hypothesis), or eukaryotes acquired introns (*Introns–Late* hypothesis) in the course of evolutionary development of life (Cavalier-Smith, 1991).

The introns–early hypothesis assumes that introns could be already found in the very first gene structures and separated the code for structural and functional domains of proteins. Following these ideas, prokaryotes have optimized their genome by eliminating introns in order to replicate efficiently (high growth rates) for the price of loosing flexibility. The main mechanism of intron evolution in eukaryotes is intron–loss which could be explained with *Retroposition*, a process where parts of spliced mRNA is re–transcribed into *Complimentary DNA* (cDNA) by *Reverse Transcriptase* (Fabry, 1995). Another convincing argument for the introns–early hypothesis is based on the likely theory that DNA is an

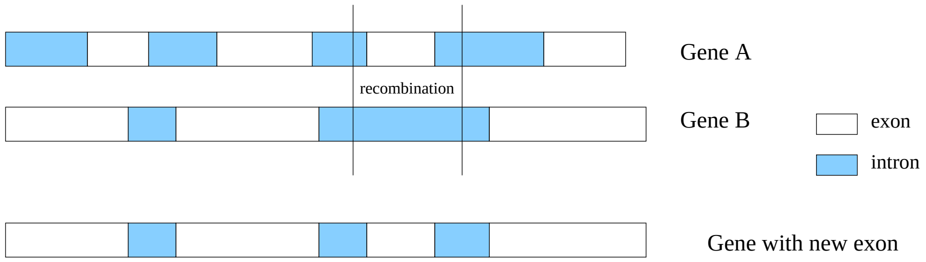

Figure 2: Exon transfer by inserting a gene module from gene A to an intron region in gene B through recombination.

evolutionary product of RNA (deDuve, 1994). Thus, in early cell structures the genetic information was encoded in rather short RNA strands. These old RNA forms seem to have had self–splicing capabilities (Sharp, 1987) in order to cut out introns, a process found with many modern RNA types. Moreover, the complex protein factors involved in splicing would have had to be present before the insertion of introns (assuming introns–late), but it is extremely unlikely that this machinery evolved just by chance (deDuve, 1994).

One of the problems with the introns–early hypothesis is the position of introns in homologous eukaryotic genes. The small variance of intron locations is difficult to explain with loss of introns, as different species would have had to lose the same introns at identical positions, and this process is of extremely small probability. Similarly, simultaneous mutation of exon and intron bases could have shifted the intron positions *Intron–Sliding* by the observed magnitude, but, again, the probability for these slidings to the correct positions tends towards zero. (Rogers, 1989).

The intron–late hypothesis states that ancient gene structures contained no introns, and these have been inserted in the course of evolution. Studies on a set of well conserved genes of higher eukaryotes showed no correlation between intron positions and structural or functional properties of the encoded protein, but could identify identical intron positions in homologous genes of related organisms. These results favor the idea of evolutionary insertion of introns which could be explained by *Reverse Splicing* of mRNA, where sequences are inserted and re–transcribed by reverse transcriptase (Fabry, 1995).

Just to give an idea of the quantitative proportions of coding to non–coding DNA, we present the percent of genome coding for proteins for some typical species in Table 1 (Smith, 1989).

As outlined above, the only prokaryote listed in Table 1 uses all of its DNA to encode proteins, whereas the eukaryotes have a surprisingly high content of non–coding DNA in their genome. Following all the debates of geneticists about the evolutionary advantage of non–coding DNA regions, we will in detail investigate the usefulness of non–coding regions with artificial chromosomes for GAs in subsequent sections. Evidently, when looking at the evolution of GAs, the intron–late hypothesis applies.

Species	Genome Size (bp)	Protein Coding DNA (%)
Escherichia coli	4.5×10^6	~ 100
Drosophila melanogaster	1.5×10^8	33
Homo sapiens	4.0×10^9	$9 - 27$
Protopterus aethiopicus	1.42×10^{11}	$0.4 - 1.2$
Fritillaria assyriaca	1.27×10^{11}	0.02

Table 1: Coding DNA content per haploid genome (basic data from (Cavalier-Smith, 1985), edited with data from (Lodish et al., 1995)).

2.3 ptGA chromosome structures

We present the main genetic mechanisms modeled by ptGA, and clear some issues concerning terminology. The biologically motivated extensions to conventional GAs are discussed briefly.

The main difference from standard GAs is the representation of parameters on the chromosome. Usually, the parameters are encoded at fixed locations on the chromosome, and the number and size is defined by the user. This is certainly sufficient for problems where the number and resolution of parameters is known in advance. However, for a variety of real–world problems this knowledge is not existent. It might be argued that natural evolution faced the same problem, thus it evolved a mechanism where the number of genes can be adapted to specific environments. Hence, we transfer the concept of promoter/terminator sequences (regions) to GAs to allow for the evolution of number, size and location of parameters. Throughout this article, we will mostly use the abbreviation *p/t–sequences*.

Frequently, a specific bit of a chromosome is termed "gene" in GA literature. However, a wild type gene is a complex structure, and we think that the term "base" would be more appropriate, as it reflects the atomic unit of both the genetic and computer code.

On a ptGA chromosome a sequence of bases p_s of a specific length p_l is defined to be the p–sequence. Similarly, the t–sequence is specified by t_s and t_l. The p/t–sequence bases are given in binary string notations (e.g. $p_s =$ "010"). The bases downstream from a promoter to the terminator represent a gene. The bases upstream from a promoter to the next terminator are a non–coding sequence. A gene is identical to its only exon which is associated with one parameter, thus, we will use the terms "gene", "exon", and "(problem) parameter" alternatively. Evidently, the usage of the terms "introns" and "exons" for non–coding and coding sequences, respectively, is not conforming to exact genetic definitions. However, by now a ptGA chromosome contains a single, unambiguous instance of non–coding segment, and we will use the terms "exon" to refer to coding regions and the term "intron" to refer to non-coding regions of a chromosome. We hope that geneticists will forgive this simplification [3]. The

[3] See also the differences between GA chromosomes and natural chromosomes.

base sequences of exons or introns will often be abbreviated as e/i–sequences.

In figure 3 an example chromosome with exons and introns induced by p/t–sequences is depicted.

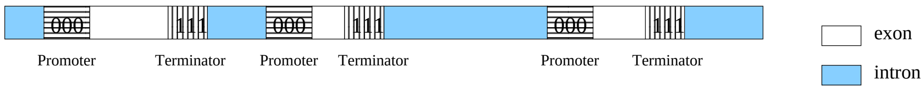

Figure 3: A ptGA chromosome with $p_s =$ "000" and $t_s =$ "111".

The only limit to the number of exons being possibly evolved by the use of p/t–sequences is the chromosome length l. Obviously, nature is also evolving the length of chromosomes (genome sizes), and we could incorporate that into ptGA, too. However, in this work we restricted experiments to chromosomes of fixed length. The process of searching for p/t–sequences and location of exons is termed transcription. More ideas could be borrowed from nature by implementing a simple form of repression and activation of gene (exon) expression (Section 2.1). E.g., a single base upstream the promoter could indicate, if the exon is to be transcribed. This mechanism might create an even more flexible representation of possible solutions.

3 Related Work

In this section we briefly survey some ideas and suggestions of specific GA research dealing with parameter location on chromosomes, adapting gene size, or changing the number of parameters during the course of algorithmic evolution. These contributions can be seen as steps towards the use of non–coding base sequences (introns) in GAs, and we review results of the first researchers having proposed artificial chromosomes containing introns.

3.1 Pre–intron GAs

The *ARGOT Strategy* (Shaefer, 1987) (Adaptive Representation Genetic Optimizer Technique) employs a form of *Lamarckian* evolution, as specific knowledge about the population is exploited to change the representation of parameters. In this scheme "roving boundaries" are used to define the space (the number of bases) for a parameter on a chromosome. These roving boundaries are constrained by "brick wall" boundaries limiting the size of a parameter. One of the main ARGOT specific operators is *Increase/Decrease Resolution*. If the population has converged for a parameter, i.e. the base values (alleles) are nearly identical across the population, the resolution of that parameter is increased by adding bases to its representation on the chromosome. Thus, ARGOT performs a finer search of the parameter space coming closer and closer to the optimal (real) value. If the parameter exhibits a highly random representation, the resolution is decreased so as to locate regions of optimal parameter values. The roving boundaries can be interpreted as promoters and terminators defining the size of a gene and the resolution of a parameter value.

In (Bui and Moon, 1994) the notion of *C–Schemata* was introduced. The authors observed that the position of defined bases in a schema of defining length δ have an impact on the destructiveness of k–point crossover (multi–point crossover). If the defined bases are clustered, the length of the sequence of don't–care symbols ('*', non–defined bases) increases and makes it more likely that crossover sites fall in these regions without destroying the clustered bases. In (Bui and Moon, 1994) a schema preprocessing heuristic (graph reordering) was used to construct chromosomes with clustered schemata. The results for graph bisection problems have been superior to standard GA results. Again, this is a predecessor of GAs with intron base sequences, where the clustered bases can be loosely interpreted as exons and the bases between clusters as introns. However, there are no non–coding sequences used, but building blocks are clustered by heuristics prior to GA runs dividing the chromosome into sequences with higher and lower density of building blocks.

Messy GAs (Goldberg et al., 1993) allow variable–length chromosomes and relax the fixed–locus assumption of conventional GAs. An example messy chromosome of length $l = 3$ has the following structure: ((10)(21)(11)), where the first number of each pair of values indicates the specific base and the second number its value. In this example base 1 occurs twice with two different values, thus, the messy chromosome is overspecified. This conflict is resolved by a first–come–first–served rule scanning the genotype from left to right. Hence, in the above example, the value 0 would be assigned to gene 1. A more severe problem arises with underspecification, especially, for parameter optimization problems over some fixed number of parameters. This can be solved by using *Competitive Templates* representing a previously found solution. The unspecified bases are filled in by this template (Goldberg et al., 1993). The recombination operator with messy GAs is splice and cut. The cut operator surprisingly cuts a chromosome at a randomly chosen position. The splice operator concatenates two genotypes which do not necessarily have to be cut. At the begin of messy GA runs the splice operator is assigned a higher priority to increase chromosome lengths, whereas in later stages of evolution the cut operator becomes more likely. Messy GAs have been applied successfully to the evolution of fuzzy controllers (Hoffmann and Pfister, 1995). This GA variant allows for the evolution of both the number of parameters and their location, whereas the length of parameters on the chromosome is fixed. Messy chromosomes also contain a form of introns, as overspecified genes being not expressed can be viewed as non–coding regions.

3.2 First intron GAs

Levenick (1991) reported on experiments with binary strings containing five genes with a length of six bases ($e_l = 6$) each being separated by introns of length seven ($i_l = 7$). He used a fitness function giving increasing credit for discovery of single genes and specific combinations of them. A credit is only assigned if all the gene's bases have the correct value (Levenick, 1991). This procedure is very similar to the credit assignment for building blocks in the *Royal Road Functions* (Mitchell et al., 1991). Comparing GA runs with and without insertion of introns showed a nearly tenfold increase in successful evolution of optimal individuals. Levenick (1991) explains that behavior with the increase of crossover sites (he used 1–point–crossover) which do not break a gene when the crossover site is located in the intron region.

The Royal Road functions (Mitchell et al., 1991) are specifically designed functions which should lay out a "royal road" for a GA, as the function values are hierarchically organized and lead the way to the optimum. This assumes the building block hypothesis (Holland, 1975; Goldberg, 1989), that new schemata are discovered via crossover which combines low–order schemata (building blocks) to higher–order schemata of increased fitness (composite solutions).

In her dissertation Wu (1996) centers on experiments with Royal Road functions with chromosomes containing non–coding regions (Wu, 1996). The use of introns at fixed locations resulted in only slight (or no) increase of GA performance on the Royal Road functions. She also investigated the use of start sequences (besides many other biologically motivated ideas, e.g., multiple gene copies, reading frames, overlapping genes) in order to freely position the basic building blocks of the Royal Road function on the chromosome. This increased GA performance remarkably, and her thesis is to our knowledge the first work proposing what we call promoter sequences.

In *Genetic Programming* (GP) (Koza, 1992) "junk" code (or *Bloat*) emerges intrinsically. A program statement like `a := a + 0;` can be destroyed by crossover without changing the fitness of a program. In (Nordin et al., 1995) it is demonstrated that GP introns (bloat) play substantial roles in preserving building blocks and reducing the disruptive effects of crossover.

4 Introns at Fixed Locations

So far, it has been implied that a main benefit of introns in GAs is the reduction of the disruptive effects of the crossover operator. If the selected crossover site(s) fall into an intron region, the exchange of genetic material cannot break up a schema in a gene, unless one gene coding for a single value is separated by intron regions. In the latter case the intron region serves as a crossover hot spot increasing the probability for crossover to occur in that region and split the schema preferably at this site. In order to analyze the effects of intron base insertion more formally, let us give some definitions.

The *Exon Base Ratio* e' and the *Intron Base Ratio* i' are given by

$$e' = \frac{e}{l} \qquad i' = \frac{i}{l} \qquad e' + i' = 1, \tag{1}$$

where e is the number of exon bases, i the number of intron bases, and $l = e + i$ the chromosome length. It should be noted here that we will use e

and i in a different context in following sections of this work, namely e as the number of exons (encoded parameters) and i as the number of introns (non–coding base sequences). We unify these definitions by thinking of the number of exon (intron) bases as exons (introns) of constant length $e_l = i_l = 1$. Hence, using that definition in this section the number of exons is equal to the number of exon bases.

4.1 Crossover disruption probabilities

Let us start with simple 1–point crossover, where the probability of destroying a schema of defining length δ is given by

$$p_d^{(1,0)} = p_c \frac{\delta}{l-1}, \tag{2}$$

where $p_d^{(1,0)}$ stands for the disruption probability of 1–point crossover without intron bases. This probability is only an approximation, as a broken schema can be repaired by genetic material from the other parent ((Bridges and Goldberg, 1987) gives a more detailed analysis of the crossover disruption probability). A chromosome without introns has a length $l = e$, i.e., the genotype only contains exon bases. With the insertion of introns the length of the chromosome is increased to $l^{(i)} = e + i$, thus, we get disruption probabilities without and with intron bases

$$p_d^{(1,0)} = p_c \frac{\delta}{e-1} \qquad p_d^{(1,i)} = p_c \frac{\delta}{e+i-1}, \tag{3}$$

and by building the ratio of disruption probabilities we get

$$\frac{p_d^{(1,0)}}{p_d^{(1,i)}} = 1 + \frac{i}{e-1}, \tag{4}$$

and further transformation using $l^{(i)}$ yields

$$\frac{p_d^{(1,0)}}{p_d^{(1,i)}} = 1 + \frac{i'}{e' - \frac{1}{l^{(i)}}}. \tag{5}$$

With $e' >> \frac{1}{l^{(i)}}$ which is a reasonable approximation for most chromosomes containing at least some exon bases we arrive at a ratio of

$$\frac{p_d^{(1,i)}}{p_d^{(1,0)}} = e' = 1 - i'. \tag{6}$$

Equation 6 simply states that for a given defining length $\delta < l^{(i)}$ the disruption probability of 1–point crossover applied to chromosomes with non–coding segments decreases linearly with the exon base ration e' reducing the disruptive effects of crossover. W. r. t. mutation the insertion of intron bases does not influence the expected number of mutated exon bases, when the mutation operator is the standard bit–flip with fixed mutation rate p_m. It should be noted that we were not concerned with the specific distribution of intron bases on the chromosome in the above equations. Obviously, a schema containing intron bases increases the defining length δ and the probability of crossover within it, thus, the building blocks should be tightly packed secured by surrounding sequences of intron bases to minimize the probability of disruption. As a matter of fact, this is exactly how the gene structure of biological systems is organized and we mimic this chromosome structure with ptGA.

Next, the disruption probability for chromosomes with fixed intron bases should be generalized to k–point crossover. For a schema to be preserved all k crossover sites must not be in the base sequence defining the schema. Hence, for the disruption probability of k–point crossover we get

$$p_d^{(k)} = 1 - (1 - p_d^{(1)})^k, \tag{7}$$

and the ratio of the disruption probability for a chromosome with intron bases $p_d^{(k,i)}$ to $p_d^{(k,0)}$ without intron bases using Equation 6 becomes

$$\frac{p_d^{(k,i)}}{p_d^{(k,0)}} = \frac{1 - (1 - e' p_d^{(1,0)})^k}{1 - (1 - p_d^{(1,0)})^k}. \tag{8}$$

Assuming $p_d^{(1,0)} << 1$ (implying $\delta << l^{(i)}$) we may use the first order Taylor series expansion $(1-x)^n \approx 1 - nx$ which simplifies Equation 8 to

$$\frac{p_d^{(k,i)}}{p_d^{(k,0)}} = e' = 1 - i', \tag{9}$$

hence, the disruption probability of building blocks is (approximately) [4] linearly decreasing with e' independent of the number of crossover sites k, when comparing chromosomes with and without intron bases.

Nice as this may be, for most real world problems the number, size and location of building blocks is not known in advance, hence, the position of intron bases cannot be determined. Therefore, it would be desirable to additionally evolve location of intron and exon bases. In analogy to biological systems and their genetic code packed in DNA strands we will induce this building block evolution by promoter/terminator sequences defining begin and end of exons and interspersed introns. But before we analyze and discuss this more complex instance of the ptGA, we review an experiment from (Levenick, 1991).

4.2 ptGA parameters

The following ptGA parameters remain unchanged for all experiments in this article:

[4] Numerical investigations of the function in Equation 8 exhibited a very good approximation by Equation 9 for a wide range of values.

- Population size $n = 100$.

- Mutation rate $p_m = \frac{1}{l}$.

- Selection method: Binary tournament without replacement.

The specific setting of the mutation rate is based on theoretical work (Bäck, 1992; Mühlenbein, 1992) giving strong evidence that this rate is an appropriate choice. Other parameters vary and will be given explicitly.

4.3 Evolution of L–birds

In (Levenick, 1991) the author experimented with a genotype containing five genes (exons) of length $e_l = 6$ being separated by a sequence of intron bases with length $i_l = 7$. In order to illustrate this example genotype and to explain the specific fitness function used, Levenick (1991) chose the evolution of a bird with rather simplified gene structure. He writes:

> Imagine the following: A is a gene for elongated finger bones (advantageous for extracting food from crevices), B is a gene for feathers (for warmth), C is for skin flaps between fingers (again for warmth – widespread fingers with skin flaps could be used to better cover young), D is for hind legs capable of a few good jumps (for catching prey), and E is for clutching feet (for holding prey).
>
> The existence of any of these genes is somewhat advantageous, the coexistence of genes D and E is more advantageous (can jump from branch to branch and hang on), genes for A, B, and C together is even more advantageous (can wrap self or young in a feathered cover; can flap and so jump farther), and all five attributes together allows the creature to fly!

A gene is only said to be developed, if all the gene's bases have the correct value (of 1). This procedure is very similar to the credit assignment for building blocks in the Royal Road Functions (Mitchell et al., 1991; Wu, 1996). Comparing GA runs with and without insertion of introns showed a nearly tenfold increase in successful evolution of optimal individuals (Levenick, 1991). In the following we will refer to that problem as *L–bird* (Levenick bird) problem. The specific fitness function is given in Table 2.

Each developed gene contributes to the L–bird fitness, and the non–linear credit for coexistence of specific genes introduces *Epistasis* to the problem. In Figure 4 the average number of generations $\overline{g}$ to evolve to a fully developed L–bird is shown over the intron base ratio i'. The introns are inserted between genes and each sequence of intron bases has the same length, e.g., insertion of ten intron bases in between the five genes (six bases each) gives $i' = 0.57$ ($\frac{4\times10}{4\times10+5\times6}$). The order of genes (gene structure) is $ABCDE$. The crossover rate is $p_c = 1.0$. The mutation of any (fixed) intron base has no influence on the fitness, hence the mutation rate is based on the number of exon bases (30)

Fitness	Genes
1	None
5	$A \vee B \vee C \vee D \vee E$
50	$D \wedge E$
100	$A \wedge B \wedge C$
1000	$A \wedge B \wedge C \wedge D \wedge E$

Table 2: Fitness function values for various gene combinations for the L–bird problem (a fitness of 1000 indicates the optimum), from (Levenick, 1991).

yielding $p_m = \frac{1}{e} = 0.0333$. This gives a constant mutation probability of an exon base allowing comparisons independent of intron base ratios.

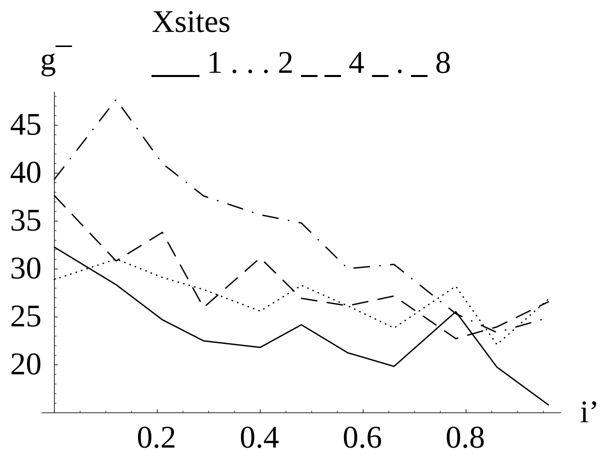

Figure 4: Average number of generations $\overline{g}$ to evolve to an L–bird with gene structure $ABCDE$ for various number of crossover sites $k = 1, 2, 4, 8$ over intron base ratio i' (averaged on 50 runs, i' recorded in steps of ≈ 0.1).

As the gene structure seems to be amenable to the mechanism of 1–point crossover (the two development stages ABC and DE can be directly combined by recombination at one crossover site), we also investigated 2–, 4–, and 8–point crossover for comparisons. Generally, all four crossover operators exhibit a decrease of average generations when the intron base ratio increases. Also, for constant intron base ratios the number of average generations decreases with the number of crossover sites. In the light of the previous analysis (Section 4.1) this behavior is reasonable, as a lower number of crossover sites increases the probability $p(k)$ that none of the sites fall into exon base regions ($p(k) = i'^k$). With high intron base ratios the difference of $p(k)$ for variable k becomes smaller (most crossover sites will be in intron base regions) as does the difference in average generations in Figure 4. Nevertheless, 1–point crossover still improves performance at $i' > 0.8$ which could be credited to the assumed synergy of this operator and the specific gene structure.

Following these arguments, it could be expected that an L–bird gene structure with non–adjacent genes of a specific development stage exhibits very similar behavior w. r. t. the number of crossover sites. In Figure 5 the previous experiment has been run with a different arrangement of genes ($ADBEC$).

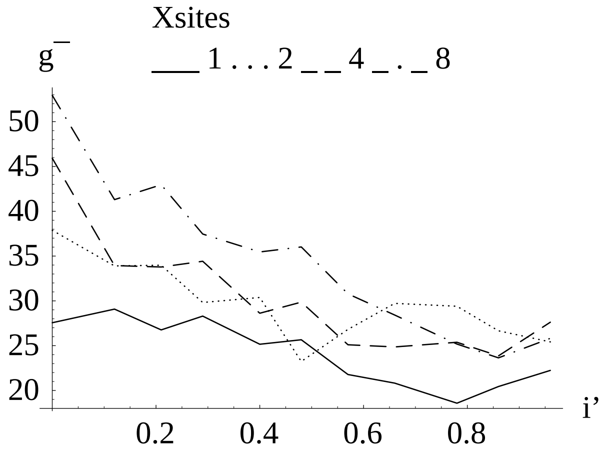

Figure 5: Average number of generations $\overline{g}$ to evolve to an L–bird with gene structure $ADBEC$ for various number of crossover sites $k = 1, 2, 4, 8$ over intron base ratio i' (averaged on 50 runs, i' recorded in steps of ≈ 0.1).

Though cooperating genes are spread out on the chromosome, the average number of generations to reach the optimum is comparable to the gene structure $ABCDE$. Varying intron base ratios and different crossover operators have the same general influence on the optimum finding capabilities. The performance of the investigated number of crossover sites again converges at high intron base ratios including 1–point crossover, as the modified gene structure does not favor this specific operator.

Based on these experiments it can be stated that the insertion of introns at predefined, fixed locations is able to accelerate evolution. However, considering more realistic problems, the insertion of non–coding sequences at predefined sites separating building blocks (exons) is hardly possible, as number, size, and location of building blocks is mostly not known in advance. For this reason we will make use of p/t–sequences for the evolution of L–birds in the next section, where the ptGA allows for evolution of number, size, and location of genes (exons).

5 Evolution of Introns

First, we introduce some more definitions. Let us begin with the *Marker Length* m_l given by

$$m_l = p_l + t_l \ , \tag{10}$$

with the p–sequence length p_l, and the t–sequence length t_l. The *Normalized Chromosome Length*

$$\bar{l} = \frac{l}{\overline{m_l} + \overline{e_l}} \tag{11}$$

should serve as an estimator for the maximum possible number of exons for a given chromosome length l, an average marker length $\overline{m_l}$ with an average exon length $\overline{e_l}$. As we will not be concerned with varying marker lengths on the same chromosome, we set $\overline{m_l} = m_l$. In the specific case of the L–bird problem the exons (genes) are of constant length, hence, we may write $\overline{e_l} = e_l$, and the normalized chromosome length becomes

$$\bar{l} = \frac{l}{m_l + e_l}. \tag{12}$$

Equation 12 implies that a p/t–sequence without any exon bases in between is also counted as an exon. For a problem, where the number of exons (parameters) is known in advance, the *Minimal Length* of a chromosome is given by

$$l_{min} = \bar{l}(m_l + e_l). \tag{13}$$

As the chromosomes should offer more space than the minimal length, we further introduce a *Space Factor* s

$$s = \frac{l}{l_{min}} \tag{14}$$

which simply gives the ratio of actual to minimal chromosome length. Intuitively, it should be easier for the ptGA to find the optimum, if the chromosomes are longer than the minimal length required.

Looking back to Equation 1 for the intron and exon base ratios (where i and e are the number of intron/exon bases), and counting p/t–sequences as exon bases (as they are an essential part of an exon), we may write

$$e + i = l \Rightarrow l_{min} + i = s\, l_{min}, \tag{15}$$

assuming that l_{min} is the space needed to encode the solution. Dividing Equation 15 by l gives the intron base ratio

$$i' = 1 - \frac{1}{s} \qquad s \geq 1.0. \tag{16}$$

Again, it should be emphasized that these definitions can be made only when we know the optimum (optimal number of exons and their length) in advance. This is only true for the simple test problems used to analyze various effects of the ptGA chromosome structure, but cannot be said of more complex problems ptGA is laid out for.

5.1 Randomly generated ptGA chromosomes

Before we continue with our test problem, we present empirically found distributions of the number of exons and the exon length (e–sequence without p/t–sequences) in randomly generated start populations. Each base has been generated with equal probability $p(0) = p(1) = \frac{1}{2}$. We used the *Pseudo Random Number Generator* (PRNG) implemented with SGA-C (Smith et al., 1991) employing a subtractive method taken from (Knuth, 1969). This PRNG displayed a departure from randomness for specific choices of p/t–sequences. Basically, the probability of two adjacent bases of different values was higher than that of identical values. However, other commonly used PRNGs exhibited a very similar tendency (Mayer, 1997).

Figure 6 shows exon (length) statistics extracted from randomly generated ptGA chromosomes with p/t–sequence lengths $p_l = t_l = 2$.

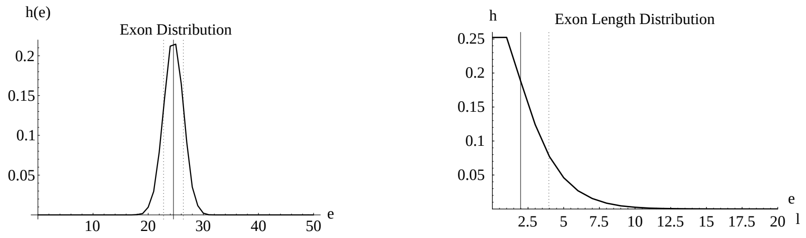

Figure 6: Distribution $h(e)$ of the number of exons (left) and $h(e_l)$ of the exon length (right) for $\bar{l} = 50$ and p/t–sequences $p_s = t_s =$ "01" (averaged on 10,000 chromosomes, the discrete distribution is presented as a continuous graph).

The *Maximum–Likelihood* estimators of above data for mean and standard deviation support evidence that the number of exons obeys a *Gaussian* distribution and the exon length an *Exponential* distribution. The latter corresponds very well with a result in genetics (Gilbert and Glynias, 1993), where the authors state that the exon length is exponentially distributed, when introns are inserted randomly into a gene. We did essentially the same, as the random generation of p/t–sequences induces a (random) separation of exons (coding sequences) and introns (non–coding sequences) on the chromosome.

5.2 Chromosomes with fixed p–sequences

We will now analyze the behavior of ptGA gene structures with p–sequences inducing evolution of non–coding sequences. Recalling the problem statement of L–bird evolution in Section 4.3, the fully developed L–bird genotype contains the five genes (exons) $ABCDE$. The tag of the exon has been determined by its location with the experiments in Section 4.3, and location is now subjected to evolution. This requires an additional exon tag next to the exon's p–sequence for the following experiments. This tag determines which of the five genes is expressed by a specific exon.

By now, we will focus on the mere use of p–sequences, as the exon length is constant with the L–bird problem (3 tag bases + 6 parameter bases). Table 3 gives the exon tags for the L–bird problem specifying the five different genes.

Gene	Tag
A	001
B	010
C	011
D	100
E	101

Table 3: Gene tags specifying L–bird genes.

With a p–sequence $p_s =$ "00", a marker length $m_l = p_l = 2$, an exon length $e_l = 9$, and $\bar{l} = 5$ L–bird genes we get a minimal length of a chromosome $l_{min} = 55$. Figure 7 shows the average number of generations $\overline{g}$ to evolve a fully developed L–bird for various numbers of crossover sites, and space factors s ranging from 1.5 to 6.0. The fitness function (Table 2) remains unchanged. Multiple occurrence of identical genes is possible, but the fitness is only determined by the (non–)existence of a specific gene.

Consistently, the average number of generations $\overline{g}$ decreases with increasing space factor s, and at $s \geq 4.0$ (corresponding to an $i' \geq 0.75$, Equation 16) $\overline{g}$ is even less than with introns at fixed locations (Figure 4). This behavior is remarkable, because the discussed gene structure requires 55 bases to evolve to the correct value, whereas it takes only 30 with the simpler structure. As with fixed intron bases, the weaker performance of crossover operators with increasing number of crossover sites seems to stem from the increased probability of crossover sites falling into p–, or e–sequences, where the disruption probabilities are higher than in i–sequences. Again, this difference in performance is reduced with increasing space factor.

As mutation on its own is more aggressive than with conventional gene structures (mutation of a p–sequence base destroys the complete exon, mutation of an i–sequence base might create a new exon), we introduce a crossover operator without disruptive effects by exploiting the knowledge on the position of e/i–sequences on a ptGA chromosome.

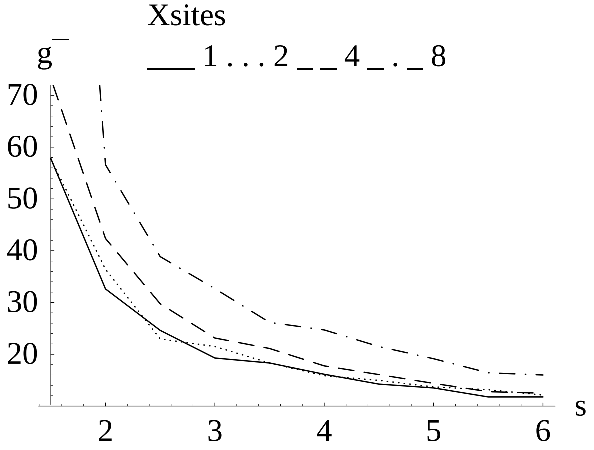

Figure 7: Average number of generations $\overline{g}$ to evolve to an L–bird for various number of crossover sites over chromosome length, $p_c = 1.0$, $p_m = \frac{1}{l}$, $p_s =$ "00", and $t_l = 0$ (averaged on 50 runs, space factor s recorded in steps of 0.5).

5.3 Spontaneous crossover

Whereas conventional k–point–crossover selects crossover sites randomly with equal probabilities for all sites on the chromosome, thereby, possibly destroying p/t–sequences, *Spontaneous Crossover* only selects sites between intron bases or sites between a p/t–base and an intron base. Thus, the probability of an exon being split or even destroyed by crossover is reduced to exactly zero. The generation of additional exons in intron regions by this operator is dependent on the intron base values. Therefore, introns could also be evolved to base sequences activating or suppressing the generation of exons, e.g., for a p–sequence $p_s =$ "00" the creation of exons could be activated by 0–rich introns, whereas 1–rich introns would suppress the formation of new exons (also depending on the t–sequence). As this crossover operator does not destroy building blocks, but has the ability to create additional exons, we call it *Conservative–Generative.*

The main benefit of this scheme is the preservation of building blocks (exons). Once a building block has emerged, it can only be destroyed by mutation. Spontaneous crossover could propagate good building blocks in the population and combine various exons on chromosomes to a better solution. The specific operations associated with spontaneous crossover can be seen in Figure 8.

Technically, for each genotype a *Crossover Mask* of equal length is stored, and the possible crossover sites are marked with 0, whereas sites in between exon bases or p/t–bases are marked with 1 indicating that the chromosome cannot be broken at these spots. Instead of externally defining a crossover rate p_c, the occurrence of crossover is based on homologous introns in both parents. The detection of crossover regions is achieved by the logical OR of the crossover masks of the two parents. The resulting *Crossover Template* is then scanned

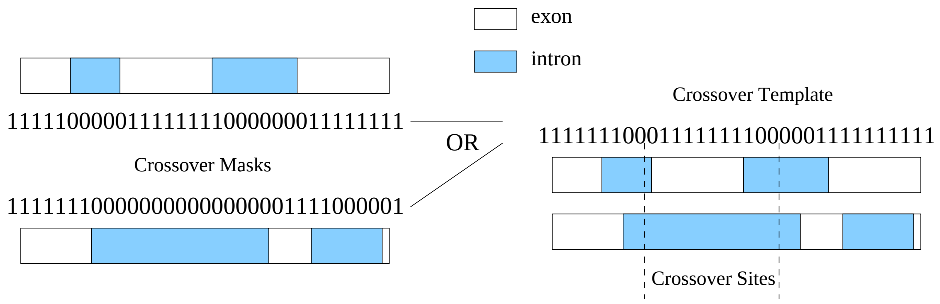

Figure 8: Mechanisms of spontaneous crossover.

for strings of 0s representing possible crossover regions. In the following we will just consider the case of using the midpoint of each 0–string as a crossover site, i.e. the crossover sites are defined as

$$c_{site,i} = \frac{b_{stop,i} - b_{start,i}}{2}, \tag{17}$$

where $b_{start,i}$ and $b_{stop,i}$ are the first and last base positions of the i–th 0–string on the chromosome. The crossover site $c_{site,i}$ is the "gap" upstream from the corresponding base position, i.e., $c_{site,1} = 2$ denotes the position between the first and second base on the chromosome.

As this crossover procedure is solely governed by the internal chromosome structure, and does not rely on a preset crossover rate, we have named it spontaneous crossover. The number of crossover sites is variable with spontaneous crossover and can be 0, i.e., no crossover, if the crossover template contains no single 0. It should be noted that this crossover type has (to our knowledge) no biological analog, and is mainly motivated by the properties discussed above, namely self–organization of position and number of crossover sites, and preservation of exons.

Figure 9 compares spontaneous crossover with the previously well–performing 2–point crossover (Figure 7).

For all space factors s the conservative–generative mechanism of spontaneous crossover outperforms 2–point crossover on the L–bird problem. In order to obtain a more detailed picture of the internal evolutionary processes associated with spontaneous crossover, we have a look at the number of crossover sites emerging from the ptGA gene structure using different space factors. Figure 10 depicts two typical runs searching and finding the optimal L–bird. The *Effective Crossover Rate* (observed crossover events / possible crossover events) was $p_c = 0.9924$ and $p_c = 1.0$ for the left and right graph, respectively.

In the left graph (Figure 10) we can clearly observe self–organization of the gene structure, as the number of crossover sites levels off at four at $g \approx 40$. At this time nearly all chromosomes have the same basic structure of five genes (exons). However, the small space factor $s = 1.5$ does not allow for multiple copies of a specific gene on a chromosome.

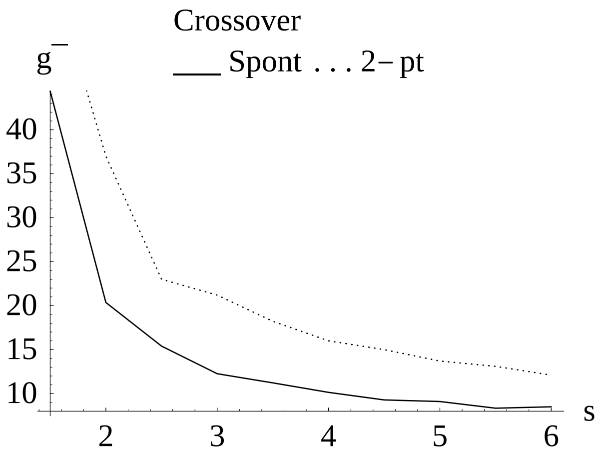

Figure 9: Average number of generations $\overline{g}$ to evolve to an L–bird comparing spontaneous with 2–point crossover, p_s = "00", and $t_l = 0$ (averaged on 50 runs, space factor s recorded in steps of 0.5).

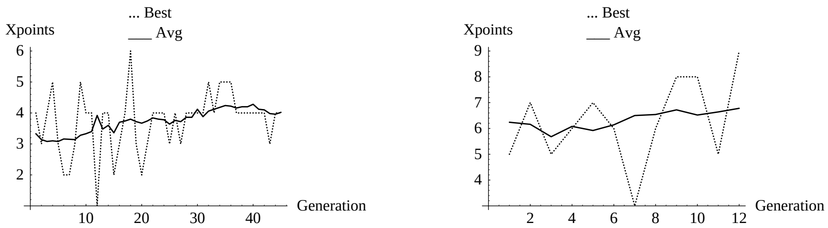

Figure 10: Average number of crossover sites and of best individual per generation used by spontaneous crossover for $s = 1.5$ (left) and $s = 3.0$ (right), p_s = "00", and $t_l = 0$ (averaged on 50 runs).

With $s = 3.0$ (right graph of Figure 10) the picture changes. Here, the early discovery of the optimum (generation 12) leaves too little time for emergence of a common gene structure which is indicated by the varying number of crossover sites (ranging from $3-9$) the best individual per generation has been generated with. However, the doubled space increases the total number of developed L–bird genes in the population, and allows for gene copies on a chromosome. Due to the relatively high number of crossover sites utilized by spontaneous crossover the genes are rapidly spread in the population without destroying them. Basically, the synergy of these effects, namely, large space factor, multiple genes, and non–disruptive crossover at numerous sites, enable the good performance of spontaneous crossover.

5.4 Chromosomes with fixed p/t–sequences

We now deliberately increase the search space for the L–bird problem by the use of t–sequences and by increasing p/t–sequence lengths. This is done so as to investigate ptGA performance towards more complex, realistic problems, where the size of parameters (genes) might vary and should be evolved to (sub)optimal lengths, too. It should be stressed that this is not necessary with the L–bird problem, as we know the size of genes (exons) in advance (3 tag bases + 6 parameter bases), but meanwhile we pretend to forget.

Though, it might seem to be a very simple task to assign tags to the various genes (Table 3), the interactions of tag–sequences, t–sequences and sequence lengths have to be observed carefully. For example, a tag of tag_s = "000" could be chosen for gene A, but this excludes all t–sequences contained in tag_s. With t_s = "00" an exon with an exon tag tag_s = "000" could never evolve using this specific t–sequence. On the other hand, it would be suitable to employ a t–sequence with mostly 0s, as the parameter bases in the L–bird exons should evolve to 1s by definition. The tag–sequences in Table 3 have been "optimized" for a p–sequence p_s = "111" and a t–sequence t_s = "000" resulting in a minimal chromosome length $l_{min} = 75$.

Evidently, for a more complex problem the p/t–sequences cannot be determined, as the optimal solution is unknown, i.e., with a t–sequence being identical to an optimal exon base sequence the optimum cannot be discovered. A possible solution to that problem is the evolution of p/t–sequences we proposed in (Mayer, 1997) where only the p/t–sequence length has to be set in advance, and the p/t–sequence bases are evolved by ptGA.

Figure 11 shows the average number of generations to find the optimum employing both spontaneous and 2–point crossover.

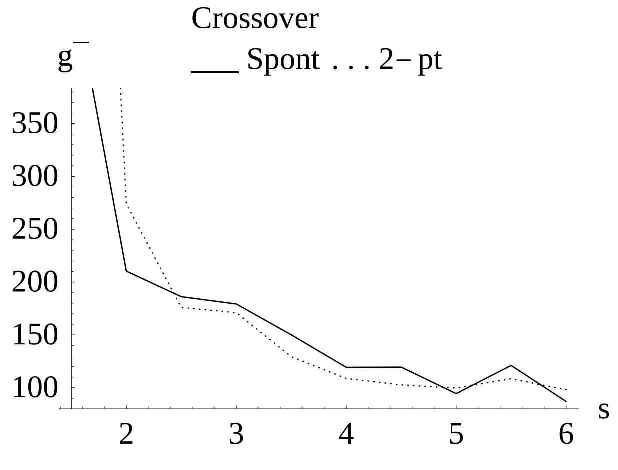

Figure 11: Average number of generations $\overline{g}$ to evolve to an L–bird comparing spontaneous with 2–point crossover, p_s = "111", and t_s = "000" (averaged on 50 runs, space factor s recorded in steps of 0.5).

Again, the influence of increasing space factor is the most notable characteristic exhibited by the ptGA runs with $p_l = t_l = 3$ ($\overline{g}$ decreasing from $\approx 500 - 100$). However, spontaneous crossover only improves ptGA performance with $s < 2.5$. One of the reasons might be the simple fitness function used. As only exons containing 9 bases have a chance to evolve to the correct L–bird genes, all other exons not only do not need to be preserved, but should be split to make room for creation of exons of correct length. Intrinsically, spontaneous crossover does not break exons which in this case may reduce the benefits suggested above.

With a slightly changed fitness function, where exons of correct length (9 bases) receive a small credit regardless of their value, the difference between spontaneous and 2–point crossover was much more pronounced (in favor of spontaneous crossover) (Mayer, 1997). Still, with the simple fitness function and a space factor $s = 6.0$ spontaneous crossover took 87 generations on average to find the optimum (Figure 11). With the corresponding $i' = 0.83$ (Equation 16) for a chromosome with introns at fixed locations it took 25 generations, because only 30 bases had to evolve to the correct value, but 75 when using p/t–sequences.

6 The Zero/One Multiple Knapsack Problem

After having analyzed the ptGA in the context of the simple L–bird problem, we next challenge the ptGA with an NP–complete problem. The 0/1 multiple knapsack problem is a combinatorial optimization problem which is also of practical relevance, as it can be used to model industrial opportunities like cutting stock, cargo loading, and the capital budget. We experiment with three problem instances of increasing complexity and compare ptGA performance to results employing conventional GAs (Khuri et al., 1994). The various instances of 0/1 multiple knapsack problems have been obtained from the OR-library [5] (Beasley, 1990).

The 0/1 multiple knapsack problem is a variant of increased complexity of the 0/1 simple knapsack problem. For the latter, there are only two algorithms besides GAs delivering optimum solutions, i.e. *Dynamic Programming* and the *Branch–and–bound* approaches. Only one of them (branch–and–bound) can be used to solve multiple knapsack problems (Khuri et al., 1994).

The less complex, but still NP–complete, 0/1 simple knapsack problem can be visualized by imagining a knapsack of capacity C and n candidate objects to be put into the knapsack. Each object has a weight w_i and a profit p_i. The goal is to fill the knapsack with objects that yield maximum profit under the constraint that the overall weight of the objects must not exceed the capacity of the knapsack. To put it more formally, we are searching a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ with $x_i \in \{0, 1\}$ that satisfies the constraint $\sum_{i=1}^{n} w_i x_i < C$, and maximizes the profit $P(\vec{x}) = \sum_{i=1}^{n} p_i x_i$. This definition leads to a straight forward encoding of

[5] http://www.ms.ic.ac.uk/info.html

the problem suited for a conventional GA. The vector of objects is simply represented as a binary string, where the 1s and their position on the chromosome correspond to objects in the knapsack, and the 0s denote all objects not to be packed into the knapsack.

The generalization of the 0/1 simple knapsack problem is the 0/1 multiple knapsack problem. We, again, have n objects, but this time we imagine m knapsacks. If we decide to put one object in a knapsack, it is virtually put in all m knapsacks each of them having a specific capacity $c_1, c_2, \ldots, c_m$. An object's weight is dependent on the knapsack it is put in, thus, an object i in knapsack j has the weight w_{ij}. This models the consumption of resource j by object i within a certain budget for each resource (knapsack capacities). The profit should be maximized obeying the constraints of limited resources.

Khuri et al. (1994) used a fitness function with penalty terms (Richardson et al., 1989) enabling survival of infeasible solutions in the population, especially, when they overfill only few of the m knapsacks. The fitness function (Khuri et al., 1994) is given by

$$f(\vec{x}) = \sum_{i=1}^{n} p_i x_i - o p_{i,max}, \tag{18}$$

where $p_{i,max}$ is the most expensive of the n objects, and o the number of overfilled knapsacks.

6.1 Interpretation of exons

An evolved exon on the ptGA chromosome represents the binary coded index of an object i. If an exon generates an index $i > n$ (the number of objects), this parameter is discarded (invalid object). If two or more exons (multiple exons) yield the same index, this index and the corresponding object is only put once into the m knapsacks (according to the problem statement). All the valid objects encoded in one ptGA chromosome are then used to evaluate the fitness function.

As the maximum number of objects is known, we only employ a p–sequence with an exon of constant length. By setting the exon length to the minimal value sufficient to encode all object indices $e_l = \lceil \log_2 n \rceil$ (n is the number of objects), we incorporate problem knowledge on the encoding level, as the search space is restricted to object indices $1 - 2^{e_l}$. Still, invalid objects can be generated (unless the number of objects is a power of 2), and the number of exons (objects) varies.

The problem in discussion is a good test for the ptGA, because the number of objects for the optimal solution is unknown, and the ptGA has to evolve the correct number of exons. Additionally, it has to discover valid objects that can be put into the knapsacks, i.e., the range of object indices. We will use the results of the conventional GA runs to determine the set of optimal object indices. From this we can calculate the minimal length l_{min} (Equation 13), and use a space factor $s = 3.0$ for the experiments. Thus, problem knowledge is also incorporated via chromosome length, but the number of exons can still vary from 0 to the triple number of optimal object indices. It is essential to note that the search space the conventional GA faces is smaller than that generated by the ptGA encoding, for the standard GA only operates with valid objects and does not have to bother with discovery of valid object indices and their number.

6.2 Experiments with three problem instances

In the following we compare ptGA runs with the above encoding to the GA runs recorded in (Khuri et al., 1994). We will label their results as KBH (for Khuri, Bäck, Heitkötter).

Khuri et al. (1994) used a conventional generational GA with population size $n = 50$, a mutation rate $p_m = \frac{1}{l}$, crossover rate $p_c = 0.6$, proportional selection, and 1–point crossover. For comparisons ptGA is also run in conventional mode labeled as *(pt)GA* using 2–point crossover and binary tournament selection. The ptGA parameters are set to the default values used with most experiments in this article ($p_s =$ "00", 2–point ($p_c = 0.6$) and spontaneous crossover).

The number of objects n is given by the specific problem instance label, the number of knapsacks is always $m = 10$, and the number of fitness function evaluations is identical to (Khuri et al., 1994). The evolutionary search is performed 100 times. The results are presented in Table 4.

Problem		KBH	(pt)GA	ptGA(2–pt)	ptGA(SpontX)
Knap15	$\#opt$	83	93	91	95
$f_{opt} = 4015$	$\overline{f}$	4012.7	4014.3	4014.1	4014.5
5×10^3 *eval.*	σ	–	2.56	2.88	2.19
Knap20	$\#opt$	33	21	81	97
$f_{opt} = 6120$	$\overline{f}$	6102.3	6099.5	6115.8	6118.7
10^4 *eval.*	σ	–	20.07	10.75	8.49
Knap28	$\#opt$	33	36	39	7
$f_{opt} = 12400$	$\overline{f}$	12374.7	12382.9	12368.8	12345.1
5×10^4 *eval.*	σ	–	19.03	54.54	37.16

Table 4: *0/1 Multiple Knapsack Problems* – Statistical comparison ($\#opt$…found optima, $\overline{f}$…mean fitness, σ…standard deviation) of results of conventional GA (KBH and (pt)GA) and ptGA experiments using 2–point and spontaneous crossover, space factor $s = 3.0$, 100 runs.

Generally, the results clearly demonstrate the capabilities of ptGA to solve difficult problems, however the sometimes big performance differences might be an indication for complex interactions of ptGA encoding and parameters. An example for this observation are the results of the Knap20 problem, where an exon length $e_l = 5$ results in the possible expression of 12 invalid objects (exons). Although only 4 invalid objects can be generated with Knap28 using the same exon length, the Knap20 results are significantly superior to conventional GA results, while the Knap28 results are not. After increasing the exon length to $e_l = 6$ (36 invalid objects) with Knap28, the ptGA with spontaneous crossover found the optimum 14 times with a mean fitness $\overline{f} = 12385.6$ and a standard deviation $\sigma = 10.67$. Thus, despite enlarging the search space, in this specific case ptGA performance improved.

A possible explanation for this behavior is that with increasing exon length specific bases can be intrinsically used as a switch, i.e., an activator or repressor of an exon. E.g., mutation of the most significant base in the above example ($e_l = 6$) mostly alters a valid to an invalid object or vice versa. This creates a form of memory storing previously successful objects in the chromosome while evaluating other solutions. Further evidence along these lines is given by Mayer (1997) in the analysis of ptGA populations in which the optimum has been found. Often a great number of specific exons coding for an invalid object index can be observed. A single mutation at various loci of these exons – termed *Joker Exons* in (Mayer, 1997) – changes the non–valid object to a valid object [6].

7 Summary

In this article we have reported on experiments with non–coding segments on artificial chromosomes for GAs. It has been formally shown that the insertion of "junk" code at fixed locations decreases the disruptive effects of conventional k–point crossover proportionally to the content of non–coding GA bases. Experiments on the simple, epistatic L–bird problem revealed that the insertion of non–coding segments at fixed locations improves GA performance measured by the average number of generations to find an optimal solution. This is departing from previous work on non–epistatic problems (Wu, 1996; Wu and Lindsay, 1996), where non–coding regions only slightly (if at all) reduced the average number of generations. However, with evolution of the location of non–coding sequences Wu (1996) reported a remarkable decrease of generations used to find an optimum when compared with conventional GAs.

With the use of p/t–sequences defining beginning and end of a gene, respectively, the location, number, and length of coding and non–coding segments is subjected to evolution. Although, the gene structure becomes more complex, ptGA performance is comparable to conventional encoding with problem parameters at fixed locations, and increases with ptGA chromosome length. Larger space on the chromosome allows for a higher number of genes (building blocks) and multiple copies of identical genes.

In order to efficiently distribute building blocks, we devised a new crossover operator called spontaneous crossover. This is a conservative–generative operator preserving existing building blocks, while possibly creating new ones. Crossover sites emerge due to size and location of introns on the chromosome.

However, the generally similar behavior (finding optima more rapidly with increasing ptGA chromosome length) of the (also) destructive k–point crossover

[6] This also requires a specific encoding where an exon is divided into subunits.

and the non–destructive spontaneous crossover suggest that non–coding segments' main benefit is the locus–independent evolution of building blocks on separate (or even identical) ptGA chromosomes. Once a building block has evolved, it can be quickly distributed in the population, and the occurrence of identical building blocks at different locations (depending on chromosome space) ensure more likely survival of parts of the solution.

Another advantage of the proposed chromosome structure might be the self–organization of gene locations, possibly clustering cooperating genes and reducing epistasis. However, this remains to be studied more closely, and it should be noted that cooperating, natural genes in eukaryotes often are not even located on the same chromosome (Beebee and Burke, 1992).

The choice of p/t–sequences enabling evolution to an optimal solution will be extremely difficult for realistic problems. Specifically, the t–sequence heavily interacts with the exon bases representing the value of a problem parameter. Selecting a t–sequence which is part of an exon base sequence required for the (sub)optimal solution of a problem makes it impossible to find that optimum. Either, we do not utilize a t–sequence at all by defining a p–sequence (causing less difficulties) and a constant length of exons, or we change the representation of exon values.

In (Mayer, 1997) we began to experiment with another idea borrowed from genetics, specifically, the translation of genetic code to proteins. The exon is split into *Tuplets* of equal length (triplets in nature), each tuplet encoding a part of the exon value (amino acid in nature). Combination of tuplets allows for a very flexible and redundant encoding and significantly reduces the interactions of exon bases and t—sequence. A similar approach has been proposed in (Yoshikawa et al., 1997).

Thinking of future applications, we expect ptGA to be a good choice when the number of parameters of a (sub)optimal solution is not known in advance. Problems in numerics (e.g., optimal number and distance of grid lines), statistics (e.g., optimal number and location of clusters), image processing (e.g., optimal number and location of segments), computer graphics (e.g., triangulation), or computer networks (e.g., flow control) are examples for the potential use of ptGA. Recently, the use of *Gene Locators* (p–sequences in our notion) with fixed length bit string (exon bases) encoding the parameter value have been used with IRR (*Implicit Redundant Representation*) GAs (Raich and Ghaboussi, 1997). IRR GA outperformed conventional GAs remarkably compared on a mechanical structure optimization problem. Some results of experiments on the 0/1 multiple knapsack problem (Section 6) confirm this observation.

Investigations of the applicability of these ideas, comparisons of ptGA performance with other algorithms on the class of problems indicated above, further variants of spontaneous crossover, and theoretical work on tuplet encoding will be amongst our future research activities.

A Acknowledgments

I would like to thank *Prof. Horst D. Clausen* and *Prof. Jochen Pfalzgraf* for supporting this work and that of my "evolutionary" colleagues in a field new to the department. Special thanks to *Hannes Flöckner* (Center for Applied Molecular Engineering, University of Salzburg) for long hours of assistance in molecular genetics, and *Christian Blaschke* (Department of Plant Physiology) for many discussions on evolutionary genetics. I very much appreciate the very constructive and detailed reviews of two anonymous referees which helped to improve the article in many aspects.

References

Adams, R. L. P. (1991). *DNA Replication*. In Focus. Oxford University Press, New York. ISBN 0-19-963216-2.

Bäck, T. (1992). The interaction of mutation rate, selection and self–adaptation within a genetic algorithm. In Männer, R. and Manderick, B., editors, *Parallel Problem Solving from Nature 2*, pages 85–94, Amsterdam. Elsevier.

Beasley, J. E. (1990). OR–Library: Distributing Test Problems by Electronic Mail. *Journal of Operational Research Society*, 41(11):1069–1072.

Beebee, T. and Burke, J. (1992). *Gene structure and transcription*. In Focus. Oxford University Press, New York, 2nd edition. ISBN 0-19-963317-7.

Bridges, C. L. and Goldberg, D. E. (1987). An analysis of reproduction and crossover in a binary-coded genetic algorithm. In Grefenstette, J. J., editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 9–13. Naval Research Laboratory, Lawrence Erlbaum Associates.

Bui, T. N. and Moon, B. R. (1994). Analyzing Hyperplane Synthesis in Genetic Algorithms Using Clustered Schemata. Technical Report CSE-94-026, Penn State University, University Park, PA.

Cavalier-Smith, T. (1985). *The Evolution of Genmome Size*. Wiley, New York.

Cavalier-Smith, T. (1991). Intron phylogeny: a new hypothesis. *Trends in Genetics*, 7(5):145–148.

deDuve, C. (1994). *Ursprung des Lebens: Präbiotische Evolution und die Entstehung der Zelle*. Spektrum Akademischer Verlag, Heidelberg. ISBN 3-86025-187-2.

Doolittle, W. F. (1978). Genes in pieces: where they ever together? *Nature*, 272:581.

Fabry, S. (1995). Frühe Introns – späte Introns? *Biologie in unserer Zeit*, 25(5):300–306.

Gilbert, W. (1978). Why genes in pieces? *Nature*, 271:501.

Gilbert, W. (1987). The exon theory of genes. *Cold Spring Harbor Sympiosa on Quantitative Biology*, 52:901–905.

Gilbert, W. and Glynias, M. (1993). On the ancient nature of introns. *Gene*, 135:137–144.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison–Wesley, Reading, Massachusetts. ISBN 0-201-15767-5.

Goldberg, D. E., Deb, K., Kargupta, H., and Harik, G. (1993). Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms. In Stephanie Forrest, U. o. N. M., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64. University of Illinois at Urbana-Champaign, Morgan Kaufmann.

Hoffmann, F. and Pfister, G. (1995). A New Learning Method for the Design of Hierarchical Fuzzy Controllers Using Messy Genetic Algorithms. In *Proceedings Sixth International Fuzzy Systems Association World Congress, Vol. 1*, pages 249–252.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. MIT Press.

Khuri, S., Bäck, T., and Heitkötter, J. (1994). The Zero/One Multiple Knapsack Problem and Genetic Algorithms. In *Proceedings of the 1994 ACM Symposium on Applied Computing*, pages 188–193. ACM Press.

Knuth, D. E. (1969). *The art of computer programming*, volume 2. Addison–Wesley, Reading, Massachusetts.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by means of Natural Selection*. Complex Adaptive Systems. The MIT Press, Cambridge, MA.

Levenick, J. R. (1991). Inserting Introns Improves Genetic Algorithm Sucess Rate: Taking a Cue from Biology. In Belew, R. K. and Booker, L. B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 123–127. University of California, San Diego, Morgan Kaufmann.

Lodish, H., Baltimore, D., Berk, A., Zipursky, S. L., Matsudaira, P., and Darnell, J. (1995). *Molecular Cell Biology*. Scientific American Books, New York, 3rd edition. ISBN 07167-2380-8.

Mayer, H. A. (1997). *ptGAs – Genetic Algorithms Using Promoter/Terminator Sequences - Evolution of Number, Size, and Location of Parameters and Parts of the Representation*. PhD thesis, University of Salzburg.

Mitchell, M., Forrest, S., and Holland, J. H. (1991). The royal road for genetic algorithms: Fitness landscapes and ga performance. In Varela, F. J. and Bourgine, P., editors, *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, Cambridge, MA. MIT Press.

Mühlenbein, H. (1992). How genetic algorithms really work: I. mutation and hill-climbing. In Männer, R. and Manderick, B., editors, *Parallel Problem Solving from Nature 2*, pages 15–25, Amsterdam. Elsevier.

Nordin, P., Francone, F., and Banzhaf, W. (1995). Explicitely Defined Introns and Destructive Crossover in Genetic Programming. In Angeline, P. and Kinnear, K., editors, *Advances in Genetic Programming II*, Cambridge, MA. MIT Press.

Raich, A. M. and Ghaboussi, J. (1997). Autogenesis and Redundancy in GA Representation. Extended Abstract, Workshop on Exploring Non-coding Segments and Genetics-based Encodings, Seventh International Conference on Genetic Algorithms,1997, Department of Civil Engineering, University of Illinois at Urbana-Champaign.

Richardson, J. T., Palmer, M. R., Liepins, G., and Hilliard, M. (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197, San Mateo, California. Philips Laboratories, Morgan Kaufman Publishers, Inc.

Rogers, J. H. (1989). How were introns inserted into nuclear genes? *Trends in Genetics*, 5:213–216.

Shaefer, C. G. (1987). The ARGOT Strategy: Adaptive Representation Genetic Optimizer Strategy. In Grefenstette, J. J., editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 50–55. Naval Research Laboratory, Lawrence Erlbaum Associates.

Sharp, P. A. (1987). Splicing of Messenger RNA Precursors. *Science*, 235:766–771.

Smith, J. M. (1989). *Evolutionary Genetics*. Oxford University Press, New York. ISBN 0–19–854215–1.

Smith, R. E., Goldberg, D. E., and Earickson, J. A. (1991). SGA-C: A C–Language Implementation of a Simple Genetic Algorithm. TCGA Report 91002, The Clearinghouse for Genetic Algorithms, The University of Alabama, Department of Engineering Mechanics, Tuscaloosa, AL 35487.

Wu, A. S. (1996). *Non–Coding DNA and Floating Building Blocks for the Genetic Algorithm*. PhD thesis, University of Michigan.

Wu, A. S. and Lindsay, R. K. (1996). A comparison of the fixed and floating building block representation in the genetic algorithm. *Evolutionary Computation*, 4(2):169–193.

Yoshikawa, T., Furuhashi, T., and Uchikawa, Y. (1997). The Effects of Combination of DNA Coding Method with Pseudo–Bacterial GA. In *Proceedings of the 4th IEEE International Conference on Evolutionary Computation*, pages 285–290. IEEE Press.