Extending Grammatical Evolution with Attribute Grammars: An Application to Knapsack Problems

by

Robert Cleary

B.Sc.

Supervisor: Dr. Michael O'Neill

External Examiner: Dr. Robert McKay

A thesis for the Masters Degree by Research

Submitted to the University of Limerick

June 2005

Declaration

I hereby declare that the work presented in this thesis is original except where an acknowledgment is made or a reference is given to other work and I have read the University hand book of academic administration and I accept its procedure.

Student: Robert Cleary

Signed:

Date: July 7, 2005

Supervisor: Dr. Michael O'Neill

Signed:

Date: July 7, 2005

Abstract

Research extending the capabilities of the well-known evolutionary-algorithm (EA) of Grammatical Evolution (GE) is presented. GE essentially describes a software component for (potentially) any search algorithm (more prominently an EA) - whereby it serves to facilitate the generation of viable solutions to the problem at hand. In this way, GE can be thought of as a generally-applicable, robust and pluggable component to any search-algorithm. Facilitating this plug-ability - is the ability to hand-describe the structure of solutions to a particular problem; this, under the guise of the concise and effective notation of a grammar definition. This *grammar* may be thought of, as the rules for the generation of solutions to a problem.

Recent research has shown, that for static-problems - (problems who's optimum-solution resides within a finitely-describable set, for the set of all-possible solutions), the ability to focus the search (for the optimum) on the more promising regions of this set, has provided the best-performing approaches to-date. As such, it is suggested that search be *biased* toward more promising areas of the set of all possible solutions.

In it's use of a grammar, GE provides such a bias (as a *language-bias*), yet remains unable, to effectively bias the search for problems of constrained-optimisation. As such, and as detailed in this thesis - the mechanism of an *attribute grammar* is proposed to maintain GE as a pluggable component for problems of this type also; thus extending it's robustness and general-applicability.

A family of academically recognised (hard) knapsack problems, are utilised as a testing-ground for the extended-system and the results presented are encouraging. As a side-effect of this study (and possibly more importantly) we observe some interesting behavioral findings about the GE system itself. The standard GE one-point crossover operator, emerges as exhibiting a mid-evolutionary change-of-role from a constructive to destructive operator; GE's ripple-crossover is found to be heavily dependent on the presence of a GE-tail (of residual-introns) in order to function effectively; and the propogation of individuals - characterised by large-proportions of such residual-introns - is found to be an evolutionary self-adaptive response to the destructive change-of-role found in the one-point crossover: all of these findings are found with respect to the problems examined.

Acknowledgments

If learning nothing else from working by research, it is easy to see that it's all about attributing credit to the work of others, which has inspired your own: *credit where credit-is-due* (so to speak). As such and without delay, it wouldn't be right to continue, without paying homage to some music which helped-me-out when times were not so good: that is, the soulful - music of Bob Marley; voice of Johnny Cash; and lyrics of Jewel (to name but a few), "*get-up, stand-up, don't give-up the fight*".

Now then, to a more conventional focus! To my supervisor, Dr. Mike O'Neill: *a gentleman and a scholar*; who never failed to send me *out* of his office, in a more positive mood than when I went in; thanks for persevering with me Mike. Miguel Nicolau (who suggested I try an experiment where crossover is restricted); without which, (I can only presume) would have me currently scratching my head over the empty-templates entitled, Chapter 5, 6, 7, and 8. Dr. Atif Azad was ever-helpful, on several occasions - doubling-up as a LaTeX companion; thanks Atif. Hammad Majeed (soon to be Dr., I have no doubt), who helped me to see why conditions are allowed in an attribute-grammar. Dr. Tony Brabazon, for his opinion on where to weight the emphasis of the thesis. The others from the general area of CSG028, you should know who you are - thanks.

For the times when my arms felt very-short - and my pockets deep (to the inverse-proportion): Mrs. Margaret O'Carroll (Carlesberg don't do land-ladies . . .); Mr. and Mrs. Gilbert - who had blind-faith in my ability to build their stone-wall (I'll be back soon now, to finish it, I promise); similarly, Janice O'Connell who had blind-faith in my ability to teach Java. Thanks here also, to the brothers (and sister), uncle and parents. All of the above, I must thank especially - for providing *life*-funds when time ran as scarce as money. The CSIS department, for funding my trip to EuroGP - also!

To the people who saw my moments of most-oddness (at closest-range) and put-up with me nonetheless; Brendan, John, Colm and Mary - particular acknowledgment to the latter two, for an un-yielding affinity to cups-of-tea (and talk of fishes, cattle and the like) in the wee hours of the morning of 1A's kitchen - thanks lads and *bird*!

To my brother Steve: who's stubborn display of abuse to the human-body (in physical exercise), both amazes me and shows a strength which inspires me. To his combined efforts with the big bro' Jim to get me into running, without which, I would never have completed the marathon; thanks lads. Dave and Kev . . . for Coosan-runs, wine - food, porter; UML and Java. Anne, just for being *the-sister*. The rest of yee know who yee are (ya!, nephews, nieces, in-laws and Spanish-contingent alike).

The friends!: Gill, Ray, Harr's, Finty, Sutts, Sly and Rayo (standing to mind)- thanks lads.

Finally, to my parents - for your continuing patience as I succeed again-and-again (mystifyingly - no doubt), to grow-older - without growing-*up*: as a good-friend of mine recently said - "it's not the destination, but the journey there" that counts. Bear with us!.

Contents

List of Figures

List of Tables

"*It is our doubts that unite us; it is only our certainties that keep us apart*"
Sir Peter Ustinov

Chapter 1

Grammars For Constrained-Optimisation

1.1 Introduction

A grammar (it will be seen), can be used as a very powerful expressive force; defining a language, over which an evolutionary-algorithm (or EA), can explore. Combined, they present what is referred to as a *grammar-based EA*, and in this way Grammatical Evolution (GE) provides a grammar-based exploration of all the possible solutions to a problem; the goal being, to result in an optimal (or near optimal) solution being proposed as a solution to that problem. GE harnesses what is referred to as a context-free grammar (CFG) which allows the definition of a broad range of languages; tailorable to almost any problem-domain. As such, the task of the EA can be reduced to a directed-search through the finite space of legal *sentences* of the language defined. Directing this search - is an (artificial) abstraction of the underlying processes which are believed to be the driving-force of biological-evolution: the evolutionary operators of the EA.

The work of this thesis then, documents the application of GE to *knapsack* problems: a category of NP-hard constrained-optimisation problems[1]. These knapsack-problems can be used as a generalization for many real-world industrial problems of great difficulty - whereby, an improvement in the best known optimum (or current way of solving the problem) can afford huge financial gains and time savings, for companies working in these industries. Knapsack problems and their solutions allow theoretical application to the practical fields of airline scheduling and container-packing, as well as the lumber and shipping industry (to name but a few); such theoretical works providing the basis for a real-world practical application. In this thesis then, we propose the use of knapsack problems as a sample test-bed, for research into the more general application of GE to problems of constrained-optimisation. We are thus interested, not only in problem-specific results - but the general applicability of GE to problems of constrained optimisation; in this role, a secondary (and possibly more significant) interest lies in observing the behavior of GE under this type of problem.

Solutions to problems of constrained-optimisation are said to be describable by context-sensitive languages - and therein an *attribute grammar* is proposed as a method to describe such languages. We propose that attribute grammars provide a gateway for the application of GE to problems of constrained-optimisation (it will be seen, that standard GE with a CFG, was insufficient in this role). The attribute grammar, provides a powerful method to extend the capabilities of standard GE; allowing it to describe solutions to problems which are context-sensitive in nature.

[1]NP-hard describes the complexity of a decision problem, implying that it is intrinsically harder than those that can be solved by a non-deterministic Turing machine in polynomial time.

It is observed, that GE - when adapted with an attribute grammar - is applicable to knapsack problems. The results of experiments are promising, and we observe some interesting behavioral findings about the general algorithm itself. In observing it's behavior we highlight areas of interest to research; asking questions of the approach to learning provided by the EA, and hope to provide an impetus for curiosity in the line of future work.

There follows a very-brief introduction into the concept behind constrained-optimisation - with a short review of grammars and grammar-based EAs; a particular focus is maintained, on those contributing to the inception of GE. GE itself is described, the contributions of this thesis noted, and the thesis-outline documents the work of each individual chapter forthcoming.

1.2 Constrained-Optimisation: A Gentle Introduction

Chinneck states that "Practical optimisation is the art and science of allocating scarce resources to the best possible effect" [Chinneck, 2000]. In essence (from the perspective of the problems researched in this thesis)with a knapsack-problem - or any of the more common constrained-optimisation problems - we are dealing with a *static* problem. Some real-world problem exists where a set-of-items have a desirable property, and we wish to find out - the arrangement of these items - that would produce the most beneficial outcome to their user. A mathematical-model can be used to reduce the solution (of this problem) to an algebraic-equation where an optimal set-of-values for each variable (item) will provide us with the best possible arrangement. That is, every possible arrangement of variable co-efficients, maps to one particular point on a graph of the function for the algebraic-equation. Figure 1.1 attempts to better explain this. An example optimisation-problem is plotted as a graph; where points on the line of each plot relate to a particular set-of-values for the variables of the problem; therein mapping to an *objective-function* value (i.e. $f(x)$).

In Figure 1.1, the graph on the right shows how - in constrained-optimisation, some other functions (as opposed to the objective-function) define constraints on the problem; as *cutting-planes* over the now defined search-space[2]. The figure describes constraints which are linear in nature, and the surface below their intersection with the absolute search-space, defines the feasible (legal) region of search. The problem can be reduced to a search (over this static-space) - defining arrangements-of-values for the co-efficients of the objective function; the search becomes a search for the optimal peak of the graph. As each arrangement-of-items provides a *fitness* measure (or objective-function value) for that given solution, the set of all arrangements maps-out what is described as a *fitness-landscape* (See Section 1.3).

1.3 Evolutionary Algorithms (EAs)

The field of evolutionary computation is based on Charles Darwin's notion of biological evolution; suggesting, that it is natural-selection which drives the evolution of a species [Darwin, 1859]. Also heavily incorporated into artificial-evolution, is the principle of "survival of the fittest" first introduced by Herbert Spencer [Spencer, 1864]. Given a population of individuals where each individual represents a possible solution to a particular problem (e.g. an optimal arrangement of items), each individual is evaluated; and a reflection

[2]We refer the reader to Chinneck's text for a more in-depth discussion of practical-optimisation [Chinneck, 2000]. As such, here - it is just important to understand that constraints define cutting-planes over the absolute search-space.

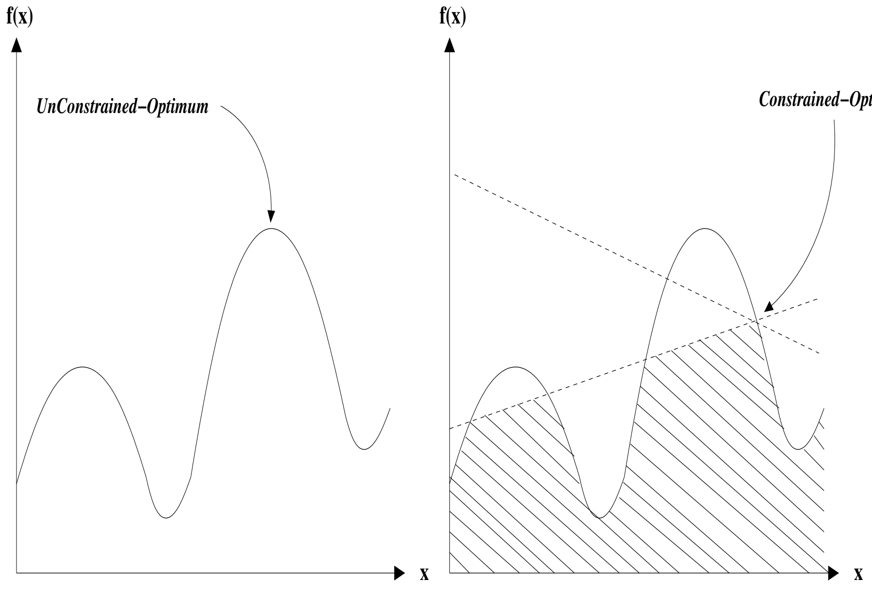

Figure 1.1: Highlighting the difference between constrained and unconstrained optimisation. Constraints in the former define *cutting-planes* over the unconstrained problems search-space. The shaded-area defines the *feasible* region of search.

on it's ability to tackle the problem is determined. This performance measure, is referred to as *fitness*. As such, the evaluated fitness of each (possible) solution - when graphed, maps out what the previous section referred to as the fitness-landscape, or search-space explorable by the EA. The operation of an EA, can thus be thought of as a search through a fitness-landscape.

Individuals, or chromosomes (the two terms are used interchangeably) that exhibit superior performance, are assigned a better fitness value. These individuals are seen to better adapt to their environment (*solving the problem*) and thus, are provided with a better chance to reproduce with other individuals; it follows, that these individuals are more likely to pass their genetic-material to the next generation. In artificial-evolution, the process of selecting individuals to reproduce, is an abstraction of natural-selection. The idea being, that over time, reproduction and natural-selection allow evolution from a gene pool of progressively fitter individuals (individuals which exhibit a strong ability to adapt to their environment).

Prior to an artificial evolutionary run, a population must be initialised; typically by a random process, but in some cases it is desirable to seed the population with fit solutions from a previous run; or attempt to impose a diverse arrangement of structure on the first generation (See Chapter 7). This is commonly the case in Genetic Programming (GP), where variable-length tree structures represent individuals.

```
Initialise Population
WHILE termination criteria not satisfied
DO
        EVALUATE fitness of each individual
        SELECT parents
        Apply GENETIC_OPERATIONS
        CREATE_NEW_POPULATION
END-DO

REPORT best-of-run Individual
```

As depicted in the above outline - each cycle of an evolutionary algorithm, otherwise referred to as *a generation*, is typically comprised of the phases: selection; genetic manipulation; testing (or evaluation), and replacement. During selection, individuals (based on their fitness values), are selected to take part in an artificial abstraction of sexual reproduction; this abstraction of natural sexual reproduction typically effects some genetic manipulation to these *parents*, and generates corresponding *offspring.* Offspring are subjected to fitness evaluation in the testing phase; this will determine the fitness of each child solution. The replacement strategy determines how the children, parents and other members of the current population are used to create the next generation. It can be seen that new generations are continuously created until some termination criteria have been satisfied: generally, either an adequate solution has been found; or a predefined maximum number of generations have occurred.

Traditionally there were four main approaches to an artificial evolutionary algorithm; genetic-algorithms (GAs), genetic-programming (GP), evolutionary-programming (EP) and evolution-strategies (ES). Although traditionally deciphered by the representation used - GAs using fixed-length binary strings; ES using real-valued vectors; EP using finite state machines; and with GP, Lisp S-expressions; it has been pointed-out that the boundaries of distinction between each approach are becoming increasingly blurred [De Jong, 1999] [O'Neill and Ryan, 2003a].

As the main focus of this thesis is grammar-based GP, we concentrate our introduction on GP (Genetic Programming) and its variants. The essence of GP is to evolve more complex structures than GAs: the role of parameter-optimisation predominating as the main use for traditional GAs. GP aimed at representing computer programs (or critical parts of computer programs) and as depicted in Figure 1.2, the standard GP is comprised of a population of Lisp S-expressions represented in the form of syntax trees. Inner nodes comprise a function-set, with subtrees evaluating to operands for these functions. The syntax tree representation facilitates the genetic manipulation of the evolving programs by using a crossover operator which swaps the subtrees of parents. Genetic *mutations* take the form of the random re-generation of a subtree. In order to evolve a computer program by means of GP, primitives or terminal-symbols of the system are defined.

Owing to a possible confusion with other computer science terminologies, Whigham [Whigham, 1996] has proposed the terms **GP-functions** and **GP-terminals** to clarify the two main components of a GP approach. For a

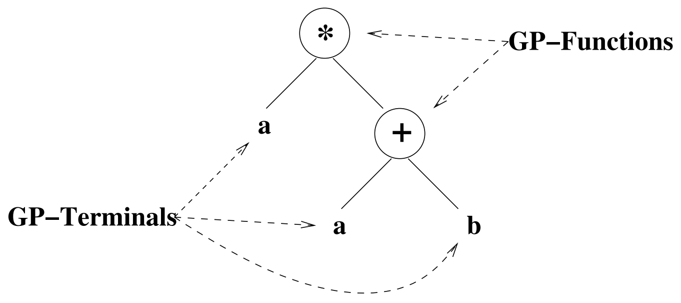

Figure 1.2: Simple GP syntax tree showing GP-functions and GP-terminals.

thorough description of GP one should refer to Koza's three books, Genetic Programming 1, 2 and 3 [Koza, 1992, Koza, 1994, Koza et al., 1999], the first GP text book by Banzhaf et al. [Banzhaf et al., 1998], and the Advances in GP series [Kinnear, 1994, Angeline and Kinnear, 1996, Spector et al., 1999].

1.3.1 Separating Search and Solution Space

In GP, reproduction is abstracted through the crossover operator, where subtrees from two parents are exchanged. The points of exchange are typically chosen randomly. To enable an unrestricted exchange, all the GP-functions and GP-terminals should produce - as output - an acceptable input to every GP-function, in the function-set. As GP uses syntax-trees to represent it's chromosomes, inner-nodes represent functions; these functions evaluate to some expression (based on their *child-node* parameters). It must be ensured that the data-type of this resulting expression, is such that it can be received as a parameter to all other functions. That is, expressed functions must be *closed* under all operations defined. This has been termed the principle of *closure* [Koza, 1992]. This imposes a restriction on the way in which genetic-manipulation can take-place, questionably also restricting the space of possible phenotypic structures explorable.

Montana [Montana, 1994] addressed this problem by the use of a restricted crossover operator. When a subtree is selected for exchange, it can only be swapped into the other parent at a node of the same data-type. Subtrees producing an integer, for example, can only be swapped with those that produce an output with the same data-type. It will be seen, that grammars (and the use of grammar-based EAs) can be used to provide a method of subtree-crossover (See Section 1.4); which both overcomes the problem of closure and allows for an arbitrary representation (as opposed to the traditional GP tree-structure)

Contradicting the traditional approach, Banzhaf [Banzhaf, 1994] argued against - what he proposed - was a deviation from a natural model of evolution. He argued that, nature normally distinguishes between the genetic representation (the genotype), and the observed traits (the phenotype) of an individual. A complex mapping process exists, producing an organism from a set of genetic instructions encoded on DNA. It was pointed out, that by ignoring this distinction - common GP approaches did not follow a natural metaphor. Banzhaf suggested a genotype-phenotype mapping (GPM) which separates the search and solution spaces of the EA. With this GPM, the genetic search continues in the search space, and the GPM translates genotypes into legal phenotypic elements (points in the solution-space). Thereafter, the resultant solutions, can be evaluated for fitness in the problem environment[3].

[3]It is worth noting that a major impetus for the use of this approach, is the ability to maintain standard genetic operators over linear genomes, whereby the GPM ensures their translation to syntactically valid solutions; the significance of this will be realised in Section 3.3

In support of this argument, he highlighted Kimura's neutral theory of molecular evolution [Kimura, 1983] as a basis for his approach. This theory states, that in nature - molecular evolution is essentially driven by mutations of neutral effect: *neutral-mutations*. This means that different genotypes can precisely code for the same phenotype; (according to Kimura, this phenomenon, is a reason for high genetic-diversity in natural population). With the GPM a many-to-one genotype-phenotype mapping exists; this allows for neutral-mutations to occur - despite change at the genotype level, the resultant mutated genotype still maps to the same phenotype.

In an EA context, it has been argued that neutral-mutations may provide a *genetic-drift* towards some desirable phenotypic effect - this effect possibly moving the EA from a local-optimum to a more global peak on the fitness-landscape [Banzhaf, 1994, O'Neill and Ryan, 2003a, O'Neill and Ryan, 2001]. Without such neutral-mutations and the consequent genetic-drift; (and in many respects, due to pressures of fitness-based selection) a deceptive problem could prevent such global heights from being reachable.

In a later work, Keller and Banzhaf [Keller and Banzhaf, 1996] noted that with common GP, the genetic-diversity becomes hindered. They state that - in searching only the solution space - GP approaches face a hard constraint; issues such as the requirement of the closure principle and the use of restricted crossover as in [Montana, 1994] can cause large regions of the search space, to become inaccessible. In Chapter 7, we observe similar findings for the importance of diversity, and our conclusions (See Chapter 8) concur with these observations as to the dangers of constrained-search.

As such, the argument for a separate search and solution space is strong, and the topic of much debate. With such an approach, issues relating to a particular genotype representation (e.g. such as that of closure) can be alleviated, by the GPM explicitly enforcing a syntactical structure on the generation of solutions. Also, in allowing the possibility of neutral mutations to occur, the GPM approach is desirable; providing a possible mechanism to overcome deceptive problems with genetic-drift. Grammatical Evolution (GE), as it's core principle, utilises such an approach and provides a unique mapping process with a *degenerate genetic-code* [O'Neill, 2001], allowing the possibility of neutral mutations and the postulated advantages thereof (See Section 6.2.1).

1.4 Grammars and Grammar-Based EAs

It has been seen that EAs (through an abstraction of natural-evolution) attempt a constructive search through the space of all possible solutions to a problem (the fitness-landscape). Grammars, particularly in a generative role, have seen popular use in the field of evolutionary methods. In this section, it will be seen that grammars provide a very workable method of defining the set of all possible solutions to a problem. Aside from this, a grammar can ensure that this space of solutions, are correctly structured (syntactically-correct) - as according to the problem at hand. As such, a grammar-based EA can be constrained to operate on this syntactically-correct, subset of the space of all sentences (*the legal subset*). The next sections provide a discussion of grammars, their relative terminology, and their influence in the subset of EAs referred to as *grammar-based EAs*.

1.4.1 Grammars and BNF

A *grammar*, describes a written-method which can formally specify all the legal sentences of a given language (i.e. the set of *sentences* expressible by that language). As it happens, this language may in fact be the subset of a larger more-complex language. For example, we may have a grammar describing the language of *noun-phrases* - which, in itself is a subset of some larger more-complex language describing an entire natural language.

Grammars describing *programming* languages, derive their definition from the formal grammars proposed by Noam Chomsky for natural languages [Chomsky, 1956, Chomsky, 1959]. A grammar G can be described by the 4-tuple $< V_t, V_n, P, S >$ where:

- V_t is a finite alphabet of tokens known as *terminal-symbols* or the terminal vocabulary.
- V_n is a set of *non-terminal symbols* or syntactic categories; each defining the make-up of a *phrase* and together contributing to the *phrase-structure* of the language described by the grammar: the non-terminal vocabulary. (These terms will be further elaborated in the text which follows).
- P, a finite set of rules, or *production-rules* which describe how each non-terminal is defined in terms of the terminal and non-terminal-symbols of the grammar (*i.e. production-rules determine the structure of a phrase*).
- S, a distinguished non-terminal: the *start symbol*, from which all sentences of the grammar G are derived.

Grammars are most commonly expressed in a notation known as *Backus-Naur form* (BNF), named after the researchers who first used it in the description of the Algol60 programming language [Naur, 1963]. BNF is a metalanguage (*i.e. a language used to describe a language*), and essentially consists of the symbol '::=', meaning "is composed of"; and '|' to denote choice. An example production-rule is thus of the form:

$$< a - phrase > \ ::= \ aba \ < b - phrase > \ \ | \ aba$$

where the non-terminals take the form <*phrase-category*> (enclosed in angle brackets), and *aba* is an example of a terminal-symbol of this language's alphabet[4]. Note, that in this instance, *aba* is a group of characters or a *token* (alone, it is an atomic symbol), having meaning according to a particular language specification. Collectively, the characters of the token make-up a single terminal-symbol. A *sentence*, then, describes a string containing terminal-symbols only - each, derived from a sequential application of the grammars production-rules. The above production-rule states, that an <*a-phrase*> is composed of the terminal-symbol *aba*, followed by the grammar's definition of a <*b-phrase*>. Alternatively, an <*a-phrase*> is composed solely of the terminal-symbol *aba*. Such a production (whereby a terminal-symbol can be derived from it's application), will subsequently be referred to as a *terminal-producing production*.

In all, there were four distinct classes of grammar according to what has become known as *Chomsky's hierarchy* of grammars: where he classifies them type 0 through type 3.

- Type 0: The *unrestricted* grammars, require only that at least one non-terminal occur on the left-hand-side of a rule, for example,
  $a \ < thing > \ b \ ::= \ b \ < thing2 >$.

- Type 1: *Context-Sensitive* grammars: adding the restriction that the right side contains no fewer symbols than the left side, for example,
  $< thing > \ b \ ::= \ b \ < thing >$.

[4]A phrase-category then, provides rules to define how a particular phrase can be structured.

- Type 2: The *context-free* grammars prescribe that the left side be a single non-terminal producing rules of the form $< A > \ ::= \ a$.

- Type 3: *Regular* grammars, being the most restrictive, allow only a terminal, or a terminal followed by one non-terminal, on the right side: that is, rules of the form $< A > ::= \ a$, or $< A > ::= \ a \ < A >$.

Owing to their lack of restriction, the context-sensitive and unrestricted grammars have seen very little practical use. On the other hand, however, the family of context-free grammars (CFG) provide a structure which allows programs like compilers to be able to parse valid strings of their language (i.e. parsing, meaning to determine syntactical correctness). In the context of GE (and indeed in the context of the work of this thesis), CFGs are used as a generative tool; but it will be seen, their is a limit to their expressiveness.

For the reader interested in more detail in the theory of compilers, we refer to the canonical texts of Aho, Sethi and Ullman [Aho et al., 1986]; Fischer and Le Blanc [Fischer and LeBlanc, 1991], and for a more modern perspective focusing on programming-language processors in Java; Watt and Brown's book [Watt and Brown, 2000] is highly recommended.

1.4.2 Grammar-Based EAs

As seen in the Section 1.3, traditional tree-based GP systems suffer from the pitfalls of the closure issue - whereby the GP-function and GP-terminal set were constrained, such that the terminal set is *closed* under the legal argument-set of the functions employed. Montana's strongly-typed system [Montana, 1994], overcame this problem to an extent; again a problem arose however, requiring problem-specific analysis to define controlled genetic operators. This heavy reliance on domain-knowledge to provide suitable operators, questioned the robustness of GP - it's usefulness, and scalability to larger more complex problems.

Whigham [Whigham, 1995] [Whigham, 1996], utilised grammars to produce an initial population of candidate-solutions - guaranteed to be syntactically-correct under the definition of a CFG for solutions to the problem examined. The role of grammars in the initialisation process provided an initial population - constrained to the language of that CFG. As such, he denotes a *language-bias* through their use [Whigham, 1995]. Derivation-trees serve to represent individuals in a population; which are generated by the random selection of production-rules, chosen from a CFG in a depth-first (*left-most*) fashion. Limits are placed on the depth to which derivation-trees may grow; and a subtree-crossover, similar to that proposed by Montana [Montana, 1994], maintains a type-sensitive crossover; with crossover restricted over the phrase-structure of the language defined. This is similar to the subtree-crossover described in the experiments of O'Neill et al., for grammatical-evolution (GE) [O'Neill et al., 2003] (See Section 6.2).

Wong and Leung [Wong, 1995, Wong and S., 1997] published a similar system to that of Whigham, whereby the force directing the algorithm was a logic-grammar working with a tree-based GP system. Geyer-Schulz developed a similar derivation-tree system which also imposed a bias on the search-space; this time, utilising ideas from the field of fuzzy-logic [Geyer-Schulz, 1995].

Furthering his earlier work, Whigham devised a method to bias the structure of the grammar, as the evolutionary run progressed. This bias produced an order-of-merit for rules which showed-up to be popular in the generation of highly-fit individuals. He then added a so called *replacement* strategy, which replaced old, unfit individuals - with newly generated ones; defined now, over the biased grammar. Banzhaf, and together with Keller

[Keller and Banzhaf, 1996], introduced the use of grammars, in the context of what they term a repair mechanism[5]. In this approach, fixed-length binary genomes comprised of 5-bit codons are used, to define a many-to-one genotype-phenotype mapping process[6]. Each codon translates to the solution-symbol of a pre-defined genetic-code; and "repair", is introduced as an LALR(1) parser which scans a solution generated by the EA. Upon detecting syntax-errors in the generated sentence, and working with a pre-defined genetic-code, the parser selects syntax corrections from the LALR(1) *FOLLOW()* set [Fischer and LeBlanc, 1991, Watt and Brown, 2000]. Syntax corrections which are closest in hamming distance to the symbol specified by the original gene-value are selected. An advantage cited in this (and the influential earlier work [Banzhaf, 1994]) argues the benefit of allowing unconstrained genetic-operators; while still overcoming the closure problems of the more traditional GP tree-based representation[7].

The GADS system (Genetic Algorithm for Deriving Software) introduced by Paterson and Livesley [Paterson and Livesey, 1997] - as with Whigham, again saw the use of a CFG in a generative role; to specify the output language of the system. Like Banzhaf [Banzhaf, 1994], fixed-length linear genomes with a binary encoding, allow the use of unconstrained genetic operators. In this approach however, there is no explicit genetic-code. Rather, gene values directly represent the choice of production to be made. The genomes are comprised of n-bit integer genes, where n is sufficiently large to represent all the production-rules from the grammar. The grammar is

[5]In the context of this thesis, we see their approach as the use of an LALR(1) parser, to act - not as a repair - but more correctly, as a *decoder* for legal sentences (*Section 3.5.3*).

[6]A genome-length of 25, 5-bit codons is specified - defining a search-space of 16^{25} possible genomes.

[7]Section 3.4.2 further discusses the significance of unconstrained evolutionary operators.

adapted, such that, each non-terminal has a default production, which can be applied; in the event that the fixed-length genome was unable to exhaust them all. During the mapping process, a derivation-tree is generated by initialising the root node to the start symbol of the grammar. Starting from the left-most gene, the genome is searched for a rule corresponding to the left-most non-terminal in the derivation-tree. If a gene does not correspond to a suitable production, it is skipped and the next gene is read. Mapping terminates upon reaching the end of the chromosome. At that point, any unmapped non-terminals are replaced by the corresponding default symbols. The method of skipping unsuitable genes, is reported as a problem in the proliferation of introns (*non-coding or redundant genes*).

GADS2 [Paterson, 2002] seeks to address some of the issues found with it's predecessor. A mapping function is provided such that each gene can be used to select a production for any non-terminal. With this mapping function, comes properties of polymorphic genes and a redundant genetic code[8]. GADS2 also saw the use of CFG's coupled with attribute grammars for what is termed as context-sensitive programming.

Freeman [Freeman, 1998] used an approach similar to GADS, entitled Context Free Grammars GP (CFG/GP); again using fixed-length linear genomes with integer-valued genes. Gene values are restricted to the indexes of the rules in the grammar. The major difference with this approach, being, the ability to map an arbitrary non-terminal as opposed to just the left-most non-terminal. A gene is applied to a non-terminal regardless of its location; with the left-most being chosen only in the case of multiple candidates. If the gene can not be applied, it is skipped and the next gene is read. The CFG/GP method reduces the proliferation of introns over that of GADS.

[8] *Section 6.2.1 discusses similar properties, but in the context of GE*

Hoai et. al. [Hoai et al., 2002, Hoai et al., 2003] present an extension of the derivation-tree based GP proposed by Whigham; incorporating Tree Adjoining Grammars (TAG). Joshi and Schabes [Joshi and Schabes, 1997], define the TAG using a quintuple; with the standard grammar terminal and non-terminal sets; S the start symbol; and two sets defining subtrees are included in place of the P (*See Section 1.4.1*). I the set of *initial trees*, essentially mapping to the derivation-tree equivalent of phrase-categories; and A, a special type of derivation-tree with a leaf-node that has the same non-terminal as the root node of the subtree itself. With the TAG system, two operations of *substitution* and *adjunction* define operators analogous to the production expansion of CFG-based systems. In their work, the TAG3P system is a tree-based system, whereby the genotype records the ordering and location of adjunction and substitution operations. Applying this encoded sequence of operations yields a derivation-tree, allowing the phenotype to be abstracted in the usual way; from the leaf nodes of the tree. TAG3P utilises standard GP operators of restricted sub-tree crossover and mutation, but they are performed on the TAG tree as opposed to the derivation-tree. The resulting representation is less constrained than that of GP or grammar-based GP, whereby a wide-range of genetic operators are also supported.

Core to the work of this thesis, Grammatical Evolution (GE) has proven a very successful form of grammar-based EA [O'Neill and Ryan, 2003a] [O'Neill and Ryan, 2001]. It stands out as one of the few variable-length linear genome approaches using a binary encoding; and as pointed out in [O'Neill, 2001] - answers Koza's critique of linear systems at the time; i.e. does "the initial selection of string-length limit in advance the number of internal states of the system and the computational complexity of what the system can learn" [Koza, 1989] (See Section 3.4.1).

Deviating slightly from the standard use of grammars, Keijzer presents the *adaptive logic programming* (ALP) system [Keijzer, 2002]. Instead of using a context-free grammar, it uses a (Prolog) logic program to enlist the rules of a language's syntax: a *logic grammar*. When queried, Prolog finds all the matches to a query (in a depth-first traversal of the language), to identify constructs which may match the query. Due to its declarative nature, logic programming is very suitable for defining computer languages and any constraints there-imposed, upon them. Such a logic program then, consists of a definition of all valid computer programs; in effect, the logic program defines both a parser and a generator for the language. When such a logic program is run (using the Prolog search strategy), it will enumerate all possible computer programs in the domain defined by the logic program. When the search is for that one particular computer-program that performs best on some problem, and the number of possible programs is large, such an enumeration is not a viable search strategy. As such, the ALP system uses GE to carry out this search. The ALP system stems from the earlier inception of *inductive logic programming* (ILP) [Muggleton and Raedt, 1994].

This section has given a brief introduction of some of the more prominent grammar-based EA systems and outlined their most significant properties. It is by no means a complete survey of grammar-based methods, and we refer the interested reader to the PhD thesis' of O'Neill [O'Neill, 2001] and Azad [Azad, 2003] which provide good reviews with more detailed explanation and examples. Rather, this section has served as a precursor to a discussion of the intricacies of the GE algorithm, outlining the works from the literature which have contributed to it's inception.

1.5 Grammatical Evolution (GE)

Grammatical Evolution (GE) [O'Neill and Ryan, 2003a] is a system that can evolve sentences in any language defined by a grammar. In it's standard form, the system is described as an approach to *evolutionary automatic programming* (EAP); (this, owing to it's typical use with an evolutionary algorithm (EA) - and it's common application to problems requiring the generation of programming-language expressions). Variable-length binary genomes provide the genetic-information needed to evolve, what can effectively be called - *programs* (legal sentences of a programming language). In this way, GE can be considered as a form of grammar-based genetic programming; or grammar-based GP (See Section 1.4.2).

Lying at it's core is the use of a grammar - working in a generative role. It will be seen, that such a grammar provides a powerful expressive force; but also brings with it some limitations to which the work of this thesis hopes to address. As such, the following sections seek to introduce the concept of grammars; in particular their role in the GE algorithm. An example of the GE mapping process will be provided, with a brief survey of grammar-based systems which have contributed to it's inception; and those which have subsequently emerged.

1.5.1 GE Mapping Example

To explain the Grammatical Evolution (GE) mapping process let us take an example BNF grammar definition. The following grammar was used in [O'Neill and Ryan, 2001] to solve the Sante Fe Ant Trial problem (the Sante Fe Ant Trial problem is a standard problem in the area of GP and is fully described by Koza [Koza, 1992]). Essentially, the problem concerns the automatic programming of a control program for an artificial ant, such as to allow the ant to find 89 pieces of food located in a discontinuous trail within a specified number of time steps. The following defines a CFG for the problem,

```
N={Code, Line, Expr, If-statement, Op }
T={left(), right(), move(), food_ahead(),
   else, if, {, }, (, ),; }
```

with P the set of production-rules as:

```
     S ::= <Code>                            (0)
<Code> ::= <Line>                            (0)
         | <Code> <Line>                     (1)
<Line> ::= <Expr>                            (0)
<Expr> ::= <If-statement>                    (0)
         |<Op>                               (1)
<If-statement> ::= if (food_ahead() ) {
                       <Expr>
                   }else{
                       <Expr>
                   }                         (0)
<Op>   ::= left();                           (0)
         | right();                          (1)
         | move();                           (2)
```

The GE mapping process works, by first constructing a representation of the grammar, and also a table to denote the *number-of-choices* - or productions available for each non-terminal. Table 1.1 describes such a table, while the grammar representation works - such that, left-hand-side (L.H.S) non-terminals are used as a key to a corresponding right-hand-side (R.H.S) list of

Rule	**Number Of Choices**
<Code>	2
<Line>	1
<Expr>	2
<If-statement>	1
<Op>	3

Table 1.1: A GE lookup table for the Sante-Fe Trail grammar.

possible production-rules (i.e. the index of which are specified in parenthesis). As it can be seen in Figure 1.3, the representation allows a *lookup* of each L.H.S non-terminal to produce a list of the possible re-write, or production-rules, and in conjunction with the table, the number of choices available for each rule can be readily accessed. (*Note:* Any potential representation of the grammar can be used - in this instance, for the purpose of explanation we choose a hash-map).

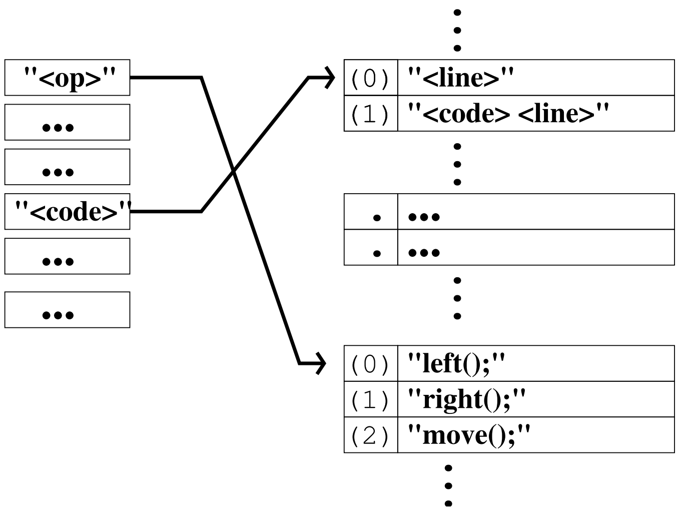

Figure 1.3: An example of the GE lookup table implemented as a hash-map.

The GE algorithm operates by modeling a left-most derivation of the pre-specified grammar; i.e. productions are sequentially applied to the start symbol S, such that at each juncture, the left-most non-terminal is always expanded to some R.H.S production (as defined by it's corresponding rule). A variable-length binary genome acts as an encoding for such a derivation, whereby 8-bit chunks, or *codons*, are sequentially read to determine which production-rule is applied to the *current non-terminal* (*i.e. the left-most* V_n). In this way, the GE algorithm begins to derive legal sentential-forms over the grammar, exhausting each non-terminal to it's corresponding R.H.S production; until finally, a sentence comprising all terminal-symbols exists. In essence, GE defines the mapping of the binary genome, to a derivation of the grammar. Production-rules are chosen by deriving the production, at the index of the current non-terminal's rule-list; as specified by the following formula:

$$Rule \quad = \quad IntegerCodonValue \,\%\, NumberOfChoices \tag{1.1}$$

where % represents the modulus operator.

Figure 1.4 provides an illustrated example of the derivation of a valid control program for a Sante Fe ant. The start production $< S >$::=$< Code >$, where no choice exists, implies, we begin the parse with the non-terminal $< Code >$. Turning then, to the table representation of the grammar, and the randomly or evolutionary generated individual (*i.e. created by initialisation, or evolutionary operators*) - the derivation begins.

As Figure 1.4 depicts, the first codon is read; transcribed to it's decimal value; and a rule choice for the $< Code >$ non-terminal, is selected by equation 1.1. The resulting production chosen is that of the 0^{th} index in the R.H.S rule-list for the $< Code >$ non-terminal (*See Figure 1.3*). Similarly, the process is continued; at each step exhausting the left-most non-terminal until a

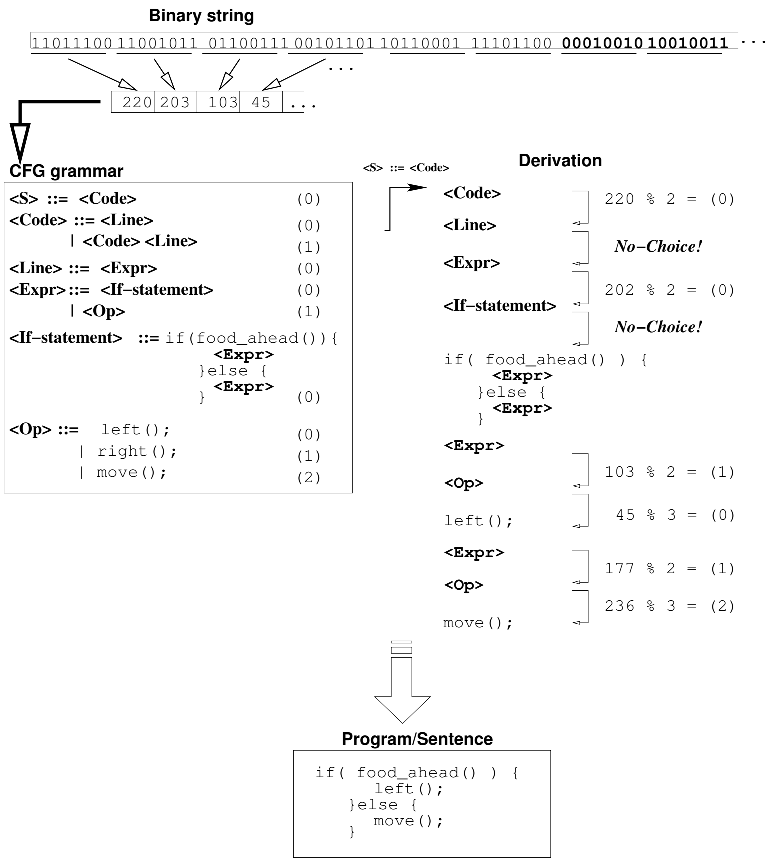

Figure 1.4: An example, illustrating the GE mapping process.

sentence of terminal-symbols only, remains[9]. This process of transcription, and subsequent translation to a phenotypic trait, stems from a biological analogy of protein synthesis [O'Neill and Ryan, 2003a, O'Neill and Ryan, 2001, O'Neill, 2001]; whereby the binary genome reflects the DNA of a biological organism, and the rules of the grammar are seen as akin to the role played by amino acids in the building of a protein.

[9]*Note:* A recent extension of the GE mapping process deviates from the standard restriction to a left-most derivation, and allows the order of non-terminal derivation to be encoded into the genome, in what has been termed πGE [O'Neill et al., 2004a].

It can be seen from the example, that where no choice of production exists, no codon is read. In addition and commonly occurring with the variable-length representation employed - it can be seen that residual codons exist; these are codons which are not used in the derivation. Codons of this type will subsequently be described as *residual-introns* (*further discussed in Chapter 5*), or more commonly as a *GE-tail* [O'Neill and Ryan, 2003a, O'Neill, 2001]. Alternatively, it may occur that the binary genome doesn't have enough codons to completely exhaust all non-terminal symbols to a sentence of the language. In this case, either the GE *wrapping* operator is employed - allowing the genome to be "wrapped" or re-read over-and-over up to some maximum number of wraps (defined as a parameter to the system); or the offending individual is given the worst-possible-fitness, such that it will eventually be eradicated from the population by the replacement strategy.

1.6 Thesis Outline

The following provides a brief outline of content, for all subsequent chapters in this thesis.

Chapter 1: Grammars For Constrained-Optimisation Summarising this first chapter then, we have presented a high-level overview of the process of artificial-evolution; pointing out it's key processes of - selection, offspring-creation (an operation of gene-exchange), and replacement. Evolutionary Algorithms (EAs) are discussed in the context of the separation of search and solution spaces: an argument for this approach is presented. Grammatical-Evolution (GE) is outlined, describing grammars and the BNF notation, Grammar-Based EAs (a brief review of works contributing to the inception of GE); and a GE mapping-process example, are presented.

Chapter 2: Knapsack Problems This chapter presents an overview of knapsack problems; with a particular focus on the 01 multi-constrained problem (01MKP), of most relevance to this thesis. A review presenting essential-knowledge of the 01MKP itself, (and in separation) required-knowledge for the understanding of the more-advanced works in the field of EA approaches to knapsack-problems, is presented. The aim of this chapter, is to equip the reader with the essential-knowledge; whilst providing a point-of-reference when encountering the more-advanced works for the first time.

Chapter 3: Review of Approaches to Knapsacks This chapter provides a survey documenting the efforts of past-research. The goal of this chapter attempting to provide the aspiring practitioner with all the required knowledge, to understand the work from the literature.

Chapter 4: GE, Attribute Grammars and Knapsack Problems Drawing knowledge from Chapters 1,2 and 3; Chapter 4 describes - the what, the why and the how: discussing, the problem faced by the existing GE system; the reasoning behind why this should be overcome; and the method by which this can be achieved (i.e. an attribute-grammar decoder for knapsack problems). We draw correlations between the fields of grammar-based GP and EAs for constrained-optimisation, before initial experiments are detailed and first-results presented.

Chapter 5: Analysis of Introns From the initial-results of the experimental systems under evaluation, Chapter 5 details a logical-trail of analysis into, the genetic make-up of the evolving population. Subsequently, experiments with intron-removal techniques led-us to a finding of improved performance; and that the presence of introns appear to be of benefit to search.

Chapter 6: A Closer Look at Evolution In the absence of any real conclusive evidence, Chapter 6 takes a view of the attribute grammar system, from a macro-level. An analysis looking now at, the driving force of evolution (or learning) - the artificial-evolution algorithm (or the EA) itself. The core processes of same (as identified in Chapter 1) are considered before an argument for the use of crossover as a black-box analysis metric over an untouched experimental system is proposed.

Chapter 7: Phenotypic-Duplicate Elimination This chapter takes a brief survey from the literature, detailing works which study the effect maintaining the population diversity of the EA; particularly for approaches with a redundant many-to-one genotype-phenotype mapping. Experiments for *phenotypic-duplicate elimination* are reported.

Chapter 8: Conclusions and Future Work The thesis conclusion - a brief summary of work carried-out, conclusion and directions for future research.

1.7 Contributions of Thesis

- A review of knapsack problems, with a focus on 01MKPs in an attempt to provide a one-stop point-of-reference for the understanding of the most advanced works in EA approaches to same; their solutions pointed out as having a context-sensitive nature to them.
- A review of EA approaches to knapsack problems, again in an attempt to provide a one-stop point-of-reference for detail in understanding same. A new/novel method for distinguishing between *decoder* and *repair* based on evolutionary learning, is proposed.

- Attribute grammars - are proposed as a mechanism to extend GE for problems of constrained-optimisation (or indeed any problems with an inherent context-sensitive nature). They allow the transformation of GE to the role of a *decoder* for feasible-only solutions, to constrained problems. Their incorporation into a new, extended GE mapping process is proposed; where the introns (or non-coding codons) emerge as a consequence.

In the context of the problems examined:

- The existence of the GE-tail of residual-introns - has been demonstrated as being essential to the successful operation of GE's ripple-crossover.
- Splicing (the use of an intron-removal technique which splices interspersed-introns or IIs), is seen to provide (the best and) improved performance over all other experimental set-ups.
- Analysis provides evidence to suggest that GE's ripple-crossover undergoes a mid-evolutionary change-of-role to become a destructive force; the population is seen to self-adapt in response - via "compression" of the effective-region to effectively cancel-out the operation of the crossover operator.
- Population diversity, and it's maintenance at a phenotypic level, is observed to improve performance (in accordance with many similar findings from the literature). Questions are raised as to the imbalance between the increased redundancy of the extended GE mapping process and the explorative capabilities of the traditional GE one-point crossover operator.

Chapter 2

Knapsack Problems

2.1 Introduction

Knapsack problems have undergone intensive study and are a hotbed of research since the pioneering work of Dantzig in 1957 [Dantzig, 1957]. Such interest stems from their immediate application to industry and financial management. They have had a more pronounced theoretical interest however, owing to their frequent occurrence in integer-programming problems. This chapter seeks to provide the reader with an introduction to knapsack problems, and serves to give an understanding of their fundamental properties; particularly those of most relevance to this thesis.

As described by Chinneck [Chinneck, 2000], knapsack problems fall into a category of combinatorial optimisation known as linear-programming (LP), where "programming" is an old meaning of the word 'planning'; so, essentially *linear-programming* means planning, or modelling a problem; where such models are linear in nature[1]. In short, LP deals with optimisation

[1]This can be thought of, as the behavior that - when a problem-size increases, so too does the effort to solve it; and in direct-proportion (i.e. they have a linear relation).

problems detailing very large numbers of variables (*e.g. sometimes millions*) where variables can be real-valued (fractional in nature). In this field, exact-methods such as *branch-and-bound* overcome the phenomenon of combinatorial explosion (where increasing the number of variables results in an overwhelming increase in the search-space) by exploring only the most promising areas of the entire search-space (sometimes referred to as, the *fully-enumerated* search-space). The search-space is explored by a directed search through a tree representation of all solutions; expanding the promising areas only. That is, search is guided such that at each level of the tree; only the most promising area is explored. Promising areas are identified by estimating a bound on the best objective value that could possibly be achieved, by exploring that point in the search space. Chinneck provides an excellent introduction to practical optimisation, and in particular; simple high-level descriptions of the workings of the branch-and-bound method [Chinneck, 2000].

The field of linear-programming deals with real-valued variables; yet, often practical problems require the modelling of a situation using integral values. As seen in the introductory chapter, common real-world usages of optimisation deal with airlines, delivery companies, manufacturers and the like. Such fields require scheduling of say, 6 or 7 people as opposed to 6.3, for example. Knapsack problems, predominantly fall into this category of real-world problem abstraction. LP solving, in it's less-constrained real-valued form, is generally used as a sub-step to solving the more constrained integer-based real-world problems; the real-world abstraction entering the domain of *integer-programming.*

Within the complex domain of integer-programming, real-valued knapsack problems (their LP generalisation), occur at each estimation of a bound for a point in the search space. As a consequence, techniques capable of efficiently solving very complicated knapsack problems; are highly sought after. In recent years Evolutionary Algorithms and their hybrid coupling with exact-methods have proven to be the most effective method of solving complex integer-based knapsack problems [Raidl, 1998, Chu and Beasley, 1998, Raidl, 1999b, Gottlieb, 2000]. That is, when problems are of an integral nature (*i.e. binary or integer programming form*), hybrid EA's can outperform exact-methods over large problem instances. When dealing with the more relaxed LP problems, however, exact-methods such as that of the MIP(Mixed-Integer Programming) solver CPLEX [CPLEX, 2005] outperform all other approaches.

The following sections detail the intricacies of knapsack problems and their fundamentals; but with a particular focus on those studied in this thesis, in what is introduced as the *01 multi-constrained* knapsack problem.

2.2 The Problems

Common to all knapsack problems, are a set of all items from which we must choose a subset. Each item has some objective attribute, most commonly profit (p_j); and some constraint-based attribute such as weight (w_j). A subset of the set of all items must be chosen for inclusion in one or more knapsacks (or containers) with a given capacity (c_i). Co-efficients of an item's attributes are generally positive integer values. In the context of this thesis, we discuss a binary-programming variant of the knapsack problem, the *Zero-One* (01) knapsack problem; where the inclusion of an item is a binary decision. An item is either in, or not in. As such, the category of knapsack problems subsequently discussed, all stem from what is commonly termed the *single-constrained* knapsack problem, whereby we have just one knapsack. It can be formulated as the following maximisation problem:

$$\begin{aligned} &\textit{maximise} && \sum_{j=1}^{n} p_j x_j && && (2.1)\\ &\textit{subject to} && \sum_{j=1}^{n} w_j x_j && \leq c, && (2.2)\\ &x_j \in \{0,1\}, && j = 1 \ldots n && && (2.3) \end{aligned}$$

where $x_j = 1$, indicates inclusion of an item in the chosen subset; and an $x_j = 0$, indicates exclusion of an item. A viable solution can thus be represented by $\vec{x}$, a vector such that $x_i \in \{0,1\}$. As can be seen from equation 2.1, solutions are constrained by the capacity of the single knapsack c. Owing to the single constraint this problem is commonly referred to as the 01 *single-constrained* knapsack problem.

Deviating from the binary-programming or 01 knapsack problems, multiple items (of the same type) can be included in the knapsack for an *integer-valued* knapsack problem. That is, the number of items of the same type, that can be included in a knapsack is bounded by some integer value (generalising here from binary, to the domain of integer-programming).

It is worth noting here, that with an integer-valued knapsack problem, components of the vector $\vec{x}$ are positive integer-values; and not bounded to a zero or a one as is the case with a 01 problem. The positive integer, will then have a multiplicative effect on the evaluation of profits (equation 2.1) and weights of a viable knapsack (equation 2.2). For example, a viable solution-vector to a 4 item integer-valued problem may be $< 3, 6, 0, 1 >$: here, both the profits and weights of x_1 will be multiplied by 3; x_2 by 6, and so on. This is further examined in section 2.4.1.

2.3 Fundamental Properties

As stated in Pisinger's PhD. thesis [Pisinger, 1995], knapsack problems are highly structured; and problem instances, can and have been formulated, with varying approaches to such structure. Recall from the previous section that a solution can be seen to be a vector which models a set-of-items $\vec{x} \subseteq \{x_1, x_2, x_j \dots x_n\}$. Items have a single profit, and one-or-more weight attributes associated to them. The set-of-items we choose is constrained by the maximum weight our containers can hold. In solving large-scale problem-instances it is pertinent to understand some fundamentals of the problems.

The goal or objective-function of knapsack problems is stated in terms of maximising the profit of a set-of-items, while keeping to a minimum the weight of this set. As a result, a fundamental property of optimal solutions to these problems is that items in the chosen set have higher profits and lower weights where possible. That is, the profit-to-weight ratio or *profit density* δ_i, can be used as a measure of the utility of any given item [Cotta and Troya, 1998, Pisinger, 1995]. It can be seen that a 'greedy' sorting of items, or a sort in terms of benefit by utility, will yield the optimum solution for the single-constrained problem. However as pointed out by Chu and Beasley "...when more than one constraint is present, it is not quite clear how this approach can be generalised"[Chu and Beasley, 1998]. As they point out, several methods for *pseudo-utility ratio's* have been researched in answer to this uncertainty; where almost all rely on a relaxation of the problem to produce some estimate of an items overall profit-density (See Section 2.5). That is, a pseudo-utility ratio refers to an estimation of a single profit-to-weight ratio or profit density for the m so-called *relative-weights* (See Section 2.4).

This 'greedy' sorting of the set of all items, is a technique which has been used in the majority of hybrid approaches [Raidl, 1999b, Raidl, 1998, Chu and Beasley, 1998]. In this context, sorting is incorporated into a hybrid local-optimisation technique or *feasibility-algorithm* (*further discussed in Section 3.5*).

The fundamental properties outlined in this section remain true to all categories of knapsack problem, where a key goal guiding the generation of a good solution; remains to be, the accurate estimation of each item's 'worth' in the context of a given viable solution. In the following sections, we switch the focus from the general properties to the specific attributes of the problem tackled in the body of this thesis' experimental section; *the 01 multi-constrained knapsack problem.*

2.4 The 01 Multi-Constrained Knapsack Problem (01MKP)

The 01 multi-constrained knapsack problem (or multi-dimensional knapsack problem) identifies a scenario, where we have a number of knapsacks (or containers); each to be filled with the same set-of-items. Each knapsack has an independent maximum capacity or *weight-constraint.* We must select a subset of the set of all items (the *vector of items*), for inclusion in all knapsacks; such that the combined weight of this chosen subset, doesn't violate the weight-constraint of any of the knapsacks. As an added condition of the problem; the weight of an item is variable, and it is determined with respect to which knapsack it is included in. This will be termed the *relative-weight* of an item. As a consequence, a possible solution or chosen vector of items will have varying weight in each knapsack.

Although the weight of an item is variable, an item has a fixed value or profit. Thus the goal or objective for this problem, is to select a vector of items, with maximal profit or worth, whilst respecting the weight-constraints of all knapsacks. The problem can be formulated as

$$\begin{aligned} &\textit{maximise} && \sum_{j=1}^{n} p_j x_j && (2.4)\\ &\textit{subject to} && \sum_{j=1}^{n} w_{ij} x_j \le c_i, \qquad i = 1 \ldots m && (2.5)\\ & && x_j \in \{0,1\}, \qquad j = 1 \ldots n && (2.6) \end{aligned}$$

Where, p_j refers to the profit, or worth of item j, x_j refers to the item j, w_{ij} refers to the relative-weight of item j, with respect to knapsack i, and c_i refers to the capacity, or weight-constraint of knapsack i. There exist $j = 1 \ldots n$ items, and $i = 1 \ldots m$ knapsacks.

The objective function (equation 2.4) tells us to find a subset of the possible items (i.e. the vector of items); where the sum of the profits of these items is maximised, according to the constraints presented in equation 2.5. Equation 2.5 states, that the sum of the relative-weights of the vector of items chosen, is not to be greater than the capacity of any of the m knapsacks. Equation 2.6 refers to the notion that we wish to generate a vector of items, of size n, whereby a 0 at the i^{th} index indicates that this item is not in the chosen subset and a 1 indicates that it is.

As described in [Khuri et al., 1994], it is also worth thinking of the problem as a matter of resource allocation. That is, we have m resources (knapsacks) and n tasks. Each resource has a budget or knapsack capacity C_i, and W_{ij} represents the consumption of resource i by task j. Thus, $task_n$ may then have a different resource-consumption, depending on which of the m resources it is applied to (i.e. it's *relative-weight*). The objective then, is to select a set of tasks to be applied to all resources simultaneously, such that, the budgets of each resource are respected, and the consumption of resources is optimised.

2.4.1 MKP as Standard Form LP

The introductory sections of this chapter discussed the methods of Linear Programming (LP) as the most commonly applied form of tackling constrained-optimisation problems. Although stated above as a standard algebraic expression, the MKP is often described as a standard form linear programming model (standard form LP). Different authors vary in the form of expression, and often switch between the two. Generally the problem is introduced as a standard algebraic expression; but for example, Bruhn and Geyer-Schulz adopt the standard form LP to describe examples in their journal paper [Bruhn and Geyer-Schulz, 2002]. The standard form LP has the following characteristics: *a*) an objective function to be maximised; *b*) constraints of type *less-than-or-equal-to* ($\leq$); *c*) constraint right-hand sides which are non-negative, and *d*) variable-bounds are restricted to non-negativity.

Thus, in the standard form LP, as described by Chinneck [Chinneck, 2000], the MKP may take on the following algebraic representation for a problem with m constraints (knapsacks) and n variables (items):

- **Objective Function:** $maximise\ Z = p_1x_1 + p_2x_2 + \ \ldots + p_nx_n$ where p_i indicates the profit coefficients for each item x_i. For example, $maximise\ Z = 300x_1 + 100x_2 + 0x_3 + 10x_4$, would indicate the profit coefficients of the set of 4 items to be the ordered list $(300, 100, 0, 10)$ objective function units respectively. A corresponding viable solution vector $\vec{x} =< 0, 1, 1, 1 >$ would infer a profit evaluation of 110 objective-function units. As previously stated, a viable solution to an integer-valued problem would see a multiplicative effect on these co-efficients ($e.g.$ *$\vec{x} =< 2, 4, 0, 0 >$ would imply an evaluation of 1000 objective-function units for Z*)

- m **Functional Constraints,** describing problem constraints as functional inequalities

$$\begin{aligned}
&w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \ldots + w_{1n}x_j \leq c_1 \\
&w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \ldots + w_{2n}x_j \leq c_2 \\
&\qquad\vdots \\
&w_{m1}x_1 + w_{m2}x_2 + w_{m3}x_3 \ldots + w_{mn}x_j \leq c_m
\end{aligned}$$

  where c_i are the resource constraints for knapsacks; and the w_{ij} are the *relative-weights* of the j^{th} item with respect the i^{th} knapsack. In this way, the sum of the weights of all items, included in the current viable solution; must not exceed the capacity of the knapsack for which their weight-determination is relative.

- n **Non-Negativity constraints:** $x_j \in \{0, 1\}$

  Such that, each index of $\vec{x}$ the candidate solution, holds a non-negative value of either zero or one (*thus describing the 01 variant*). Note that non-negativity constraints for an integer-valued version would be expressed as $x_1 \geq 0, x_2 \geq 0 \ldots x_n \geq 0$.

It can be seen that the above expression of the problem maps to the standard algebraic equations of equation 2.4, 2.5, and 2.6. The standard-form LP model then, serves as an expansion of the standard algebra; and its explanation provides the reader with the relevant knowledge required for it's understanding.

2.5 Relaxations

A *relaxation* to a knapsack problem generally refers to a break from the more constrained (or more specific) optimisation problem, to the more general. For example, the 01 binary-programming problem can be transformed to a less constrained linear-programming problem when bounded by the fractional values in the range [0,1] (as opposed to integral 0 *or* 1). Similarly, transforming a multi-constrained problem to an approximately equivalent single-constrained problem, is said to *relax* the problem. Relaxations to the 01MKP have been commonly used in the literature, particularly in the application of hybrid algorithms to perform local-optimisation - but more frequently in the performance evaluation of very-large problem instances [Chu and Beasley, 1998] [Raidl, 1998] [Raidl, 1999b] [Gottlieb, 1999b] [Raidl and Gottlieb, 1999a] [Raidl and Gottlieb, 1999b] [Raidl and Gottlieb, 1999c].

This section seeks to equip the reader with knowledge of the most common relaxations referred to in the literature; particularly when solving complex knapsack-problems. Generally, such relaxations are used as a performance-metric for very large instances of knapsack-problem - or as a sub-process in solving them. For single-constrained problems, exact-methods alone (without the need for relaxations), suffice.

Relevant in the field of EA approaches to constrained-optimisation problems, there are predominantly two categories of relaxation *1)* heuristic guided relaxations; and *2)* those just relaxing the integrality property. In the latter, we slip from *integer* linear programming (ILP), to the less constrained domain of linear programming (LP), in what is termed *the LP-relaxation*; in this field - exact-methods thrive. In the former - heuristic-guided mathematical formulae resolve a multi-constrained problem to (an approximately equivalent) single-constrained problem for solving.

The LP-relaxation is often used, as a measure of performance for very difficult ILP problem-instances (instances which are computationally intractable to the point that the value for an optimal-solution is unknown). An LP-relaxed solution is generally used to give an upper-bound on the maximum possible objective-function value for these unsolvable ILPs (e.g. a large 01MKP instance). As such, the *gap* between the best LP-relaxed solution achievable (by exact-methods) - and that achieved by some new approach; serves as an upper-bound to the ILP's optimum. In practice, this performance measure is generally referred to as the *percentage gap to the LP-relaxed solution* [Raidl, 1999b, Raidl and Gottlieb, 1999a, Raidl and Gottlieb, 1999b, Chu and Beasley, 1998, Raidl, 1998].

In terms of heuristic-guided relaxations to the 01MKP the *surrogate relaxation* (SR) approach of Pirkul [Pirkul, 1987], and the *Lagrangean-relaxation* of Magazine and Oguz [Magazine and Oguz, 1984] predominate. These relaxations have been empirically considered in [Raidl and Gottlieb, 1999b] [Raidl and Gottlieb, 1999a] [Raidl, 1999b] [Chu and Beasley, 1998] [Raidl, 1998], and all have concluded to the better performance of the SR relaxation to the problem. As a consequence we omit the Lagrangean relaxation from our discussion and briefly outline the basic workings of the surrogate method. First, let us consider the LP-relaxation.

LP Relaxation The most widely known relaxation to the 01MKP is the linear-programming or LP-relaxation to the problem. This relaxation is just as previously described, where the problem now takes the form of what Pisinger refers to as a *bounded* problem (in that context, bounded in the range[0,1] [Pisinger, 1995]). As pointed out in [Chu and Beasley, 1998] exact-methods such as the CPLEX MIP solver [CPLEX, 2005] have absolutely no problem in solving the LP-relaxation to the problem. To date, they are by far and away the most efficient and successful methods to solve this relaxed version of the problem.

The LP-relaxed solution can provide an approximation to an optimum solution for an ILP problem-instance; where the possibly infeasible LP solution, provides an upper bound on the maximum objective-function value obtainable for the ILP problem. However, this does **not** imply that simply rounding the LP-relaxed values will provide the optimum. On the contrary, Chinneck shows the dangers of rounding values in constrained optimisation; clearly showing that adopting this approach will, more often than not, produce an infeasible solution [Chinneck, 2000].

Surrogate Relaxation The fundamental properties of knapsack problems (*Section 2.3*), told us that a greedy sort of items by *profit-density* or *pseudo-utility ratio*, can yield the optimum solution for the single-constrained problem. Also highlighted was the uncertainty inherent in calculating a pseudo-utility for a multi-constrained problem. That is, when dealing with a single weight constraint; it is logical to think that a high pseudo-utility will suggest a more beneficial item than a low pseudo-utility. The situation becomes fuzzy when each item has a list of weights associated with it: one weight for each knapsack (*relative-weight*). The surrogate relaxation is a heuristic method to transform the multi-constrained problem to a single constrained equivalent. A set of surrogate multipliers a_i (i=1 $\ldots m$) which satisfy the following constraint:

$$\sum_{j=1}^{n}\left(\sum_{i=1}^{m} a_i w_{ij}\right) x_j \leq \sum_{i=1}^{m} a_i c_i \tag{2.7}$$

are used to develop μ_j, the *pseudo-resource consumption* of an item. Both Chu and Beasley [Chu and Beasley, 1998], and Raidl provide prominent examples of this relaxation [Raidl, 1999b]. This is essentially a best-guess approximation to the pseudo-utility of an item when dealing with multiple *relative-weights*. Any set of weights which satisfy the constraint of equation (2.7), will suffice to transform the multi-constrained problem to an approximately equivalent single-constrained version. Common practice sees the values of the LP-relaxed solution being used as the values for legal surrogate multipliers. Chu and Beasley state, "One of the simplest methods to obtain reasonably good weights is to solve the LP-relaxation of the original MKP ... and use the values of the *dual variables* as the weights" [Chu and Beasley, 1998]. Subsequently, a greedy sort in terms of p_j/μ_j will yield a locally-optimised sorting of the items in terms of how beneficial they are for inclusion in a near optimal solution.

Let us look now at the details of data-sets used to define different problem-instances, and the terminology used to describe such data sets.

2.6 A Note on Problem-Instances

As previously discussed, items of a knapsack problem have attributes of weight and profit. Using this information, and knowing that a knapsack capacity must exist to constrain the problem; problem-instances can be formulated as test-data. An instance, or a *problem-instance* is generally categorised by the number of items involved (*i.e. a 100 item problem*). For multi-constrained problems, the number of constraints will also be included (*i.e. a 100-50 problem*), to indicate fifty constraints, or knapsacks.

A problem-instance consists of a data-file containing a list values for profits (p_j's), possibly multiple lists of values for weights (w_j's) or relative-weights (w_{ij}'s), and a list of knapsack capacities or constraints (c_i's). These correspond to the attributes for each item in the problem. For a multi-constrained problem then, *relative-weights* will be represented as a list of m weight values, bound to each particular item; possibly on n separate lines for each item. In any case, items will have but a single profit-value bound to them[2].

New problem-instances can then be formulated. Given the aforementioned knowledge, of the properties of knapsack problems; a test data-set can be created where attributes are randomly generated; but bounded by the following concerns. Capacities are bounded, relative to the summed value of the m item-weights (determining the *tightness* of constraints); with profits also bounded relative to weights (defining the *correlation*, or pseudo-utility relation).

As a consequence, the difficulty of a problem instance is referred to in one of two ways: *a*) *Correlation of the generated instance*, and; *b*) *Tightness of constraints*. *Correlation* refers to the relatedness of weights to profits. In *highly-correlated* data-sets, profits are generated in direct proportion to the weights; whereas, in *weakly-correlated* data-sets, the relation between objective-values, and constrained-attribute values is weak [Michalewicz, 1996].

Constraint-bounds can be set to varying degrees of tightness. Generally this is achieved by expressing knapsack capacities as a percentage of the total weight of all items. For example the *tightness-ratio*, $\alpha \in \{0.25, 0.5, 0.75\}$ is commonly used to generate capacity constraints (c_i); according to the following simple formula[3]

$$\alpha \sum_{j=1}^{n} w_j$$

[2]Deviating from this definition, defines a multi-objective knapsack problem. Refer to Zitzler's work for detail on these [Zitzler, 1999].

[3]Specialised to: $\alpha \sum_{j=1}^{n} w_{ij}$ for the multi-constrained problem

The values of α determine the *restrictivity* of the problem. Other values for α may be used, but these are the most common throughout the literature. As stated in [Michalewicz, 1996] Martello and Toth report that increasing the capacity (*or, decreasing the restrictivity*) does not increase the necessary computation time for classic algorithms [Martello and Toth, 1990].

For more information on the generation of problem-instances, and suggested procedures, Michalewicz [Michalewicz, 1996] provides more detail, and Chu and Beasley [Chu and Beasley, 1998] cite a more canonical point of reference in the work of Freville and Plateau [Freville and Plateau, 1994]. A standard set of greater than 270, 01MKP problem-instances have been generated in [Chu and Beasley, 1998], and are available at the OR Library [Beasley, 1990][4]. These instances have served as a common ground for the comparison of the performances of new algorithms and approaches. In essence, the OR library serves as a common-area available over the world wide web, set-up as a point of reference for a distributed set of test-data covering a variety of optimisation problems.

2.7 Conclusion

This chapter has served as an introduction to knapsack problems. Trailing from it's theoretical grounding as a linear-programming problem - we present an overview of the key properties of knapsack problems; with a view to equip the aspiring EA practitioner - with all the necessary background knowledge to fully-understand the works (from the literature) of EA approaches to knapsack problems. Only the information of most relevance to this thesis is detailed, giving reference to more detailed works when required.

[4]OR Library available at http://www.brunel.ac.uk/depts/ma/research/jeb/info.html

The common properties of all knapsack problems have been outlined (in a general description of the single-constrained problem); from which most *if not all* knapsack problems are derived. A discussion on the fundamental properties of this family of problems, revealed a common approach-to-solution; in the concept of an item's *pseudo-utility*.

A focus on the particular 01 multi-constrained problem, of most relevance to this thesis, was presented; outlining the detailed properties of this problem. Owing to the *alternate* forms of expressing the problem (in particular, the examples of Bruhn and Geyer-Schulz [Bruhn and Geyer-Schulz, 2002]) - it's alternative expression as a standard form linear-programming model is explained (standard form LP). A review of the literature also revealed, that state-of-the art approaches to solving this problem utilise local-optimisation techniques, heavily based on, or working with - relaxations of the problem. Relaxations are defined and a discussion of the uses and methods of the most prominent disclosed. Finally, a section devoted to the aspiring practitioner, outlines the data-sets used to test new methods; and describes the terminology used to describe them. This knowledge is deemed a requirement, for those wishing to gain a full-understanding of both the past works from the literature, and the work of this thesis.

Chapter 3

Review of EA approaches to Knapsacks

3.1 Introduction

There exists some confusion in how to classify evolutionary approaches to knapsack problems. Various representations for the evolutionary chromosomes; and in conjunction, differing strategies to ensure that - both a feasible and optimal solution results from the evolutionary search - have been developed. As a result, the different authors discuss similar concepts, but with slightly differing terminology (or overlapping classification). From a review of the literature, we see the application of EAs for constrained-optimisation: primarily, as a decision for the encoding of the problem (i.e. the search-space explored); and thereafter - a twofold consideration pertaining to the core-design decisions facing the designer of an EA for knapsack problems: the *choice of representation*, and the *choice of approach to feasibility*[1].

[1] *Feasibility* refers to the wish to ensure the outcome of search is a solution satisfying all of the problem-constraints.

The chapter begins, outlining ambiguities from the literature, which face the designer of an EA for knapsacks; therein, we outline the motivation behind our subsequent choice of classification (the structure of the chapter). Following from this, we present a review of the literature, within this classification. The primary choice for problem-encoding is first discussed, dividing it into a decision between a direct or indirect approach. The choice of representation details concerns of variable versus fixed-length approaches, and their subsequent affect to the artificial evolution algorithm. The choice of approach to feasibility is subdivided into a decision between maintaining a population *allowing* or *omitting* infeasible solutions. As they are of most relevance to the approach taken in this thesis - and among the most successful approaches - the remainder of the chapter focuses on the latter choice of maintaining a population of feasible-only solutions. Having provided all the necessary background-understanding required to appreciate the most prominent approaches from the literature, the chapter concludes with a review of such EA approaches to knapsack problems.

Where possible, we present the core concepts for each section in isolation; and follow with a discussion of same, from the works of the literature. In this way, the reader familiar with the topic-area may choose to view the discussion alone, or alternatively the unfamiliar reader may gain an understanding for the concepts without being overcome by the detail of the discussion.

Finally, the forthcoming review of the literature for knapsack problems, chooses a level of abstraction of most relevance to the reader of this thesis. If wishing to assess design-considerations for the application of EAs to constrained-optimisation - we refer the reader to the work of Michalewicz (which details an implementation-specific focus) [Michalewicz, 1995]; similarly - but for an understanding of how and where constrained-optimisation affects the standard EA - Yu and Bently present a classification-framework [Yu and Bentley, 1998] outlining methods to generate legal-phenotypes.

3.2 Existing Ambiguities

In setting about applying an EA to knapsack problems, a review of the literature may leave the inexperienced designer confused. This confusion, lies within the differing authors', preference for explanation. For example, the works of Hinterding [Hinterding, 1999]; Raidl and Gottlieb [Raidl, 1998] [Raidl and Gottlieb, 1999a] [Gottlieb, 1999a] [Raidl and Gottlieb, 1999b] [Raidl and Gottlieb, 1999c]; Michalewicz [Michalewicz, 1996] and Bruhn and Geyer-Schulz [Bruhn and Geyer-Schulz, 2002]; each classify the different approaches available to the designer, often describing the same concepts with slightly differing terminology or overlapping classification.

Hinterding [Hinterding, 1999], whilst discussing the single-constrained problem gives possibly the best classification; dividing the task of problem approach into a choice of representation and constraint-handling technique. Michalewicz [Michalewicz, 1996] (also discussing the single-constrained problem), briefly describes feasibility issues; in terms of - penalty, repair, and decoder methods (See Section 3.5). Gottlieb [Gottlieb, 1999a], classifies approaches by the search-space explored; introducing the notion of a *direct* and *indirect* approach; whilst together with Raidl [Raidl and Gottlieb, 1999a, Raidl and Gottlieb, 1999c], they classify approaches in terms of decoder-based EAs. Bruhn and Geyer-Schulz [Bruhn and Geyer-Schulz, 2002], as with Michalewicz, choose to classify the approach in terms of feasibility.

Owing to a position of retrospect and the benefit of hindsight, this thesis takes a very simple approach to classify the considerations for an EA approach to knapsack problems. Issues of encoding (or search-space explored) are first discussed, with subsequent sections relating to - the choice of approach to representation, and the choice of approach to feasibility. A review focusing on required background knowledge is outlined, with particular areas of ambiguity addressed where relevant. The aim is not to re-classify; but to present the review from a position of abstraction most suited to the reader of this thesis.

3.3 Direct versus Indirect Encoding

In deciding how to represent a knapsack problem with an evolutionary algorithm, we must choose a suitable data-structure, and decipher, how a solution (to the problem) transcribes to that data-structure. That is, we determine how the problem is *encoded*. The problem-encoding can either *directly* or *indirectly* represent solutions to the problem at hand; and as discussed in the works of Gottlieb, this implicitly determines the search-space explored [Gottlieb, 1999b, Gottlieb, 1999a][2]. A *direct* representation refers to a chromosome whereby the search-space explored by the population is equivalent to $S = \{0, 1\}^n$; the space of possible solutions. That is, the problem is viewed as a search through the 2^n permutations, or possible configurations-of-items. Alternatively, an *indirect* representation refers to a search in some other space; mapped into S, by some mapping process; i.e. the separation of search and solution space (See Section 1.3.1).

[2]Gottlieb provides the definitive classification of EA approaches (in terms of search-space explored); in this section (and throughout) we subscribe to this system (and recommend it's use); our chapter incorporates this - but seeks also, to resolve naming confusion from the other works within the literature.

If indirectly encoding the problem, some form of genotype-phenotype mapping process exists. As will be seen at different points in this chapter, there are many advantages to an approach of this type; among which robustness and general applicability are to be noted. In the field of EAs for knapsack problems - direct-approaches (when coupled with hybrid local-optimisation algorithms) have proven to be the most successful [Gottlieb, 2000].

3.3.1 Permutation-Based EAs

When used as direct encodings, chromosomes encode a permutation of the set of all items. As a consequence, Gottlieb [Gottlieb, 2000], and also together with Raidl [Raidl and Gottlieb, 1999a, Raidl and Gottlieb, 1999c], use the term "permutation-based EA" to generalise a category of fixed-length representations of size n. Gottlieb uses the following relation to categorise chromosomes of this type:

$$\pi : J \rightarrow J \tag{3.1}$$

for $J = \{1 \ldots n\}$ items [Gottlieb, 2000]. The relation presents the chromosome (π), and states that their be a *one-to-one* mapping between - the chromosome and the set-of-items encoded; that is, the search-space defined is $S = \{0, 1\}^n$. Their definition of this term however, not only refers to the representation; but also it's coupling with a decoder using a *first-fit algorithm* (*Section 3.5.2*). For the purposes of uniformity - use of the term "permutation-based EA" or "permutation-based decoder", will have the same meaning and intent as Gottlieb; however the term *permutation-based representation* will be used to refer to the chromosome's representation alone[3].

[3]From Equation 3.1 - the representation requires, only that the chromosome π, represent a relation with a *one-to-one* mapping to the set-of-items; with $J = \{1 \ldots n\}$ items.

3.4 Choice of Representation

As the *problem-encoding* describes how a chromosome maps to the solution of a problem; the *choice of representation* addresses decisions for the chromosome-representation itself. For example, the type of data-structure used; whether it is of fixed or variable-length, etc.; the following outlines a discussion of these considerations in terms of the literature for knapsack problems.

Largely, there are two approaches to a chromosome's representation for knapsack problems. Chromosomes can be represented as a sequence of binary, or numeric numbers. The traditional GA approach of using binary strings of 0*'s* and 1*'s* is more common, and has shown the most success to-date; more recently however, various different types of numeric chromosomes have shown promise and have proven to be provide comparable results (if only slightly inferior). Regardless, the choice of chromosome representation can be categorised as follows:

Binary: A *binary* or *bitstring* representation, generally refers to a fixed-length n-bit binary vector $\vec{x}$, such that $\vec{x} = \{x_1, x_2 \dots x_n\}$ and $x_i \in \{0, 1\}$ (for an n item problem). An $x_i = 1$ index indicates inclusion of the corresponding item. This form of representation has emerged in the most successful approaches to knapsack problems to-date [Raidl, 1998, Chu and Beasley, 1998, Khuri et al., 1994].

Numeric-Integer: A numeric representation which generally refers to a fixed-length integer vector of size n (for an n item problem). This representation generally encodes a solution-candidate, by a *one-to-one* mapping between the numerical value of a gene and the corresponding item. The most common use of this representation views the chromosome as a configuration of the set-of-items (*i.e. a permutation*)[4]. Numeric-integer representations have been comprehensively studied in [Hinterding, 1994, Hinterding, 1999] and [Raidl and Gottlieb, 1999a] [Raidl and Gottlieb, 1999b, Raidl and Gottlieb, 1999c].

Numeric-Real: A numeric representation which generally refers to a fixed-length real-valued vector, of size n for an n item problem. Generally, a chromosome for this type of representation encodes a parameter-sequence. This sequence is then passed to a *decoder* (See Section 3.5.3), which carries out an interpretation of the chromosome in order to *build* a legal knapsack solution. Hinterding in [Hinterding, 1999], studies a random-key encoding strategy of this type and more recently Raidl's *weight-coded* GA [Raidl, 1999a, Raidl, 1999b] and Cotta's hybrid-GA utilise this representation for *problem-space search* (See Section 3.6.2).

It can be seen above, that both the binary and numeric-integer representations, readily imply the use of a direct-encoding (the exception to the case, being the stated ordinal-representation discussed in a subsequent section). In both approaches and as described by Gottlieb - approaches of this kind can be seen to be searching the solution-space $S = \{0, 1\}^n$, the space of possible solutions [Gottlieb, 1999b]. The latter numeric-real approach essentially describes a form of indirect-encoding (See Section 3.3).

[4]The *ordinal-representation* later discussed in Section 3.6.1 presents a numeric-integer representation which deviates from this definition.

3.4.1 Variable versus Fixed-Length

With regard to representation, the designer of an EA must also decide whether to use a fixed or variable-length structure for chromosome representation. Throughout the literature for EA approaches to knapsack problems, fixed-length approaches dominate, and have seen to provide the best results when coupled with hybrid-algorithms (See Section 3.7) [Michalewicz, 1996, Hinterding, 1994, Hinterding, 1999, Chu and Beasley, 1998, Raidl, 1998].

Although a fixed-length approach can very conveniently represent each of the n items of the problem; their scalability and robustness to problems of higher complexity has been questioned (See *Koza's critique* Section 1.4.2). Variable-length EA's, on the other hand readily adapt to problems of increasing difficulty. Based on a model of real-world viral-evolution, many of the evolutionary strategies (for survival) observed in that of a real-life virus - have been observed with variable-length representations simulating evolution (variable-length EAs) [Ramsey et al., 1998].

Despite their success in other problem domains, there are few examples of variable-length representations from the literature of knapsack problems; Hinterding's [Hinterding, 1994] "selection-based" GA being one of the only empirical examples[5]. Within a genetic-programming (GP) context, (whereby variable-length of the genome is inherent), one approach to knapsack problems has been reported to give successful results for the integer-valued MKP (See Section 2.2) [Bruhn and Geyer-Schulz, 2002]. Interestingly however, given the time of writing of this thesis; there have been no variable-length *binary* representations applied to knapsack problems.

[5]In this instance, variable-length being bounded by by the number-of-items; i.e. $\leq n$.

3.4.2 Representation, and Evolutionary Operators

As pointed out in the previous section's description of permutation-based EAs - a direct-encoding implicitly defines that a chromosome represent a permutation of the set-of-items. A problem arises with this approach, when using a numeric-representation: there is an imposed restriction on the generation of new chromosomes; that is, they must maintain the permutation property (*i.e.* represent a sequence of $\leq n$ numbers without repeats).

As a consequence the choice of this form of representation requires the use of order-preserving evolutionary operators. Hinterding [Hinterding, 1994], for example, reported the use of a uniform order-based crossover and swap-mutation for this purpose. Michalewicz [Michalewicz, 1996] while studying these representations, concluded that future-work should explore the use of crossover operators which are sensitive to preserving the position and order of items. Gottlieb in later years, carried out such a study [Gottlieb, 2000]: his results showing a uniform order-based crossover to prevail over a novel position-sensitive operator. A resulting conclusion pointing-out that sensitivity to order (more-so that position) is of importance for the 01MKP.

In contrast, a binary representation affords the use of classical recombination and mutation operators. Not only have such approaches proven the most successful [Gottlieb, 1999b, Raidl, 1998, Chu and Beasley, 1998], but their use also provides for a more robust system; few problem-specific changes are required to apply the same algorithm to different areas of application. In general, this advantage is afforded to most if-not-all indirect-encodings (the genotype-phenotype mapping process, and separation of search from solution space providing for this). As a consequence the *problem-space search* and the *ordinal-representation* approaches of Section 3.6; all avail of the ability to use the more classical evolutionary operators .

3.5 Choice of Approach To Feasibility

Having outlined the considerations for the choice of representation, this section serves to examine the second of the two core design decisions: the choice of approach to feasibility. A *feasible* candidate-solution[6] refers to a viable solution, which guarantees not to violate *any* of the problem constraints (i.e. it is *feasible* or makes sense to consider this solution). Alternatively, an *in*feasible candidate-solution refers to a solution - which in it's raw (un-interpreted) form - causes the violation of at least one of the problem constraints. In choosing an approach to feasibility - we essentially face a binary decision: to choose between the allowance or omittance of infeasible candidate-solutions (within the population of the EA).

In terms of the work of Cleary and O'Neill [Cleary and O'Neill, 2005], we can reduce our decision to a choice between a population with *infeasible solutions* or *feasible-only solutions*. Irrespective of the approach, either case demands, that the population be governed in terms of feasibility; where our key concern, is the wish to guarantee that the result of search is at least a feasible-solution (if not close-to, or on the optimum).

With the former approach, an algorithm in the role of a *constraint-violation handler* is required; this, to ensure emerging *in*feasibility is controlled. With the latter (feasible-only) approach, a heuristic-encoding of the problem-specific domain knowledge: *a feasibility-algorithm*, narrows the search space to F ($F \subset S = \{0, 1\}^n$) the feasible-subset of the space of all possible solutions [Gottlieb, 1999b]. In the domain of EA approaches to knapsack problems, this narrowing of the search-space explored is commonly

[6]When discussing issues of feasibility, it is more common to refer to the chromosomes in terms of solutions [Michalewicz, 1996, Chu and Beasley, 1998, Gottlieb, 1999b, Bruhn and Geyer-Schulz, 2002]

referred to as *heuristic-bias*; directly transferring to the more general EA definition of a *genotype-phenotype* mapping.

The following presents a review discussing the issues involved in handling infeasible solutions. Subsequently, the role of heuristics in the maintenance of a feasible-only population, and what is meant by a hybrid-approach is discussed in a section detailing heuristic-bias; before the feasible-only approach (of most relevance to this thesis) is itself described.

3.5.1 Infeasible Solutions

With a direct-encoding or permutation-based representation (Section 3.3.1), the possibility may arise that a candidate-solution represents a set-of-items which violate a capacity constraint of the problem. The issue arises, as to how to evaluate the "goodness" of such individuals.

As argued by Richardson et al. [Richardson et al., 1989], it can be viewed that such individuals contain good genetic-material - which is of benefit to the evolutionary-search - and thus, they should be allowed access to the reproduction of new candidate-solutions. Furthermore, both Michalewicz and Richardson [Michalewicz, 1996, Richardson et al., 1989] present evidence to suggest that the exclusion of this genetic-material (omitting infeasibles from search), can cause premature-convergence in highly-restrictive problems.

In any case, when choosing to allow the emergence of infeasible-solutions in the EA - care must be taken when attributing fitness to these individuals: we wish to ensure that the population doesn't become over-run with infeasibility (to the point that no feasible-solution remains). The question arises then, as to how to police the method by which we express fitness to these individuals. A *constraint-violation handler* is required; predominately, the use of either penalty methods or selection-preferences are adopted.

Penalty Methods Penalty methods seek to reduce the fitness of candidate-solutions that violate constraints. For this purpose, the objective function can be augmented to include a penalty function such that evaluation of candidate x is determined by:

$$eval(x) \;=\; \sum_{j=1}^{n} p_j x_j \;-\; pen(x) \qquad (3.2)$$

where *pen(x)* is zero for all feasible solution candidates. As pointed out in [Michalewicz, 1996], penalties may be to the "death" (worst possible fitness); or alternatively, graded to a degree which varies according to the number of violations which occur: as a consequence this method of penalty-allocation is commonly refereed to as a *graded-penalty* method. In any case, it can be seen that penalty methods serve to *handle* solutions with constraint-violations (hence, the term *constraint-violation handler*) by reducing their fitness, and thus their chance of contribution to the next-generation's offspring; as such, their use is tightly-bound to a fitness-proportionate selection mechanism. The difficulties in calibrating such graded-penalty terms are discussed in [Goldberg, 1989, Richardson et al., 1989], where they point out that too severe a penalty may result in premature convergence to "super-individuals". On the contrary, a penalty which is too liberal, may result in a final solution which is infeasible.

Selection-Preferences Selection-preferences are presented by Hinterding and Michalewicz [Hinterding and Michalewicz, 1998, Hinterding, 1999], and look at using heuristic-rules which, when selecting a parent for reproduction - have a preference, or "prefer" the selection of one individual over another. Though not bound to any particular EA implementation, selection-preferences lend themselves more easily to work with some style of tournament-selection (this is most commonly the case). In the role of constraint-violation handler, selection-preferences work, by choosing between classes of infeasible and feasible candidate-solutions based on the following heuristic selection rules [Hinterding, 1999]:

- feasible solutions are always better than infeasible ones
- feasible solutions are compared by fitness
- infeasible solutions are compared via the number of violated constraints.

Preferences ensure that the end-solution is feasible; and as seen with the third point, minimise the probability of infeasible candidate-solutions resulting in offspring that violate constraints. Extending the use of preferences, Hinterding and Michalewicz adapt the idea of *parent matching* [Ronald, 1995], and match infeasible parents - with mates, that satisfy the constraints that they themselves, do not [Hinterding and Michalewicz, 1998]. That is, given the first parent, a mate is chosen (*based on this first parent*); such that the probability of the offspring having constraint-satisfaction properties is increased: the second parent satisfies the constraints the first does not.

Discussion of Approaches with Infeasibility

A review of the literature allows us to conclude that, in general - penalty methods require a problem-specific analysis phase, such that the penalty can be tailored to the particular fitness-landscape of the problem at hand. Gottlieb [Gottlieb, 1999b] cites that they are "quite sensitive to the problem structure at hand", and gives reference to other works which resulted in implementing various biasing-techniques in order to ensure feasibility.

Olsen[Olsen, 1994], studied twelve different penalty methods over the single-constrained problems using both weak and harsh penalties; she reported that the result of search was often an infeasible solution. Khuri et. al

[Khuri et al., 1994] applied a graded-penalty term to a test-bed of nine standard 01MKP problems (multi-constrained); but similarly report, only moderate success with the more constrained problems - one such problem-instance remaining unsolved.

More conclusively, Michalewicz [Michalewicz, 1996] observed that, (for problems with restrictive knapsack-capacities) penalty functions do *not* produce feasible results: this is in direct support of Olsen's findings. In this study, however, he concludes that a wider range of penalty methods would need to be considered before a broad conclusion could be determined: i.e. it would be wrong to discard them as a poor approach. We note that for the 01MKP, Khuri et al.'s findings do show some merit for investigation[7].

The use of Selection-preferences as a constraint-violation handler, on-the-other-hand is not so common. Powell and Skolnick [Powell and Skolnick, 1993] are one of the earliest works to introduce this scheme. Hinterding and Michalewicz adapt this approach, implementing selection-preferences coupled with a novel parent-matching system [Hinterding and Michalewicz, 1998]. Applied over a set of linear and non-linear optimisation problems, results reported were competitive to the GENOCOP system (stated to be the best adapted EA approach at the time) [Michalewicz and Nazhiyath, 1995]. In a later work [Hinterding, 1999], he provides a direct comparison with penalty-methods, reporting a preference-based system to improve over a quadratic-penalty method; again, this is over a 100 item single-constrained problem.

Although much of the work from the literature discussing constraint-violation handlers - do so with respect to the single-constrained problem, our review of the literature leaves us with the impression (as with Hinterding

[7]It is worth noting that, all of the above outlined penalty methods utilised the same bitstring representation, working with problems of varying size and constraint-tightness.

[Hinterding, 1999]) that selection-preferences provide two main advantages over penalty methods: they provide an assured feasible end-solution, and they eradicate the need for problem-dependant tuning of a penalty function. We note again, that they are particularly suited to a method of tournament selection.

3.5.2 Heuristic Bias: Maintaining Feasibility

All feasibility-algorithms, essentially serve to *build* legal knapsack solutions, from the genetic-material specified on the chromosome. A heuristic-guided construction occurs; only producing a phenotypic trait, if doing so maintains all the constraints of the problem at hand. In this way, a heuristic encoding of the problems weight-constraints is required to act out the role of *constraints-checker* (to asynchronously indicate when a constraint-violation might occur). As such feasibility-algorithms have been termed *construction-heuristics* [Cotta and Troya, 1998], *constructive base heuristics* [Avci et al., 2003] and *decoding heuristics* [Raidl, 1999b]. (We conform to the use of Cotta's term; i.e. construction-heuristic).

The term *heuristic-bias* then, describes, the amount of heuristic-encoding maintained in the feasibility-algorithms construction algorithm. For example, a hybrid-approach (incorporating knowledge of the problems objective-information, as well as the weight-constraints) is said to further narrow the search-space to the *boundary* B of the feasible-region ($B \subset F \subset S = \{0, 1\}^n$) [Gottlieb, 1999a, Gottlieb, 1999b]; that is, heuristic-bias maintains that all genotypes map into a smaller phenotypic space.

In the context of knapsack problems, there are a number of standard approaches to construction-heuristics[8]; each varying in the amount of heuristic-

[8]Martello and Toth's book is the canonical point-of-reference for an in-depth description

bias imposed. Here, we outline two of the more common heuristic-approaches, and follow with an example of a greedy heuristic (*or hybrid-algorithm*) in the *first-fit descending*.

- **Next Fit:** The most simple heuristic, which as the name suggests; scans the chromosome from left to right and tries to add the next item represented. Items are added until the constraints-checker identifies a violation;
- **First Fit:** First fit is an adaption of the next fit whereby a violating item is skipped over; and what remains of the chromosome is scanned for any other possible addition. In effect, it's next-fit but where violating items are *skipped*;
- **First-Fit Descending:** A greedy-heuristic or local-optimisation technique. A list of the set of possible items is maintained; and sorted in terms of pseudo-utility ratio (*profit-to-weight*). Subsequent to this sorting or ranking, items are added in the manner described by the next-fit heuristic.

Upon first inspection, it would appear that a greedy sort by pseudo-utility (in the latter heuristic) would result in the same sort each time. Recall however, a discussion of this assumption in the previous chapter (Section 2.3), which points out that for the 01MKP (as opposed to the single-constrained problem) - estimating the profit-density or pseudo-utility for an m knapsack problem, is in itself, an NP-hard task.

of heuristic-approaches [Martello and Toth, 1990].

3.5.3 Feasible-Only Solutions

We have seen that with the approach to feasibility, whereby infeasible solutions are allowed in the population of an EA - the search-space of all possible solutions $S = \{0, 1\}^n$, is directly explored; (with an algorithm in the role of a constraint-violation handler helping to maintain feasibility of the end-solution). Alternatively, choosing to ensure all chromosomes represent feasible-only solutions, implies the use of a feasibility-algorithm to constrain the search to, F the feasible subset of S.

Essentially, within the domain of EA approaches to constrained optimisation, three main classes of feasibility-algorithm exist: *repair*, *decoders*, and *encoders*. Upon closer inspection from the literature however [Gottlieb, 2000] [Bruhn and Geyer-Schulz, 2002] [Chu and Beasley, 1998] [Hinterding, 1994] [Hinterding, 1999] [Raidl and Gottlieb, 1999a] [Raidl and Gottlieb, 1999b] [Gottlieb, 1999b] [Michalewicz, 1996], it would seem that they are effectively synonymous, or defined in terms of the problem encoding - this is not the case. The clear distinguishing factor of a feasibility-algorithm is is choice of approach to evolutionary-learning.

With any feasibility-algorithm, heuristic-bias is enforced to ensure all chromosomes *decode* to a feasible solution; thereafter, the feasibility algorithm either follows a Lamarkian model of evolutionary-learning (in which a genetic re-engineering takes place); or a more Baldwinian interaction with the environment is followed[9]. The following brief account of each serves to clarify their distinctions: repair and encoders follow a Lamarkian-model, whereas decoders follow a Baldwinian approach to evolutionary-learning.

[9]See Mitchell's book for a definition of these learning approaches [Mitchell, 1997].

Repair With a traditional repair approach, chromosomes representing infeasible solutions are allowed in the population; and upon their detection are "repaired" or fixed. As a feasibility-algorithm - repair defines a construction-heuristic which determines the selective-interpretation of the genetic-material encoded. Thereafter, a constraints-checker (acting in the role of censorship), forbids genes from representing a constraint-violation. This much is clear.

At the point of discovering a constraint-violation then, the repair algorithm performs a genetic re-engineering of the original chromosome (changing the genetic-structure so as to subsequently represent a feasible-solution). This is essentially the defining characteristic of repair - a form of Lamarkian approach to evolutionary-learning whereby, the evolving population of genetic-material is directly effected, to reflect the changes recommended by the environment (for survival).

As hinted above, a repair algorithm can enter the realm of a hybrid-approach by incorporating a more detailed heuristic encoding; this can afford a locally-optimised order-of-visit (or sorting) of items - resulting that genes of lower phenotypic benefit get re-engineered as opposed to good-ones. The more traditional repair (relying on the natural order of the permutation encoded) is sometimes refered to as *random-repair* with the local-optimisation approach describing *hybrid-repair* or *greedy-repair.*

Although repair algorithms, are predominantly used to maintain feasibility of the *entire* evolving population; infeasible solutions *can* however, be allowed in the population to a certain degree. As Michalewicz describes [Michalewicz, 1996], a *replacement-rate* can be supported. That is, only a certain percentage of detected infeasible-individuals need actually be repaired and replaced (*placed back in the population, in their corrected genetic-state.*). Michalewicz points out that, although Orvosh deems this percentage to be a performance related metric [Orvosh and Davis, 1993]; stating that a 5% replacement is optimum for best-performance: Michalewicz presents contradictory evidence, suggesting that the percentage replacement has no effect on performance. It may be presumed that this is the reason why the most prominent repair-based approaches, use a 100% replacement rate. In other words, repair is used as a feasibility algorithm - maintaining a 100% feasible-only population.

Encoders Encoders can be seen as a specialisation of repair - whereby an encoder differs, in that, controlled initialisation and adapted genetic-operators exist to ensure that a chromosome may never represent an infeasible solution. The defining difference between an encoder and a repair algorithm, is that with an encoder, infeasible candidate-solutions *never* emerge in the population.

A second point of difference, sees that an encoder must have it's evolutionary operators fitted with a construction-heuristic; to ensure the generation of feasible-only solutions. The encoding and operators can be sufficiently matched, such that infeasible solutions never occur; e.g. this is typically the case with grammar-based EA approaches. In this way, an encoder can never be used in conjunction with a constraint-violation handler; this, as a replacement-rate is not an option. Feasibility is built into all facets of the evolutionary process.

Decoders The family of algorithms referred to as *decoders*, are distinguished by their Baldwinian approach to evolutionary-learning: interaction with the environment can never effect a change to the genetic-material undergoing evolution. In this way, (as a feasibility-algorithm), the chromosome *decodes* to an expressed (legal) phenotype; that is, it maps into F the feasible-subset of the solution space. This phenotype is legal (feasible) and can be evaluated to yeild a fitness. It is here, that the distinguishing factor between decoder and repair is exemplified: with a decoder - the (yielded) fitness is expressed back onto the untouched chromosome (no genetic re-engineeering of the original genetic material takes place).

In reviewing the literature, it can be seen that decoder approaches for the MKP are often referred to as being *order-based decoders*[10] [Hinterding, 1994, Hinterding, 1999, Raidl, 1998]. This being the case, it simply refers to the use of a decoder, as a permutation-based EA (See Section 3.3.1). Here, the genotype represents a point in the search-space S (*i.e. a permutation*), whereby the decoder's construction-heuristic maps this, to a point in the feasible solution-space F. In this case, a decoder operates exactly as a repair-algorithm; the only difference being - that no genetic re-engineering takes place.

It is an interesting point to note - that in the field of knapsack problems the best-performing feasibility algorithms to-date have proven to take the approach of hybrid-repair algorithm [Gottlieb, 1999b, Gottlieb, 1999a, Raidl, 1998, Chu and Beasley, 1998]. Contrary to a study by Whitley et al. [Whitley et al., 1994] (where a Baldwinian form of learning was shown to sometimes outperform a Lamarkian approach) this would seem to suggest - that the domain of the 01MKP is best suited to the Lamarkian approach to evolutionary-learning.

[10]The term *selection-based* has also been used by Hinterding [Hinterding, 1994]. As such we advise against the use of either - as the verbs *order* and *select* are commonly used to describe the operation of an algorithm; in the end, it only leads to confusion.

3.6 Indirect Decoders

As the work of this thesis essentially documents research into a novel genotype-phenotype mapping process (working over an indirect encoding) - this section details two such similar decoders which have seen substantial research in the literature [Raidl, 1999b, Cotta and Troya, 1998, Raidl and Gottlieb, 1999a, Raidl and Gottlieb, 1999b, Raidl and Gottlieb, 1999c]. As such, this section encompasses two core themes; *a*) to provide a brief introduction of these decoders, and *b*) to layout their operation with respect to the framework developed in the work of Gottlieb and Raidl [Raidl and Gottlieb, 1999a, Raidl and Gottlieb, 1999b, Raidl and Gottlieb, 1999c, Gottlieb, 2000]. It is felt that the relevance of the later consideration is important, as in recent years theirs is the most significant contribution within the domain of applying EA's to MKP's. In their work, they categorise decoder-based EA approaches as *permutation-based*, *ordinal-representation based*, and as we will interpret it; *problem-space search based*[11]. As permutation-based decoders, have already been discussed - the following details the remaining; focusing only the necessary detail for their understanding.

3.6.1 Ordinal-Representation Based Decoder

The *ordinal-representation* described in [Michalewicz, 1996], and empirically tested in [Raidl and Gottlieb, 1999a] [Raidl and Gottlieb, 1999b] and [Raidl and Gottlieb, 1999c] is an example of the numeric-integer representation of fixed-length, which does *not* classify as a *permutation-based* approach. Although, originally considered for the traveling salesperson problem (TSP)

[11]For the latter category, we choose to group their use of both the *surrogate-relaxation* and *Lagrangean-relaxation* into the more common term of *problem-space search* (*See Section 3.6.2*)

[Grefenstette et al., 1985], it is easily transfered to the MKP. In the ordinal representation, each chromosome is a vector $\vec{v}$, of n integers. At initialisation, a deterministic method determines the value of each gene - whereby the i^{th} element of the vector (each gene of a chromosome) has a value in the range $[1,\ (n-i)+1]$. At it's core, the decoder maintains an ordered-list of the n possible items as an internal-representation. This list supports removal of an item at any index, such that the order of all remaining items is preserved - subsequent to removal, the size of the list is decremented by 1. As this suggests, the list reaches $size = 1$ only when the last item remains. The following example adapted from Michalewicz's explanation serves to explain the internal operation of this method [Michalewicz, 1996]:

Consider $L = (1, 2, 3, 4, 5, 6)$, the internal representation (an ordered-list), and vector $\vec{v} =< 4, 3, 4, 1, 1 >$. which is built from the $[1, (n-i)+1]$ principle. It can be seen that any vector of items $\vec{v}$, will always produce a list of numbers - which (when read sequentially from left to right) result in a valid index for the removal of an item. In the case of this example, the viable candidate-solution $(4, 3, 6, 1, 2, 5)$ results.

As Gottlieb and Raidl point out [Raidl and Gottlieb, 1999a], this is effectively similar to the operation of a permutation-based decoder. That is, the fixed-length vector (chromosome) operates over the internal-representation L to result in a permutation of the set-of-items; thereafter the use of a *constraints-checker* (coupled with a construction-heuristic) can be employed to ensure feasibility. It's difference to a permutation-based representation - and a cited advantage at the time of it's inception - was the ability to maintain classical genetic operators. (As illustrated in [Michalewicz, 1996], classical single-point crossover conducted over the fixed-length vectors (*or chromosomes*), can be applied to any two parent vectors producing an offspring which respects the $\{1 \ldots n - i + 1\}$ principle. Also, a uniformly distributed random value within this same range can be used to mutate a gene in a similar way to the classical bit-mutation.)

3.6.2 Problem-Space Search (Weight Coding)

Problem-space search (PSS) was first introduced by Storer et al. in the context of applying genetic-algorithms to scheduling problems [Storer et al., 1992, Storer et al., 1995]. These works found the approach to be effective for a variety of scheduling problems. The work of Avci et al. provides a good explanation of PSS, applying it to the single-machine weighted tardiness problem [Avci et al., 2003].

In more recent years, this approach has seen successful application to the MKP. Cotta in [Cotta and Troya, 1998] utilised problem-space search in both a standard GA and a distributed version using an island model with increasing success. More prominently Raidl's *weight-coded* GAs provided highly competitive results to those published in Chu and Beasley's journal paper (these have become somewhat of a bench-mark for the measurement of 01MKP performance) [Chu and Beasley, 1998]. Similarly, and building from this work - the surrogate-relaxation based, and Lagrangean-relaxation based decoders of [Raidl and Gottlieb, 1999a] [Raidl and Gottlieb, 1999b] and [Raidl and Gottlieb, 1999c] describe an algorithm working over problem-space search.

In applying an EA to an optimisation-problem, what is referred to as a *fitness-landscape* can be used to describe the search-space[12] (See Section 1.2). Defining the shape of the fitness-landscape, is the objective-function value of

[12]We refer the reader to Poli's book for information on search-spaces and fitness-landscapes [Langdon and Poli, 2002].

each feasible set-of-items (each point in the search-space). These values define the peaks of the landscape. An evolutionary search attempts to search this fitness-landscape in order to locate the solution which maps to the highest peak. As pointed out in the introductory section on constrained-optimisation (*See Sect.1.2*), the constraints inherent in a constrained-optimisation problem; identify cutting-planes over the fitness-landscape. These cutting-planes are unique to a particular problem and never change; in effect, they exclude certain peaks from the area of feasible search $F \subset S \in \{0,1\}^n$. Problem-space search, works on the realisation that the optimal solution for a constrained problem, lies on the boundary of this feasible region (the boundary defined by the intersection of these cutting-planes). Owing to fitness-related selection pressure inherent in a standard GA, evolutionary search will often result in the convergence of the population to a particular neighbourhood (within the bounds of a peak). Often the search-algorithm cannot escape a sub-optimal peak - and therefore never finds the optimum solution.

The thinking behind PSS then, is to alter the fitness landscapes until the intersection of the cutting-planes (i.e. the optimum point of search) resides within one of the larger peaks of the landscape. The hope is that the hill-climbing algorithm, will now have an easy path to the optimum solution.

To paraphrase Cotta in [Cotta and Troya, 1998], in a caption illustrating the method of problem-space search, "The goal is to find a fitness landscape that optimally matches the requirements of a given heuristic". That is, we are trying to tailor the fitness landscape, such that the largest peak aligns with the intersection of the cutting-planes (at the optimal point in search). In this way therefore, allowing the hill-climbing algorithm to soar directly to the optimal intersection point.

3.7 Discussion of Approaches to Feasibility

From a review of the literature; the use of repair and decoder algorithms predominate over encoder approaches; for which Hinterding's description of a "*selection-based*" genetic algorithm in [Hinterding, 1994] is one of the few empirical examples. Michalewicz [Michalewicz, 1996], in a study comparing preferences, decoder, and random repair; concluded that a greedy repair method was found to outperform all others over a set of, nine single-constrained problem instances of increasing difficulty.

Subsequent to this study, the findings of Michalewicz are supported in later works utilising similar hybrid-repair approaches. For example, Chu and Beasley, in their journal paper present one of the single most outstanding works in the field of evolutionary approaches to knapsack problems [Chu and Beasley, 1998]. We attribute their work as the first to comprehensively attempt a brute-force empirical analysis of an EA approach to the 01MKP. The successful application of their hybrid-repair algorithm over 325 distinct problem instances was, and is to this day; a formidable bench-mark for comparison of any new approach to these problems. In excess of this work, they provide comparative results for the same 325 test cases as generated by the best exact approximation method of the time (V4.0 of the MIP solver CPLEX [CPLEX, 2005]). Results demonstrate that their hybrid-GA approach with problem-specific heuristic, outperforms CPLEX's exact approximation algorithm, providing improved final results and reduced computational effort. Following from this work, a comparison is performed with several heuristic methods from the literature, and results presented show the hybrid-GA to markedly outperform those compared. This work appears to be the first to utilise either the surrogate or Lagrangean heuristics (See Section 2.5) to solve knapsack problems in an EA context.

Consequently, it appears that Raidl chooses to examine the same heuristic methods in his "improved genetic-algorithm" (entitled IGA) [Raidl, 1998]. Together with Gottlieb in their studies of decoder-based EA's [Raidl, 1998] [Raidl and Gottlieb, 1999a] [Raidl and Gottlieb, 1999b] [Raidl, 1999b] [Raidl and Gottlieb, 1999c], they build on the work of Chu and Beasley's GA (entitled CHUGA); again using the surrogate and Lagrangean relaxations as part of their experimental comparison. The IGA improves upon the CHUGA which requires an average of 33 times the number of evaluations to that of the improved GA (in order to reach an optimal-solution of a high-quality). A secondary comparison shows Raidl's own IGA, and that of Chu and Beasley's CHUGA, to both significantly outperform a permutation-based decoder approach of Hinterding [Hinterding, 1994] (entitled OBGA). The difference with the order-based (OB)GA, seeing that it doesn't use local-optimisation. To provide a more fair-comparison they remove local-optimisation and the specialised initialisation of the IGA and CHUGA; and still report an improved performance over OBGA (but to a lessor extent). In this way, they can *a*) report improvement over the decoder approach, and *b*) confirm the benefit of special initialisation and hybrid-repair. Finally, IGA's improvement over CHUGA is examined and it is suggested that this improvement is as a consequence of the non-deterministic (*stochastic*) implementation in the IGA's local-optimisation algorithm. In contrast CHUGA uses deterministic methods. To test this theory, the CHUGA is re-implemented with the special-initialisation process of IGA (the difference now, narrowed to the local-optimisation technique) which results in faster convergence to near-optimal solutions, but at the cost of reduced fitness of the resultant best-found solution. This leads to the ability to conclude upon the superiority of having a non-deterministic local-optimisation process. A further point of analysis, suggests the reason for this is the lack of preservation of *diverse* high-quality solutions by the deterministic method, which results in convergence to highly fit individuals (See Chapter 7).

In [Gottlieb, 1999a], and stemming from the success of Raidl's approach, Gottlieb presents a study into the IGA's pre-optimised initialisation strategy and the effect of non-determinism in hybrid local-optimisation algorithms. His experiments show that pre-optimisation significantly improves all results, and that too-much non-determinism can degrade performance.

Summarising the most successful EA approaches to knapsack problems from the literature then, and as documented in Gottlieb's comparative study of EAs for the MKP [Gottlieb, 2000] - the IGA of Raidl provides the best performance, outperforming the hybrid-repair approach of Chu and Beasley (CHUGA) (upon which Raidl's work builds). The CHUGA, narrowly outperforms the problem-space search approaches of Raidl [Raidl, 1999b] with his best permutation-based EA being inferior to these. The permutation-based decoder approaches do show, however, an improvement over the previously tested ordinal-representation [Raidl and Gottlieb, 1999a], and random-key encoding approach of Hinterding [Hinterding, 1999][13].

3.8 Summary and Conclusion

In this chapter, an attempt has been made to disambiguate overlapping classifications from EA approaches of the literature. From our review of the literature, we identify that *problem-encoding* (and implicitly, the search-space explored) is the most important feature of concern when categorisation an EA

[13]Gottlieb's later work [Gottlieb, 1999b, Gottlieb, 1999a], examines the optimal initialisation and local search strategies for both the CHUGA and IGA implementations; we refer the reader to this work for details of same.

approach to constrained-optimisation. Gottlieb's work is cited as the canonical point-of-reference, for a description of this [Gottlieb, 1999b]). Following from this, we present a (sub-division-of-labour style) reduction, identifying two core-considerations of concern to the designer of an EA for knapsack problems: the *choice of approach to representation* and *choice of approach to feasibility*. In the former, we identify a definition for permutation-based EAs, before viewing the latter choice of approach to feasibility as a binary-decision: to *allow* or *omit* chromosomes representing infeasible-solutions from the population of an EA.

In choosing to allow infeasible-solutions, we propose the need for a *constraint-violation handler*, and detail a brief review of the works from the literature pertaining to these. Penalty-methods and selection-preferences are discussed, with the need for problem-specific tuning of penalties being seen as a disadvantage over the latter.

In choosing to omit infeasible-solutions (aiming to ensure all individuals in the population represent feasible-solutions), *feasibility-algorithms* are described; which serve to channel the arbitrary genotypic search-space $S = \{0,1\}^n$ into the feasible-subset F, there-of. The term *heuristic-bias* was seen to describe this narrowing of the search-space to a smaller solution-space; and a correlation made to it's more common description as a *genotype-phenotype* mapping within the GP domain. As such, heuristic-methods are briefly introduced. From the literature, feasibility-algorithms of *repair*, *encoders* and *decoders* were identified. A novel method of distinguishing between repair and decoder was proposed in the approach to evolutionary-learning employed. Repair and encoders are differentiated from decoders by a Lamarkian, as opposed to a Baldwinian form of learning.

A focus on prominent indirect-decoders presented two specific forms of decoder, before a discussion reviewing the research of EA approaches to knapsack problems identified the best-performing approaches relative to the worst. Fixed-length hybrid-repair strategies were seen to be the best.

Chapter 4

GE, Attribute Grammars and Knapsack Problems

4.1 Introduction

The work of this thesis thus far, has served to provide us with - a perspective showing the expressive capabilities of Grammatical Evolution (GE); an introduction to knapsack problems as context-sensitive problems; and a survey of the literature, focusing on EA approaches to knapsack problems. It is now time to realise the aims of the approach taken in this thesis: to extend the generative power of GE by embedding an attribute grammar in it's mapping process. Core to an understanding of the experimental approach described here, is the need to be familiar with the terminology of grammars (*Section 1.4.1*); and the knapsack problems approached (*Section 2.4*).

The introductory chapter introduced the GE mapping process, and it's grammatical background; here we will analyze the limits imposed by it's *context-free* specification of a grammar. A *context-sensitive* language will be presented, and the limitations of a context-free grammar (CFG) exposed.

The language of knapsack problems (particularly 01MKP's) will be shown to require a context-sensitive derivation. The options available to fulfill this requirement, will be briefly discussed before *attribute grammars*, will be proposed as a viable solution - furthering the expressive capabilities of the standard GE. In line with their importance, this chapter will be subdivided into two parts: the first outlining an introduction to attribute grammars - therein contained, a section describing why, and when, they are needed; and the second discussing the experimental approach in retrospect of their study. The requirement for use of an attribute grammar will be proposed in the context of the *language* of 01 knapsack problems (*See Section 1.4.1*). The attribute grammar workings and internals will subsequently be explained in the context of CFG's. As will be seen, attribute grammars can be used to drive a context-sensitive derivation over a context-free grammar (CFG); furthermore, experiments are carried out which subtly introduce two grades of context-sensitivity by varying degrees of attribute grammar, respectively.

Attribute grammars will be introduced purely in their role within our extension of the GE mapping process. In a discussion on the experimental approach, liaisons will be made with our study of evolutionary approaches from Chapter 3; and subsequently the experimental systems will be outlined in terms of this previous work. By no means is our intent to provide an in-depth definition of the formal semantics of attribute grammars. The material provided here is introductory, but sufficient to understand the workings of our approach. For further detail, we refer the reader primarily to Knuth's seminal paper [Knuth, 1968], and the text of Slonneger and Kurtz [Slonneger and Kurtz, 1995][1]; who provide a very well explained account of both the workings and formal semantics of attribute grammars.

[1]Also available as an online text at http://www.cs.uiowa.edu/ slonnegr/plf/Book/

4.2 Language, Limitations and Attribute Grammars

Consider the following example BNF specification of a CFG, G:

```
<S> ::= <A>       (0)
<A> ::= a <B>     (0)
      | a b c     (1)
<B> ::= b <B>     (0)
      | c         (1)
```

The above grammar specifies three production rules, where index numbers have been added to allow reference to particular rules. In referring to a single rule, the left-hand side (L.H.S) non-terminal, coupled with the index number in subscript, will be used. For example, *rule* A_1, will refer to the rule $A\ ::=\ a\ b\ c$. Also note, that the use of capitalisation will denote non-terminals when referring to such rules. Finally, for clarity of explanation, we deviate from the BNF and adopt the notation of Knuth [Knuth, 1968] when referring to production-rules within the body of text: as such, the above rule will be specified as $A \rightarrow a\ b\ c$. This decision has been made - particularly to provide clarity in explanation of the attribute-grammars described in this chapter.

As the terminology used by Aho *et. al* [Aho et al., 1986], in describing the formal syntax of programming languages, a grammar can be seen as a *phrase-structure generative grammar*; whereby, rules of the grammar, outline the structure, by which syntactically correct sentences of the language can be derived. At closer inspection, the grammar G specifies, or defines *a language*. This language, written $L(G)$, determines the set of legal (*or syntactically correct*) sentences, which can be generated by application of the grammar's rules. For example, the above grammar, and also within the illustration (Figure 4.1), defines the language $L(G) = a\ b^*\ c$: the set of strings starting with the terminal-symbol 'a' - ending in the terminal-symbol c; and having zero-or-more of the terminal-symbol 'b' in between (Note: the * symbol denotes, *zero-or-more*) [Aho et al., 1986].

The non-terminal symbols 'A' and 'B' define the phrase-structure of the language. They define *A-phrases* and *B-phrases*, from which the language is contained. These would be similar to constructs such as *noun-phrases* in spoken language, or for example, a *boolean-expression phrase* from the abstract syntax of a programming language. In terms of the example grammar, a syntactically valid *A-phrase* contains an 'a' followed by a *B-phrase*. A recursive definition of the *B-phrase* thereafter, defines the previously stated language of the grammar. In this way, the structure of the syntax of an entire language can be defined in a concise and effective notation.

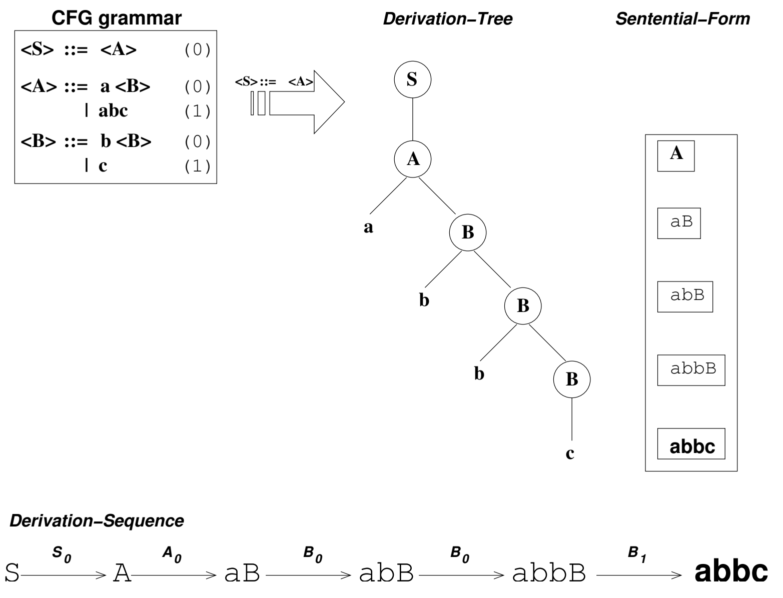

Figure 4.1: A Sample CFG derivation and the relevant terminology.

Furthering the explanation, consider the example depicted in Figure 4.1, which provides an illustration of these concepts. As stated, rules of the grammar are referred to as *production-rules*, and as such $A \rightarrow aB$, can be read as, "A" *produces* "aB". Similarly it can be said that "aB" is *derived* from A. In applying productions then, the goal is to *derive* a sentence of the language. As such, a completed set of productions - yielding a sentence of the language, is said to be a *derivation.* Each application of a production rule, can be seen as a *derivation-step*, whereby at any such derivation-step the applied production is said to yield a derivation in the *sentential-form.* A completed derivation results in a sentential form consisting solely of terminal symbols - i.e. a sentence of the language. Figure 4.1 also depicts how a derivation can be represented as a tree structure, with non-terminal symbols defining the nodes of the tree, and terminal symbols defining the frontier of the tree (*i.e. the leaf nodes*). As such it can be seen that an arbitrary derivation over the grammar G can also be described in a linear form as:

$$S \rightarrow \underline{A} \rightarrow \underline{aB} \rightarrow a\underline{bB} \rightarrow ab\underline{bB} \rightarrow abb\underline{c}$$

with each derivation-step leading to a derivation in the sentential-form; and underlines showing the replacement for the previous step's, left-most non-terminal. There exists however, a limit to the generative power of the CFG. As the name suggests context-free grammars cannot express a language in which; legal phrases, depend on the context in which they are applied. For example, we cannot define a CFG, G - such that $L(G) = a^n\ b^n\ c^n$. That is, a grammar describing the language of strings having an equal number of a's, b's and c's - namely the set $\{abc, aabbcc, \ldots, aaaabbbbcccc\ldots\}$ cannot be described by a CFG. As Slonneger points out [Slonneger and Kurtz, 1995], such a language is context-sensitive or *type-1* language according to Chomsky's hierarchy, and requires knowledge of the *context* at the point of each derivation-step. That is, we need to know how many a's have been previously derived, in order to derive 'b' and 'c' terminals. In order to get a feel for the problem, let us take this previously described context-sensitive language, and attempt to describe it with a CFG:

```
<S> ::= <RepeatLetter>                               (0)
<RepeatLetter> ::= <Apart> <Bpart> <CPart>           (0)
<Apart> ::= a                                        (0)
        |   a <Apart>                                (1)
<Bpart> ::= b                                        (0)
        |   b <Bpart>                                (1)
<CPart> ::= c                                        (0)
        | c <Cpart>                                  (1)
```

The CFG attempts to capture the nature of the language by specifying three recursively defined production rules; each allowing the multiple generation of the 'a', 'b' and 'c' terminals. However, it can be seen that at the point of a given derivation-step, for a terminal-producing production, we require knowledge of the current state of the derivation-tree. For example, consider a derivation:

$$S \rightarrow \underline{RepeatLetter} \rightarrow \underline{Apart\ Bpart\ Cpart} \rightarrow \underline{aApart}\ Bpart\ Cpart$$
$$\ldots \rightarrow a\underline{a}\ Bpart\ Cpart \rightarrow ??.$$

If at this point we wish to generate a legal *Bpart* phrase - we must know the structure of the afore-derived *Apart* phrase. For example, a derivation of $Bpart \rightarrow b$ would break the syntax of the language. The generative power of the CFG needs to be extended, and as such we have two choices: *a)* define a context-sensitive grammar, or *b)* adapt the CFG to overcome the context-sensitive nature of the problem solutions.

Now, designing a context-sensitive grammar (CSG) is a complex task; and the resulting grammar is tightly-bound to the specific context-sensitive language described. As such, for this approach - we fear both a loss of robustness and a dependency on expert-driven domain-knowledge (seen already, as the pit-fall of penalty-methods in Section 3.5.1). Additionally, it can be almost impossible to define a parser for some CSGs. As a consequence, the choice is narrowed to the adaption of the CFG. Although both Wong and Leung and Keijzer adopt the *definite clause grammars* approach of prolog to achieve this CFG extension [Wong and S., 1997, Keijzer, 2002], we choose the use of an attribute grammar in the same role. This choice stems both from the desire to explore their viability, and their ease of extension to both a CFG, and GE. With an attribute grammar, the context-free grammar essentially remains; but the features of the attribute grammar define a method to uphold a context-sensitive derivation. The following sections serve to provide an explanation as to how this can be achieved.

4.2.1 CFG Limitations for Knapsacks

Attribute grammars (AGs) were first introduced by Knuth [Knuth, 1968], as a method to extend CFGs by assigning attributes (or pieces of information), to the symbols in a grammar. In their traditional role, attribute grammars are used in programming-language compilers, to detect syntax errors, over context-sensitive features of the programming-language in question. In effect, an understanding of an attribute grammar is more easily attained, if we consider the derivation-trees produced by applying productions of a grammar. First, let us consider the following CFG which attempts to describe the language of 01 knapsack problems (*See Section2.4*).

$$
\begin{aligned}
&S \rightarrow K \\
&K \rightarrow I \\
&K \rightarrow IK \\
&I \rightarrow i_1 \\
&\quad\vdots \\
&I \rightarrow i_n
\end{aligned}
$$

Recalling that the 01 property of knapsack problems, requires that when proposing a candidate-solution - no item be represented twice (See Section 2.2). Our wish, is to present a grammar which can be used as a generative force to derive feasible-solutions which are legal: those which adhere to the 01 property. Thinking in a grammatical frame of mind, such feasible-solutions can be thought of, as those within the language of 01 knapsack solutions. This language should govern the generation of strings of terminal symbols, where no terminal symbol is repeated. We can attempt to generate a sentence in the language of 01 knapsack solutions, by the application of productions from the grammar, such that, only terminal symbols remain (aiming in the process to yield a string from the set of items $\{i_1, \ldots, i_n\}$). Consider however, the problem of generating such a string, given only the CFG definition above. Figure 4.2 illustrates the point at which a CFG fails to be able to uphold context-sensitive (*context-specific*) information.

Reflecting the figure, the following derivation-sequence points out the scenario where i_3 has been derived, and the next derivation-step should ensure that i_3 is not produced again:

$$S \rightarrow \underline{K} \rightarrow \underline{IK} \rightarrow \underline{i_3}K \rightarrow i_3\underline{IK} \rightarrow i_3\underline{?}K.$$

What this derivation-sequence provides is a *context*; that is, re-deriving an i_3 violates the semantics of the language of 01 knapsacks. A CFG has no

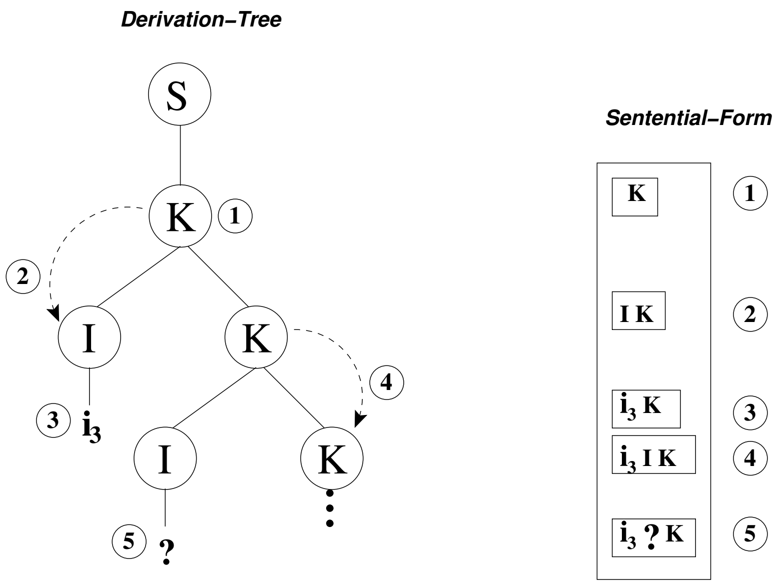

Figure 4.2: Diagram showing inability of a CFG to drive a context-sensitive derivation.

method of encoding this context-sensitive information and hence, like the language of the example grammar in Section 4.2 ($L(G) = a^n b^n c^n$), it cannot be captured by a CFG.

Similarly, the same problem arises when a potential production attempts to add an item, which would result in a weight-constraint of a knapsack being broken. Say, for example that we're dealing with a knapsack capacity of 10 *units*, and from the diagram above, i_3 has a weight of 7 *units*. It follows, that the item derived at step 5 above, can have a maximum weight of 3 *units* (*i.e.* $(7) + 3 \leq 10$). The following sections provide a definition of attribute grammars, with examples to demonstrate how an attribute grammar can be used as a plug-in component of GE - therein allowing us to drive the context-sensitive derivation of 01 knapsack problem solutions.

4.2.2 Attribute Grammar Fundamentals

Distinguishing attribute grammars from CFG's, attributes can be specified for the terminal and non-terminal symbols of the CFG. As stated by Slonneger and Kurtz [Slonneger and Kurtz, 1995], "A finite, possibly empty set of attributes is associated with each distinct symbol in the grammar". In essence, this translates to a representation, where, as well as the symbol of the grammar, nodes of the derivation-tree now represent attribute-data. These attributes are defined (given meaning) by functions associated with productions in the grammar. These shall be termed the *semantic-functions*. In effect, the specification of these semantic-functions, is just a formal specification of a routine; the routine being executed when carrying-out a production. Semantic-functions serve to calculate the values of attributes in the context of the current derivation-step. As depicted in Figure 4.3, attributes are thus pieces of data appended to nodes of the tree, and can be evaluated in one of two ways. In the first, the value of an attribute is determined by information passed down from parent nodes. That is, a child's attribute is evaluated based on information which is *inherited* down from parent nodes. The figure depicts a limit attribute, originating at the root node S, which is inherited down the tree as the derivation progresses.

With the second method of attribute evaluation, the value of an attribute is determined, by the value of the attributes of child nodes. That is, the evaluation of a parent attribute can be *synthesized* or made up of it's child's attribute values. As an example, two attributes are associated with the I non-terminal: *weight*, addresses the weight of the item; and *item* provides a string representation of the item for the growing solution. Originating values for these terminal-symbols, typically come from predefined, known values, encoded in a data-structure. These originating values, feed the attribute

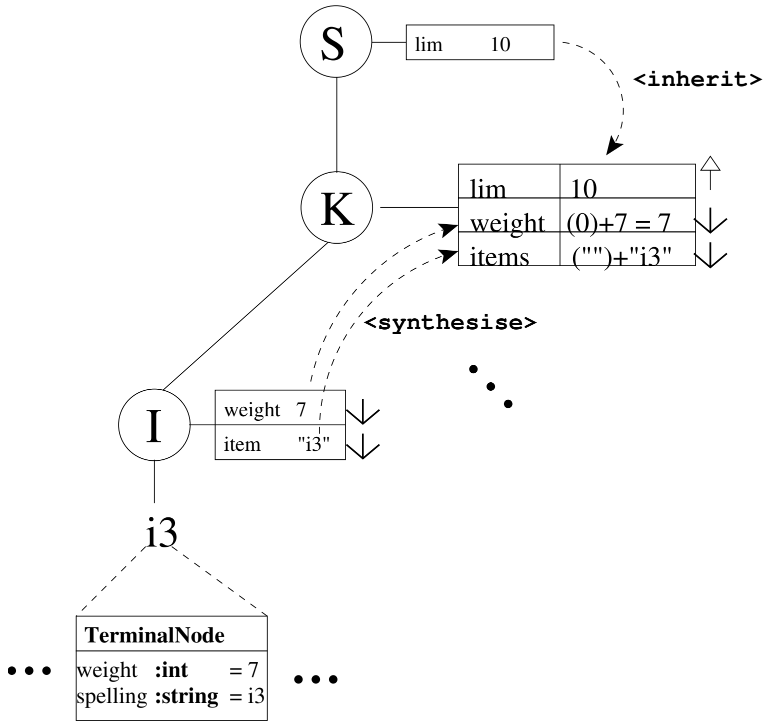

Figure 4.3: Diagram detailing the fundamental workings of node attribute-evaluation: inherited or synthesised attributes. Inherited attributes are demarcated with an up arrow, and synthesised with a down arrow.

evaluation of all other nodes within the derivation-tree.

In addition to the two main forms of attribute evaluation, attributes of a particular node *can* be evaluated in terms of the other attributes, of that same node. In the figure, for example, K's weight could be evaluated depending on the value of K's limit attribute.

Information however, must originate either from the root node S or leaf nodes of the tree; which generally provide constant values, from which, the value of all other nodes in the tree are synthesized or inherited. As an example, the diagram illustrates how a derivation of the terminal symbol i_3, may require access to a predefined data-structure to produce these *originating* data values. The start node's limit value could be attained in the same way.

In terms of understanding AGs then, it's best to think of the growth of the derivation-tree, through the application of productions. The grammar symbols become nodes of the tree; the root node of the tree is S (*the start symbol*), and it's children - the symbols of the applied production. A portion of a derivation-tree descended from a single non-terminal node yields a derivation in the sentential-form; and constitutes a derivation-step. The term *terminal-producing production* will be used to refer to a sentential-form which contains one-or-more terminal symbols.The fundamental operation of an AG then, is effectively the same as a CFG; except where attributes are evaluated at each derivation-step. The following two sections detail how an attribute grammar's *semantic-functions* succeed to carry out this task. An illustrated example of the attribute grammar workings are left to Section 4.2.4 (describing an attribute grammar which subsumes the functionality of the first, and presents a method for generating feasible-solutions for 01MKPs); comprehension of the grammar described therein, is deemed to be of more benefit to the remaining work of the thesis.

4.2.3 An Attribute Grammar for 01 Compliance

In the following attribute grammar specification, it can be seen that attributes can be used, to preserve 01 compliance; when deriving strings in the language of 01 knapsack solutions. This attribute grammar is identical to the earlier CFG, with regard to the syntax of the knapsacks it can generate. The difference here being, the inclusion of attributes associated with both terminal and non-terminal symbols, and their related *semantic-functions*. Following the notation of Knuth [Knuth, 1968], we use a subscript notation to differentiate between like-occurrences of the same non-terminals. This is necessary as each symbol (node) maintains it's own separate attribute set. Finally, as the formal semantics of attribute grammars dictate, each semantic-function is a mapping; which maps values of certain attributes of one-or-more symbols *into* the value of some other symbol [Knuth, 1968, Slonneger and Kurtz, 1995, Aho et al., 1986][2]. As no standard notation exists for conditional mappings, we choose to use a clear distinction by the keyword *condition* in bold face, followed by a colon.

$$
\begin{array}{lll}
S \rightarrow K & & \\
K \rightarrow I & & \mathit{items}(K) = \mathit{items}(K) + \mathit{item}(I) \\
K_1 \rightarrow I K_2 & & \mathit{items}(K_1) = \mathit{items}(K_1) + \mathit{item}(I) \\
 & & \mathit{items}(K_2) = \mathit{items}(K_1) \\
I \rightarrow i_1 & \textbf{Condition}: & if(\mathit{notinknapsack?}(i_1)) \\
 & & \mathit{item}(I) = \text{``}i_1\text{''} \\
\vdots & & \\
I \rightarrow i_n & \textbf{Condition}: & if(\mathit{notinknapsack?}(i_n)) \\
 & & \mathit{item}(I) = \text{``}i_n\text{''}
\end{array}
$$

As can be seen, the grammar is much the same as the CFG for knapsacks described earlier in section 4.2.2; the difference lying in the inclusion of *semantic-functions* listed to the right of the grammar's rule-set. Intuitively, one can see that these represent methods for evaluating attributes to their relevant values. Conditions govern the firing of the set of semantic-functions directly below them. The following describes the nature of the appended attributes:

items(K): A synthesized attribute that records all the items currently in the knapsack (i.e. the items which have been derived thus far).

[2]Mathematically, a mapping between sets is essentially a functional relation; and as such, can be governed by a conditional-expression. As the case may be, it is common to see attribute grammars with conditions listed amongst the semantic-functions.

item(I): A string representation, identifying which physical item the current non-terminal will derive. For example $item(I) = \text{“}\,i_1\text{”}$; where the I non-terminal derives or produces i_1 (for *item* 1) of the problem.

notInKnapsack?(i_n): A boolean condition, indicating whether the 01 property can be maintained by adding this item (i.e. given the current derivation - has this item been previously derived?). This is represented as a string comparison of *item(I)* over *items(K).*

Consider the above attribute grammar, when applied to the following derivation-sequence:

$$S \rightarrow \underline{K} \rightarrow \underline{IK} \rightarrow \underline{i_1}K \rightarrow i_1\underline{IK} \rightarrow i_1\underline{(i_\lambda \in \{i_2...i_n\})}K \rightarrow ...$$

At the point of mapping I, given the above context, it can be seen (from the attribute grammar definition), that it's *items(I)* attribute will only be evaluated to “i_λ” if the *notinknapsack?()* condition holds[3]. Following this the parent node will have it's *items(K_1)* updated to include “i_λ” which can from then on be passed down the tree by the inherited attribute of *items(K_2).* This in turn allows for the next *notinknapsack()* condition to prevent duplicate items being derived.

It must be noted, that at the point of a *notinknapsack?()* condition failing (or recognising the attempted derivation of a duplicate), we have a *syntax-error.* This will be discussed in the context of an attribute grammar mapping for GE in a proceeding section (*Section 4.2.5*). The next section follows to provide a deeper example; showing how we can include the evaluation of weight-constraints - at the point in a derivation where we derive a *terminal-producing production* (See Section 1.5).

[3]The CFG alone could only ensure $i \in \{i_1, i_2, \ldots, i_n\}$ is derived from I. An explanation of what happens when another i_1 is produced, is left for a GE mapping example in section 4.2.5

4.2.4 An Attribute Grammar for Constraints Checking

Further attributes can be added, in order to extend the context-sensitive information captured during a derivation. The following outlines these attributes and their related semantic-functions in a full AG specification for valid 01MKP solutions: those which maintain both 01 compliance; and constraint-violation information (*See Section 2.4*).

lim(S): A global attribute containing each of the m knapsacks' weight-constraints. This can be inherited (or passed-down) to all nodes.

lim(K): As lim(S); just used to inherit to each K_2 child node.

usage(K): A usage attribute, records the total weight of the the knapsack to date. That is, the weight of all items which have been derived at this point.

weight(K): A weight attribute, used as a variable to hold the weight of the item derived by the descendant I to this K.

weight(I): A synthesized attribute, made-up of the descendant item's physical weight.

weight(i_n): The physical weight of item i_n(*the weight of item i_n as defined by the problem instance*).

The corresponding attribute grammar is given below with an example showing how it's attributes are evaluated. At the point of deriving a left-hand side production, the corresponding right-hand side semantic-functions are evaluated/executed. Again, at the point of their satisfaction, conditions govern the firing of the set of semantic-functions directly *below* them. Such is the design of this AG, that conditions serve to test whether it is possible to evaluate our synthesised attributes (*synthesis*); this in turn allows us to evaluate our inherited attributes (*inheritance*) - passing their information down to the next level.

$$
\begin{array}{ll}
S \rightarrow K & lim(K) = lim(S) \\
K \rightarrow I & \textbf{Condition}: \quad if(usage(K) + weight(I) <= lim(K)) \\
 & weight(K) = weight(K) + weight(I) \\
 & items(K) = items(K) + item(I) \\
K_1 \rightarrow IK_2 & \textbf{Condition}: \quad if(usage(K_1) < lim(K_1)) \\
 & weight(K_1) = weight(K_1) + weight(I) \\
 & items(K_1) = items(K_1) + item(I) \\
 & usage(K_1) = usage(K_1) + weight(I) \\
 & lim(K_2) = lim(K_1) \\
 & usage(K_2) = usage(K_1) \\
 & items(K_2) = items(K_1) \\
I \rightarrow i_1 & \textbf{Condition}: \quad if(notinknapsack?(i_1)) \\
 & item(I) = \text{“}i_1\text{”} \\
 & weight(I) = weight(i_1) \\
\vdots & \\
I \rightarrow i_n & \textbf{Condition}: \quad if(notinknapsack?(i_n)) \\
 & item(I) = \text{“}i_n\text{”} \\
 & weight(I) = weight(i_n)
\end{array}
$$

For clarity of explanation, the following example will assume a single knapsack weight-constraint; however, in terms of the problem being solved, *lim(K)* is actually a list of constraint-bounds for *each* of the m knapsacks, but the more complicated problem can be extracted by altering the below conditions to have *lim(K)* as an array of constraint-bounds; as opposed to a single integer value. Figure 4.4, and the following derivation sequence, serve to

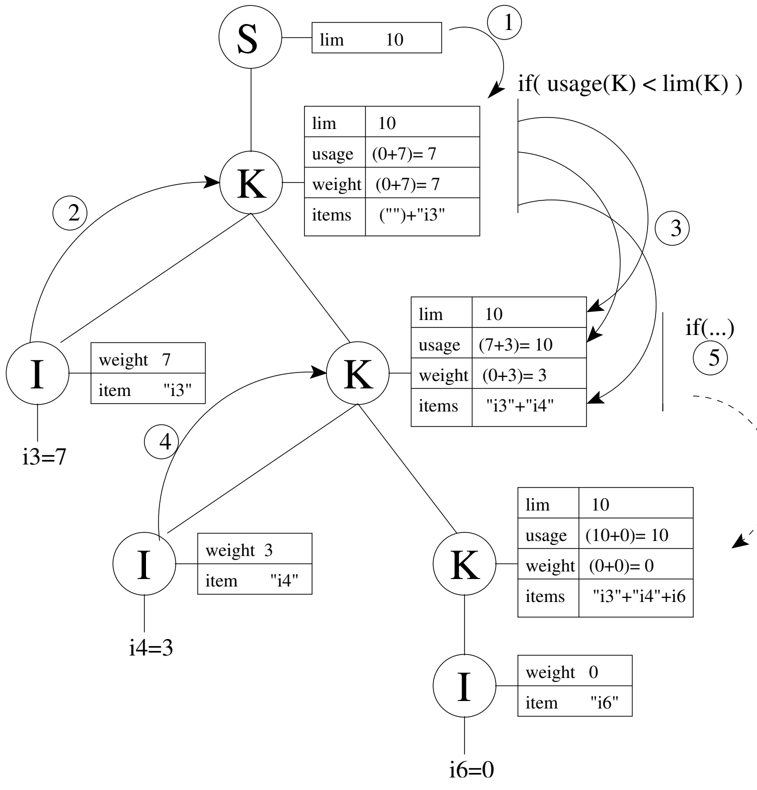

Figure 4.4: Diagram showing synthesized and inherited message-passing for evaluating attributes in the derivation-tree of an attribute grammar.

show the synthesized and inherited message passing involved in evaluating derivation-trees for the above attribute grammar.

$$S \rightarrow \underline{K} \rightarrow \underline{IK} \rightarrow \underline{i_1}K \rightarrow i_3\underline{IK} \rightarrow i_3\underline{i_4}K \rightarrow i_1 i_4\underline{I} \rightarrow i_1 i_4 \underline{i_6}$$

We can see, that initially the global limit is passed down to K by the first semantic-function. From the grammar, we can see that following this, the first three semantic-functions of K, for *weight, items,* and *usage* attributes respectively, involve evaluation by synthesis of the I node's attribute values (these, in their own evaluation, are a synthesis of the *originating* values). The tree must expand in depth to evaluate I, prior to it's K parent's evaluation.

A condition checks to see that we haven't violated a weight-constraint[4], and passing this the first three semantic-functions may execute. These evaluations are followed by the subsequent three semantic-functions for K; involving inheritance evaluations.

Similarly, the derivation continues; and the fully decorated derivation-tree results in a valid 01MKP solution. The next section examines details of our mapping approach; when things do not work out so well.

4.2.5 GE Mapping Example

Our approach then, has been to extend the mapping process of GE; such as to embed the above explained attribute grammars, into it's mapping algorithm. This section provides an example of how this effects the mapping process, and describes some of the important implementation-specific details, of our approach.

As was seen in the previous section (and that which is inherent in the language of 01 knapsack solutions), preventing the re-mapping a duplicate item, is controlled by conditions defined within the attribute grammar. Similarly, attempting to derive an item which causes a constraint-violation, must be prevented. We need to discuss what happens upon a condition failure. In failing a condition, our approach is to re-map the offending I non-terminal; using the next codon on the genome. Figure 4.5 shows how this results in the offending codon being set to an *intron.* (The term *intron*, stems from it's use in genetics where it is used to describe pieces of non-coding DNA.) So in effect - and by design - our wish is to provide a model, close to that which occurs in nature. In our model, at the point of constraint-violation or a 01

[4]For clarity, we assume that the *notInKnapsack(i_3)* condition has passed and its values have synthesized up the tree.

collision, the codon causing the error is skipped (becomes an intron) and the subsequent codon read. It is worth noting that this is not dissimilar to the approach taken in the GADS system [Paterson and Livesey, 1997] discussed in the introductory chapters review of the literature (*Section 1.4.2*). In terms of the previous chapter's discussion of *feasibility-algorithms* - it can be seen that we are adopting a Baldwinian approach to evolutionary-learning; (whereby interaction with the environment has no bearing on the genetic make-up of an organism) - as such, we define an attribute-grammar *decoder* for knapsack problems[5] (See Section 3.5.3) [Cleary and O'Neill, 2005].

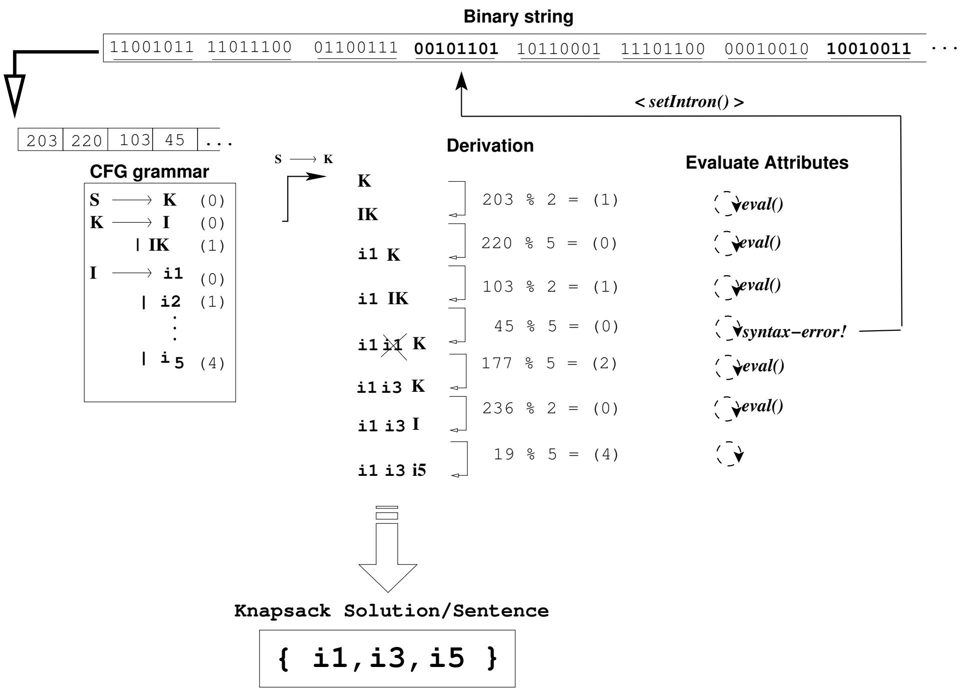

Figure 4.5: Diagram showing GE mapping process with a full attribute grammar encapsulating the 01MKP solution language.

[5]It is worth noting, that skipping codons which result in infeasibility introduces a bias to the search; this may or may not be of benefit to the algorithm; Chapter 5 attempts to further elaborate on the effect of this.

It can be seen from Figure 4.5, that the addition of attribute grammars to the mapping process, involves the evaluation of each productions semantic-functions (This has been represented by the processing symbol in the diagram). In this way, a controlled derivation can take place and always ensure a sentence of the (now, context-sensitive) attribute grammar's language is derived. The only occasion when this will not be true, is when we do not have enough genetic material in the genome to continue the process to a fully exhausted valid solution; in this case, the GE individual will be penalised by what Michalewicz terms a "death-penalty" (evaluated to the worst possible fitness) [Michalewicz, 1996].

4.3 Experimental Design and Initial Results

As pointed out in Chapter 3, this thesis sees an EA approach to knapsack problems as a twofold consideration: the choice of representation, and the choice of approach to feasibility. Considering that we have decided to use GE as the core evolutionary algorithm, the experimental design is focused on the latter consideration - approach to feasibility (an indirect variable-length binary representation as the GE standard, deciding the first). We choose to test the standard GE system against the two previously discussed attribute grammar extensions; each capable of controlling a context-sensitive derivation to the language of knapsack problems.

It has been seen that the standard GE mapping process is incapable of providing us with a feasible-only approach (infeasibles can occur). In terms of the considerations in Section 3.5 (discussing feasibility) - we can now view the CFG, as defining a heuristic-bias from the standard GE genospace $S = \{0, 1\}^{\alpha}$ into the language of the integer-valued MKP (the set of variable-length lists of numbers with repeats allowed)[6]. At this point - the heuristic-bias matches what Whigham [Whigham, 1996] describes as *language-bias*. That is, we have constrained the search to the language of integer-valued MKP solutions (a solution-space). Now, what the attribute grammar provides (and where it's power lies), is a method to further constrain the mapping from search to solution-space. In GE terminology, it is said to further increase the redundancy of the encoding, or from our new stand-point, we get an increase in heuristic-bias. Again, Whigham discussed this as increasing the *language-bias* [Whigham, 1996, Mitchell, 1997].

By it's definition, the attribute grammar of Section 4.2.3 extends the standard CFG mapping process and provides for the generation of 01-compliant solutions. It's extension introduces the first increase in heuristic-bias; yet, even with this constriction it is possible for a 01 compliant solution, to violate a weight-constraint of the problem (infeasible solutions may arise). As such, the fully specified attribute grammar of Section 4.2.4 increases the context-sensitivity captured in the mapping process and essentially provides a mechanism to produce feasible-only solutions. In terms of the work of Gottlieb [Gottlieb, 1999b, Gottlieb, 1999a] we have constrained the search-space to F the feasible-subset of the space of all possible solutions ($F \subset S = \{0, 1\}^n$).

In the design of our initial experimental research, we define three variants of GE: standard GE (*GE*) - to act as the benchmark system; GE using an attribute grammar for 01 compliance (*AG(01)*) - to examine the introduction of context-sensitivity to the mapping process; and GE using a fully specified attribute grammar (*AG(Full)*) - to examine the effect of a context-sensitive mapping procedure that focuses search on the feasible region of the search

[6]α here defining upper-bound on the maximum variable-length achievable by a GE chromosome.

space. GE, and AG(01) both allow the generation of infeasible solutions, and so require what has been introduced as a *constraint-violation handler* to ensure the result of search is a feasible-solution. With the AG(Full) system, a *feasibility-algorithm* exists; the attribute-grammar mapping process providing for feasible-only solutions (See Section 3.5.3). The following sections outlines these experimental systems and their problem-instance test-bed.

4.3.1 Experimental Systems and Problem Instances

The choice of constraint-violation handler has been chosen to be that of penalty-methods (*See Section 3.5.1*). This has been the case, as selection preferences in their standard form, require, coupling with a tournament selection mechanism. As such, for the initial analysis we wish to keep to standard GE paramaters, and thus use fitness-proportionate roulette-wheel selection.

In systems where infeasible solution candidates are an issue (GE and AG(01)), we have two choices for penalisation: a death-penalty (penalising to the worst possible fitness), or a graded-penalty (penalising in proportion - to the degree to which candidate-solutions violate the constraints of the problem (See Section 3.5.1)). At the point, then, of mapping a genotype to an infeasible candidate-solution we have chosen the former approach of the death penalty[7]. The following outlines the operation of each system.

GE This system is a standard GE setup, using the standard parameters and algorithmic configuration (*outlined in Section 4.3.2*). No attributes or semantic-functions exist to give context to the current derivation-step. Infeasible solutions and non 01 compliant solutions can be generated.

[7]Graded penalties, as used by Khuri et al. [Khuri et al., 1994] - were experimented with in [O'Neill et al., 2004b], and reflecting the works of Olsen [Olsen, 1994]; showing no improvement over the death-penalty approach.

Possible solutions are examined for 01 compliance prior to their *EA* evaluation, with failure resulting in a penalty to the worst possible fitness. The system uses a standard CFG as defined in Section 4.2.2.

AG(01) This approach is an extension of GE. The standard GE mapping process is adapted using the attribute grammar of Section 4.2.3, so as to maintain 01 compliant solutions. With regards this attribute grammar, a condition on the *item(I)* attribute - the *notInKnapsack?(i_λ)* condition, allows the identification of a previously mapped item. The inherited attribute, *items(K)* works in tandem with this process; passing down all previously mapped items to each derivation-step. In the case where an item that has been previously derived is detected, the next codon is read and used to re-map (re-derive) the offending non-terminal (I). This approach maintains 01 compliant candidate-solutions; but these may nonetheless be infeasible if violating a weight-constraint of the problem.

AG(Full) In order to further extend GE, this approach wraps the AG(01) mapper as a subset of it's own functionality. This system uses the full attribute grammar as defined in Section 4.2.4. As well as guaranteeing the 01 property, it carries out a constraints check on all m knapsacks, for each terminal-producing production. That is, at the point a terminal symbol (*i.e. an item*) is derived, 01 compliance is ensured (by re-use of the AG(01) features), before further extending GE with yet another test; it ensures that none of the m weight-constraints are violated by addition of the item in question. If a weight-constraint is violated, the next codon is read to re-map (re-derive) the offending non-terminal (I). This system *only* allows the generation of feasible solutions.

Having defined the experimental systems for comparison, it follows that a suitable problem-instance test-suite should be justified. From a review of the literature, many of the most successful works documenting EA approaches to knapsack problems choose the test-suite of problem instances first generated by Chu and Beasley [Chu and Beasley, 1998]. These are very complex problems to solve; so much so, that many do not have a known optimum-solution. Predominantly, these works deal with EA approaches tailored specifically to efficient solving of the 01MKP. They essentially wish to build upon the ground-breaking work presented by Chu and Beasley [Raidl, 1998, Raidl and Gottlieb, 1999a, Raidl and Gottlieb, 1999b, Gottlieb, 1999b, Raidl and Gottlieb, 1999c]; respecting the experience of previous work on constrained-optimisation problems, such as that of Michalewicz [Michalewicz, 1996]. Their focus is to try and develop a system improving on the state-of-the-art, and compare their results on this basis; often adopting controlled order-based genetic operators and local search to achieve these goals.

As it stands, we view our GE approach as somewhat novel; in the application of what is essentially a variable-length GA, operating over a grammar-based GP style, genotype-phenotype mapping process (See Section 1.3.1). Our goal is to examine the suitability of attribute grammars for controlling a derivation over a context-sensitive problem. The context-sensitive problem under empirical investigation here being, the 01 multi-constrained knapsack problem. At this point of research, we don't wish to excel at a particular problem, so much as we wish to investigate the benefits of retrofitting GE with some context-sensitive ability.

Owing to the novelty of the approach, it was decided to choose a problem-instance test-suite, simple enough in the first case; to validate the viability of applying GE to knapsack problems; yet growing in complexity to enable us to examine how the introduction of context-sensitivity re-acts as problems become more difficult. As a side issue, a test-suite enabling a comparison of results to give some kind of indication as to the relative performance was sought after. The test-suite first used by Khuri et. al [Khuri et al., 1994], and later by Cotta and Troya [Cotta and Troya, 1998] - ranging in problem difficulty, from 15 to 105 items seemed suitable. Khuri adopted a fixed-length bit-string representation, with graded penalty term and reported only moderate results. Cotta adopted a problem-space search approach (See Section 3.6.2) and reported superior performance; in both works, results showed that their system's were challenged by the more difficult problems.

The chosen problem instances are available from the OR library, or downloadable on-line from two separate files[8] [Beasley, 1990]. The family of problem instances with the **knap** keyword prepended can be found as instances 3 through 7 in the *mknap1.txt* data-set, and the family of instances prepended by **Sento** or **Weing** can be found in the *mknap2.txt* data-set under corresponding titles. These keywords or recognisers, used in the results of this thesis correspond to those used by Khuri et. al and Cotta and Troya [Khuri et al., 1994, Cotta and Troya, 1998], and we use the same to maintain consistency.

4.3.2 Initial Experiments

We adopt standard experimental parameters for GE, changing only the population size to that of Khuri et al. [Khuri et al., 1994], who use a population size of $\mu = 50$ running for up to 4000 generations. We adopt a variable length one-point crossover probability of 0.9, bit mutation probability of 0.01, and

[8]Available from: http://www.brunel.ac.uk/depts/ma/research/jeb/orlib/mknapinfo.html

fitness-proportionate roulette-wheel selection. A steady-state evolutionary process is employed, whereby a generation constitutes the evolution and attempted replacement of $\mu/2$ children into the current population. Replacement occurs if the child is better than the worst individual in the population. The initial population of variable-length individuals were initialised randomly, with an average length of 20 codons, and standard-deviation of 5 codons from average. Standard 8-bit codons are employed, and GE's wrapping operator is turned off.

These standard parameter settings for GE closely reflect those systems in direct comparison, with the main difference being with Cotta's choice of $\mu = 100$ population size, and what is described as Radcliffe's n-dimensional R^3 crossover operator [Cotta and Troya, 1998]. Primarily, the experimental metric of *percentage of runs yielding an optimum-solution* serve to demonstrate GE's ability to solve these problems. This metric seems to be a standard measure in this field for problems of this size [Khuri et al., 1994, Cotta and Troya, 1998]; whereby more recently, the best competing approaches use the % gap from the LP optimum-solution (*See Section 2.5*); this owing to the absence of a known optimum for these problems [Chu and Beasley, 1998] [Raidl, 1998] [Raidl and Gottlieb, 1999a] [Raidl and Gottlieb, 1999b] [Gottlieb, 1999b] [Raidl and Gottlieb, 1999c]. We also present comparative results for best and average population fitness, where relevant.

A comparison of the standard GE system, the AG(01) system, and the AG(Full) system can be seen in Table 4.1 and Table 4.2. The benefit of adopting an attribute grammar on these problem instances are clear, with the full attribute grammar (AG(Full)), clearly outperforming the two other grammars analysed. On comparison to the results obtained in [Khuri et al., 1994, Cotta and Troya, 1998] it can be seen that the results presented show that for a progressive increase in context-sensitivity in the attribute grammar systems, a corresponding improvement is seen over the standard GE. The AG(Full) system is seen on most instances to outperform the traditional GA of Khuri et al., and provide competitive results to the hybrid GA of Cotta which uses local optimisation over small problem instances. It should be noted, however, that the best of the attribute grammar results fall short of the number of successful solutions found by the best results in the literature [Gottlieb, 1999b].

Tables 4.3, and 4.4 provide results for best-fitness. This metric serves to re-enforce the argument that the addition of attribute grammars to the mapping process of GE, does indeed provide an improvement in performance. These tables show a comparison for the mean best fitness values of the population after a matching number of evaluations with that of Khuri et al.[9].

Finally, tables 4.5, and 4.6 provide results for the populations mean average-fitness. Again these results give testament to attribute grammars as a benefit to the search-capabilities of what would otherwise be a standard context-free derivation, resulting in the generation of many infeasible solutions; presumably acting as noise to the process of evolutionary search.

[9]Cotta carries out $2 * 10^4$ evaluations, but as this work involves the evolution of fitness-landscape (i.e. problem-space search) as opposed to candidate-solutions, it was deemed that the parameter of Khuri have more bearing (See Section 3.6.2).

Problem	**knap15**	**knap20**	**knap28**	**knap39**	**knap50**
Runs Opt(Khuri)	(83%)	(33%)	(33%)	(4%)	(1%)
Runs Opt(Cotta)	(100%)	(94%)	(100%)	(60%)	(46%)
Runs Opt					
GE	3.33%	6.66%	0%	0%	0%
AG(0/1)	50%	30%	3.33%	0%	0%
AG(Full)	83.33%	76.66%	40%	36.66%	3.33%

Table 4.1: Percentage of runs achieving an optimum-solution for *knap* problems: comparing the three grammars and results from [Khuri et al., 1994, Cotta and Troya, 1998].

Problem	**Sento1-60**	**Sento2-60**	**Weing7-105**	**Weing8-105**
Runs Opt(Khuri)	(5%)	(2%)	(0%)	(6%)
Runs Opt(Cotta)	(75%)	(39%)	(40%)	(29%)
Runs Opt				
GE	0%	0%	0%	0%
AG(0/1)	0%	0%	0%	0%
AG(Full)	10%	3.33%	0%	6.66%

Table 4.2: Percentage of runs achieving an optimum-solution for *Sento & Weing* problems: comparing the three grammars and results from [Khuri et al., 1994, Cotta and Troya, 1998].

Problem	**knap15**	**knap20**	**knap28**	**knap39**	**knap50**
MeanBst-(Khuri)	(4012.7)	(6102)	(12374.7)	(10536.9)	(16378.0)
MeanBst					
GE	3892.33	5996.33	12191.0	9526.8	15425.36
AG(0/1)	4000.0	6055.5	12326.66	10145.76	15819.03
AG(Full)	4011.33	6111.0	12377.66	10457.63	16266.13

Table 4.3: Mean-best fitness results for *knap* problems: comparing the three grammars and results from [Khuri et al., 1994, Cotta and Troya, 1998].

Problem	**Sento1-60**	**Sento2-60**	**Weing7-105**	**Weing8-105**
MeanBst-(Khuri)	(7626)	(8685)	(1093897)	(613383)
MeanBst				
GE	7594.4	8527.93	943661.5	587180.06
AG(0/1)	7615.26	8649.3	1067987.966	594746.06
AG(Full)	7693.2	8670.06	1075950.83	605993.63

Table 4.4: Mean-best fitness results for *Sento & Weing* problems: comparing the three grammars and results from [Khuri et al., 1994, Cotta and Troya, 1998].

Problem	**knap15**	**knap20**	**knap28**	**knap39**	**knap50**
MeanBst-(Cotta)	(4015.0)	(6119.4)	(12400.0)	(10609.8)	(16512.0)
MeanAvg					
GE	3888.67	5989.93	12191.0	9526.57	15424.26
AG(0/1)	3998.06	6053.64	12325.70	10145.30	15816.56
AG(Full)	4011.33	6111.0	12377.66	10457.57	16264.76

Table 4.5: Mean-average fitness results for *knap* problems: comparing the three grammars and results from [Khuri et al., 1994, Cotta and Troya, 1998].

Problem	**Sento1-60**	**Sento2-60**	**Weing7-105**	**Weing8-105**
MeanBst-(Cotta)	(7767.9)	(8716.5)	(1095386.0)	(622048.1)
MeanAvg				
GE	7593.322	8523.44	941470.77	586991.36
AG(0/1)	7613.75	8648.45	1059539.97	594746.06
AG(Full)	7693.2	8667.14	1067929.05	605993.63

Table 4.6: Mean-average fitness results for *Sento & Weing* problems: comparing the three grammars and results from [Khuri et al., 1994, Cotta and Troya, 1998].

4.4 Conclusion

We wished to examine the extension of the standard GE mapping process - to handle context-sensitive information, via the medium of attribute grammars. The results demonstrated a clear advantage for the attribute grammars over the standard context-free grammar on the problem instances examined.

The limitations to the expressiveness of context-free grammars (CFG's) were first outlined; with explanation by sample grammars, serving to demonstrate their potential failings. This provided the impetus for the need to explore a method to encode context-sensitivity into the derivation process. Context-sensitive grammars as defined by Chomsky's hierarchy, were dismissed owing to both - the difficulty inherent in the writing of a language processor to capture their phrase structure; and a high-dependency on expert-driven domain-knowledge.

The Fundamental principles of attribute-grammars were outlined with examples showing their workings and explaining their core features; these in the context of a context-free language for the description of candidate 01MKP solutions. The inability of CFG's again re-iterated by example over this knapsack grammar. Two attribute grammars for the different context-sensitive facets of the knapsack language were introduced; with the later a superset of the former providing a method to produce feasible-only candidate-solutions. It was noted that the attribute-grammar defines a decoder for knapsack solutions; drawing correlation between heuristic-bias and the redundant genotype-phenotype mapping. An example mapping process served to outline the intricacies of the attribute grammar decoder.

Finally, the experimental approach and setup was defined with initial results over the core statistical metrics presented; showing attribute grammars as a viable extension to the mapping capabilities of GE.

Chapter 5

Analysis of Introns

5.1 Introduction

The previous chapter outlined our approach to extending the standard GE mapping process, thereby enabling it to conduct a derivation for a context-sensitive language; the use of attribute grammars succeeded in this role. Two degrees of context-sensitivity were outlined for 01MKP solutions in the features of 01 compliance, and constraint-satisfaction; corresponding attribute grammars were defined for each (the latter grammar being a super-set of the former). In an example GE mapping process using these attribute grammar extensions, it was seen that a codon responsible for the derivation of an illegal item was skipped - marked as an intron, and subsequent codons read until a valid item was mapped (*Section 4.2.5*). Introns, we recall, are sections of the genome which do not code for a phenotypic effect.

This chapter serves to examine the genotype's make-up - in terms of these introns. At closer inspection, and as depicted in Figure 5.1, it can be seen that there are two classes of these introns occurring in the genome: those that occur in between exons (or expressed-codons), and those that result as *residue* (or unused genetic material at the tail-end of the genome). The latter form of intron results, when the individual's genotype *over-specifies* the expressed phenotype; i.e. more codons exist than are necessary to carry-out a complete derivation. We will refer to these as *residual-introns*, whereby the former will be referred to as *interspersed-introns (or* IIs)[1]. The region of the genome encompassing exons will be referred to as the *effective-region.* It is worth noting that a sequence of consecutive residual-introns will be referred to as a GE *tail*; a phrase coined by O'Neill et al. [O'Neill and Ryan, 2003b] [O'Neill et al., 2003, O'Neill et al., 2001] to describe the unused genetic material that can result from a GE derivation.

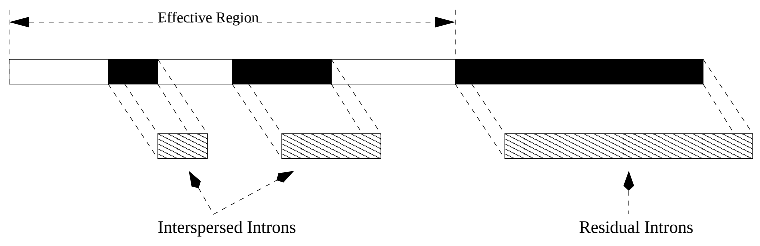

Figure 5.1: Diagram showing the genetic make-up of the genome and terminology adopted - introns are denoted in black, and the shaded regions serve to clearly identify them.

[1]Although similar in concept to the II's described by Nordin et al. [Nordin et al., 1996, Nordin et al., 1997] - these should not be confused as the same. The latter referred to evolved code expressions, evaluating to have no effect on the expressed phenotype, i.e. $x = x + 0$. Here, they represent attempted derivation steps - failing due to constraint violation.

The origin of the forthcoming analysis, stemmed from an initial observation of relatively slow run-times for those systems using an attribute grammar, to those without. Although, efficacy was not a primary goal, in the design of these proof-of-concept systems - the decision to implement the attribute grammar systems, whereby codons responsible for a syntactically incorrect derivation-step, are skipped; came with afore knowledge from the literature that intron propagation was a possibility [Paterson and Livesey, 1997, Paterson, 2002]. Nonetheless, results of other work from the literature suggests merit in an approach which allows the emergence of introns on the genome. Allowing the emergence of introns reflects a close model of nature; whereby, as pointed out by Nordin, Banzhaf and Francone [Nordin et al., 1997] up to 70% of the DNA base pairs in eucaryotic life forms do not code for phenotypic traits (i.e. they are introns). Furthermore, much interesting research has resulted from their inclusion; with Levenick [Levenick, 1991] first reporting that their explicit insertion into the genotype provided improved results. More recent research has provided evidence to support the hypothesis that the presence of IIs, may provide a mechanism to self-regulate the probability of crossover, or protect against building-block destruction [Nordin et al., 1997, Nordin et al., 1996, Nordin et al., 1999]; similarly, Streeter proposes a method of selection based on highly intronated solutions, also reporting improved success over standard methods [Streeter, 2003]. As such, allowing the emergence of IIs was seen as a logical decision - viable for research, and in keeping with our wish to maintain a close model of nature.

In any case, to better understand the behavior exhibited by our initial experiments, some analysis into the genome make-up was required. The following sections document observations of intron propagation, in what is referred to as "bloat" or "code-growth" in the GP domain [Nordin et al., 1996]

[Streeter, 2003]. Interestingly, we observe that the evolution of residual-introns dominate over II propagation.

5.2 Initial Results of Bloat

Owing to the comparative performance difference to the benchmark system (GE), and a general observation of slow execution speeds, our first metric of analysis, was to examine the length to which genomes were evolving to. Figures 5.2 and 5.3, plot the average genome-length for the population. Each plot shows the average genome-length recorded for each system as generations proceed; again, results being reported for an average of 30 runs (Note that the *knap15* problem has been omitted from the figures for clarity; given the nature and simplicity of the problem, it was thought to be of least relevance to the presented analysis). It can be seen that in all systems, the genome-length evolves to greatly over-specify the maximum possible solution size. That is, if closely examined, all three mappers essentially utilise the same CFG in a generative capacity. Consider the following CFG common to all systems[2]:

$$\begin{aligned}
&S \rightarrow K\\
&K \rightarrow I\\
&K \rightarrow IK\\
&I \rightarrow i_1\\
&\quad\vdots\\
&I \rightarrow i_n
\end{aligned}$$

[2] *This grammar is a replication of grammar in Section 4.2.1.*

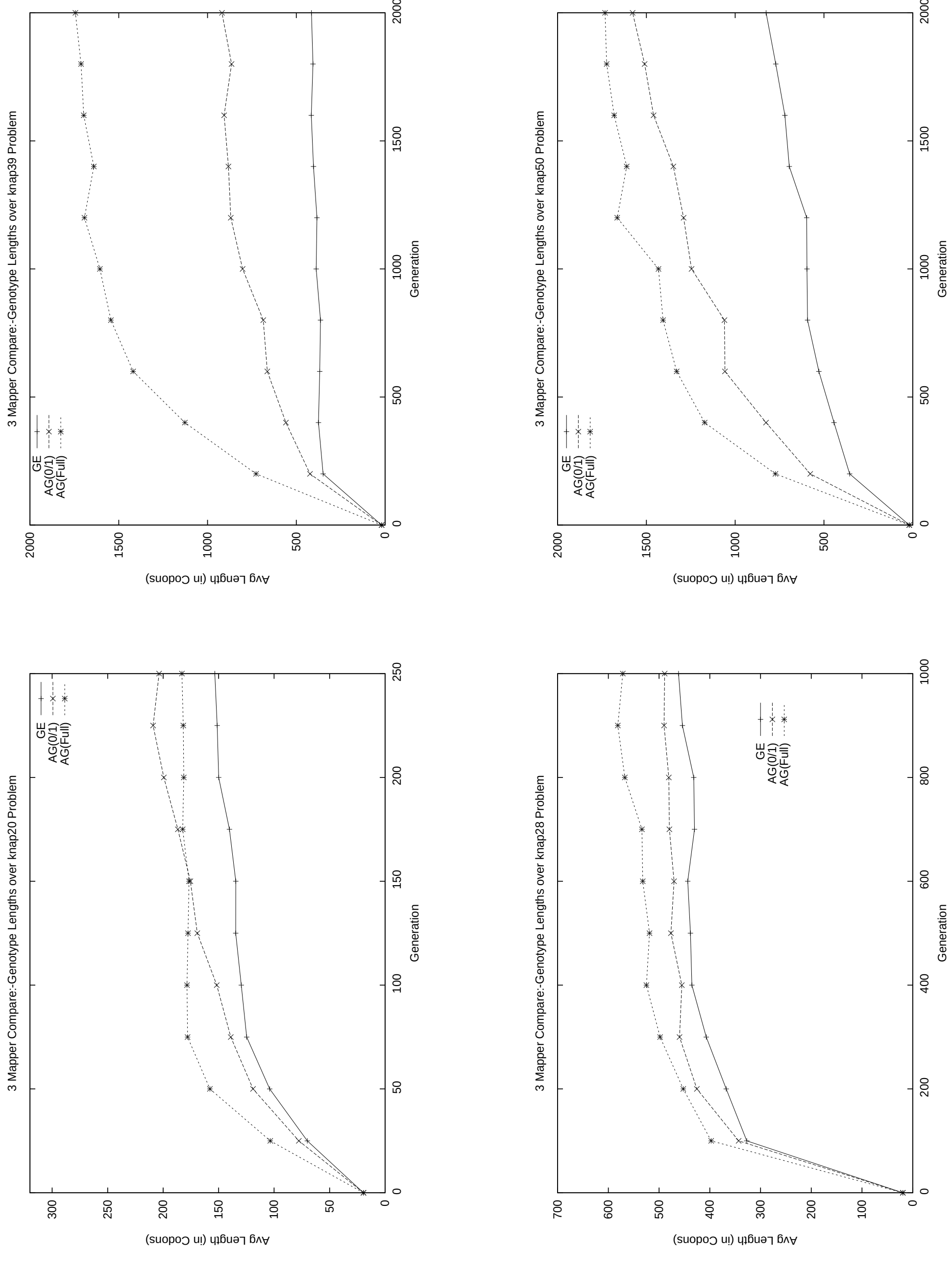

Figure 5.2: Plotting the recorded genome-lengths (averaged over 30 runs), for the 3 systems over *knap* problems.

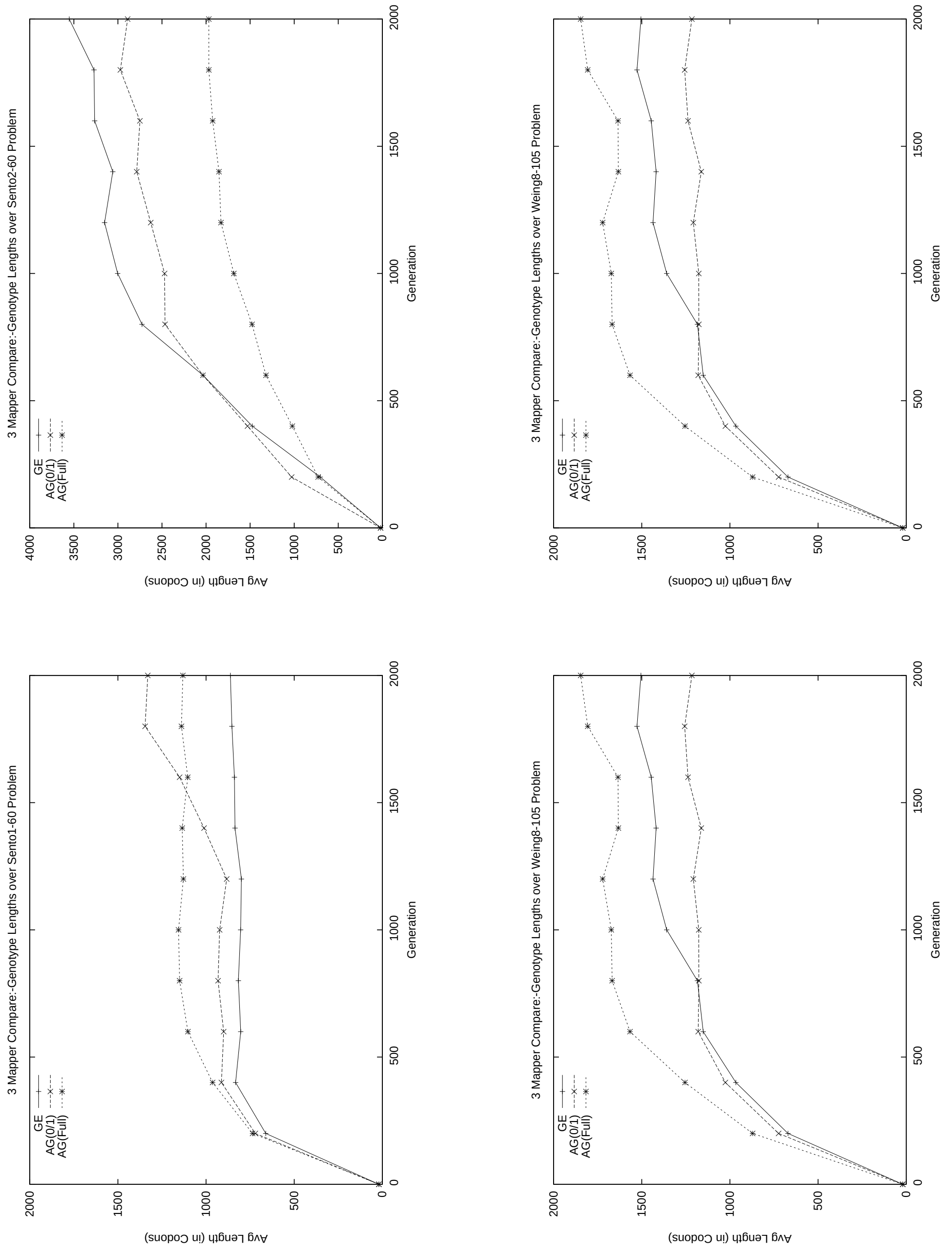

Figure 5.3: Plotting the recorded genome-lengths (averaged over 30 runs), over *Sento &* *Weing* problems.

Distinguishing the attribute grammar systems from standard GE - is the control of the derivation process via semantic-functions (specified within the attribute grammar). As such, from careful examination of this CFG, and the derivation-trees produce-able thereof - it can be seen that it takes 2 codons initially (e.g. $S \rightarrow K \rightarrow IK \rightarrow i_\lambda K \ldots$)[3], and subsequent iterations of 2 codons thereafter (e.g. $\ldots \rightarrow IK \rightarrow i_\lambda K \ldots$, OR $\ldots \rightarrow I \rightarrow i_\lambda$) in order to derive items.

Now, given an n-item problem - it is *possible* for an individual with

$$2 + (2 * (n - 1)) \tag{5.1}$$

codons, to completely specify a solution containing exactly n items (This equation is specific, according to the grammar used). As conditions dictate, a solution of this size (the set of all n items) may be infeasible; but equation 5.1 can be used as a measure for the *lower-bound* to the number of codons required to completely specify n items[4]. As such, a 20-item problem can be seen by equation 5.1 to require a maximum of 41 codons to completely specify n items (20 in this case); for the special case, that is - where all codons would result in the derivation of a legal item. Thus, equation 5.1 serves as a lower-bound to the number of codons needed to code for an n-item solution.

Turning again to Figures 5.2 and 5.3, we can see that as problems grow in size, the number of codons (i.e. the genome-length) is seen to hugely over-specify the maximum requirement as dictated by equation 5.1. Also, it can be seen in Figure 5.3, where problems significantly increase in size and difficulty, that standard GE out-grows the AG(01) system, with the AG(Full)

[3]Note that for $S \rightarrow K$ no production decision exists, and so no codon is required.

[4]In terms of a context-free derivation, it could be seen as a maximum-codon measure: the maximum number of codons required to derive n terminal symbols. As suggested above, we feel the *lower-bound* better reflects the attribute grammar system.

system producing very large individuals. In essence, we can generalise that all systems suffer from the phenomenon previously described as bloat. The following sections seek to provide a more in-depth analysis of the genome make-up in order to further understand the reason for this behavior.

5.3 Analysing the Genome Make-Up

To further analyse the genome make-up, the approach taken was to examine the average presence of the two previously described classes of intron as they occur along the genome. In order to do this, we first looked to provide a view, of the proportion of the genome comprised of these different classes of intron; in so doing, we examined the proportion of IIs (interspersed-introns) and residual-introns separately. Following from this, a specific examination of II-regions - in terms of structure and number - will be explored (*refer again to* Figure 5.1).

5.3.1 Proportion of Introns

Figures 5.4 and 5.5 show plots depicting the proportion of the genome comprised of residual-introns; that is, these plots show the average amount of the genome which is un-used (i.e. comprise of a GE-tail). As the figures illustrate, a vast proportion of the genome is comprised of redundant material. In some cases, up to 90% of the evolving genomes, are composed of tails of residual-introns. This behavior is seen to be consistent for all three systems under investigation.

Figures 5.6 and 5.7 serve to reflect the findings of the tail-proportion experiments just described; plots are shown for the proportion of the genome comprised of interspersed-introns (IIs). These plots show the average amount of the genome which is made up of IIs. Plots for standard GE, show a flat line in all graphs; this is due to it's context-free mapping process which doesn't permit the introduction of II-regions. The figures reflect the findings of the previous tail-proportion plots, showing that only a very small amount of the genome is comprised of these IIs; further examination is required in order to study their size and structure, within the effective region of the genome. At this juncture, however, it is worth noting that the AG(Full) system seems to incorporate approximately twice the amount of IIs that the AG(01) system produces. This is not surprising, as the AG(Full) system specifies a more constrained language and will obviously encounter more problem-constraint conflicts: this, resulting in the introduction and subsequent emergence of more II-regions.

This observed behavior of high tail-proportion is seen to be consistent for all three systems, with the exception of the plot for the *knap20* problem, where the AG(Full) produces a very low tail proportion comparatively (Figure 5.4). A correspondingly high II proportion (Figure 5.6) reflects this observation, but this metric alone gives no insight as to why this may be. It was suspected that this stagnation of tail-growth may be a result of the population having converged to the optimal fitness: therein removing selection-pressure for larger individuals. Indeed, and as depicted in Figure 5.8, this appears to be the case. The figure shows a plot for the mean-best fitness for each experimental system; it can be seen that after approximately 30 generations the AG(Full) system converges to the optimal fitness of 6120. This point in the evolutionary process correlates with the observed stagnation for tail growth. As such, this is suggestive that selection pressure is selecting for fit individuals having a large proportion of redundant material in the tail region; the initial sharp rise in all graphs further supports this to be the case.

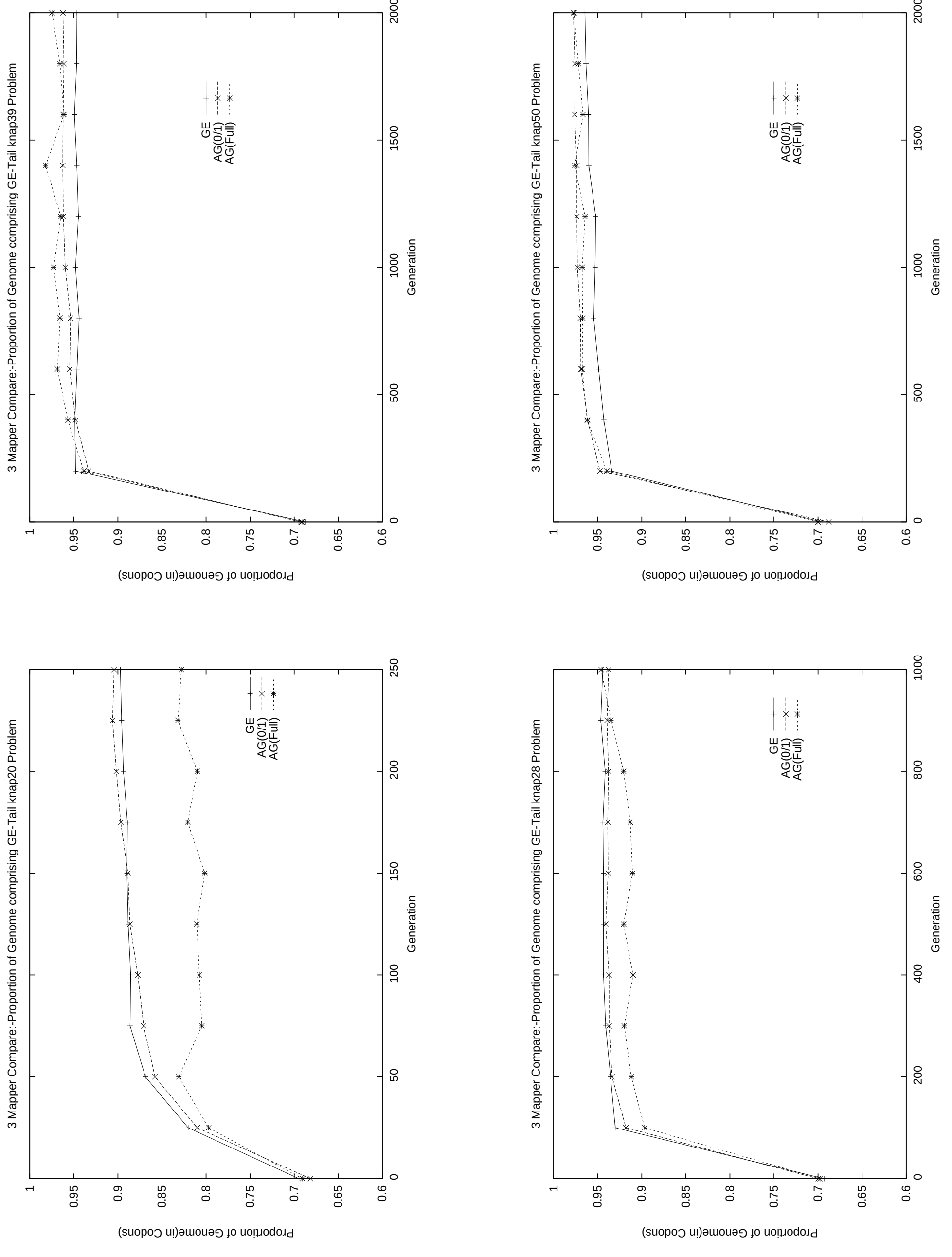

Figure 5.4: Plotting the proportion of the genome (averaged over 30 runs) comprising of residual-introns or GE-tail, for *knap* problems.

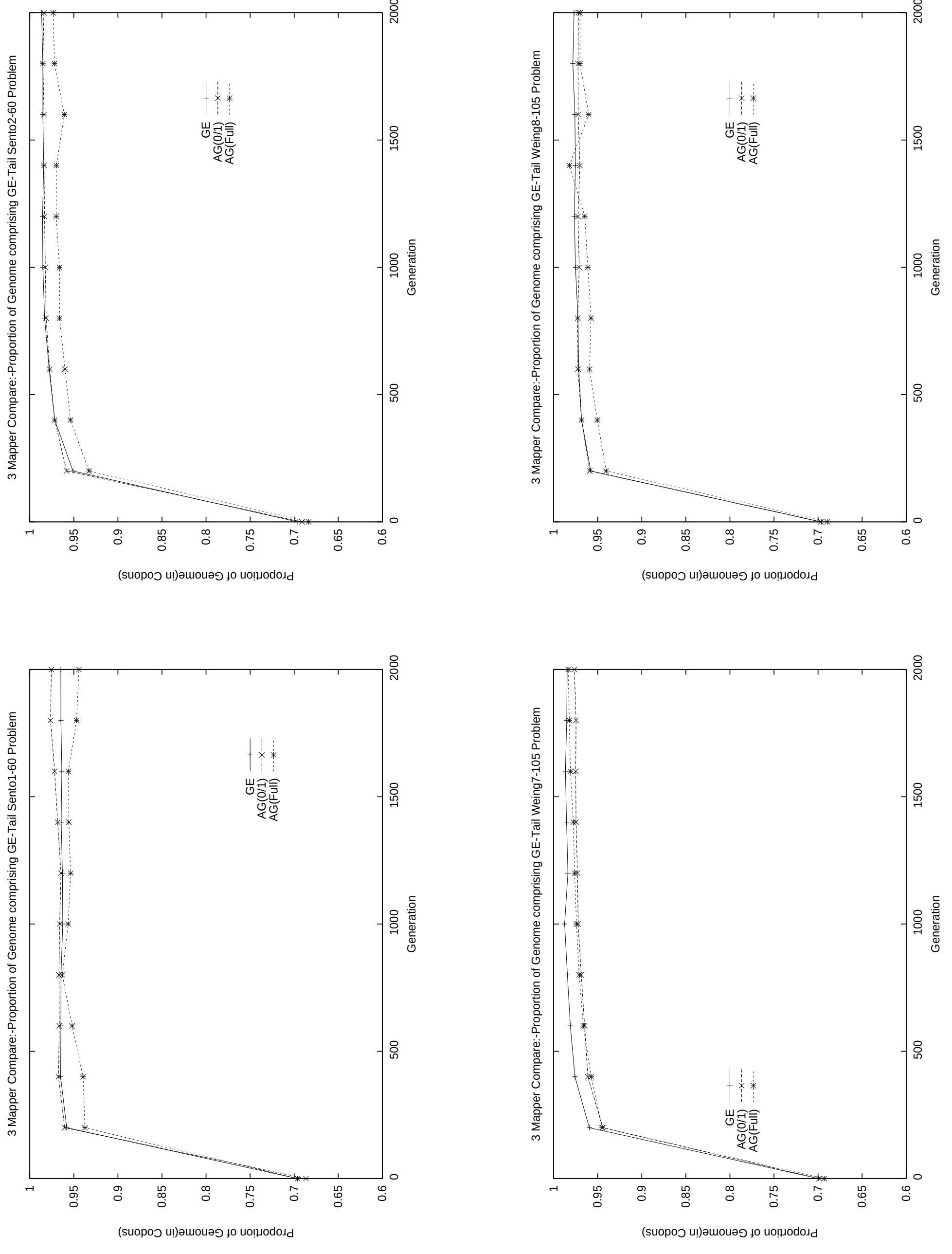

Figure 5.5: Plotting the proportion of the genome (averaged over 30 runs), comprising of residual-introns or GE-tail, for *Sento & Weing* problems.

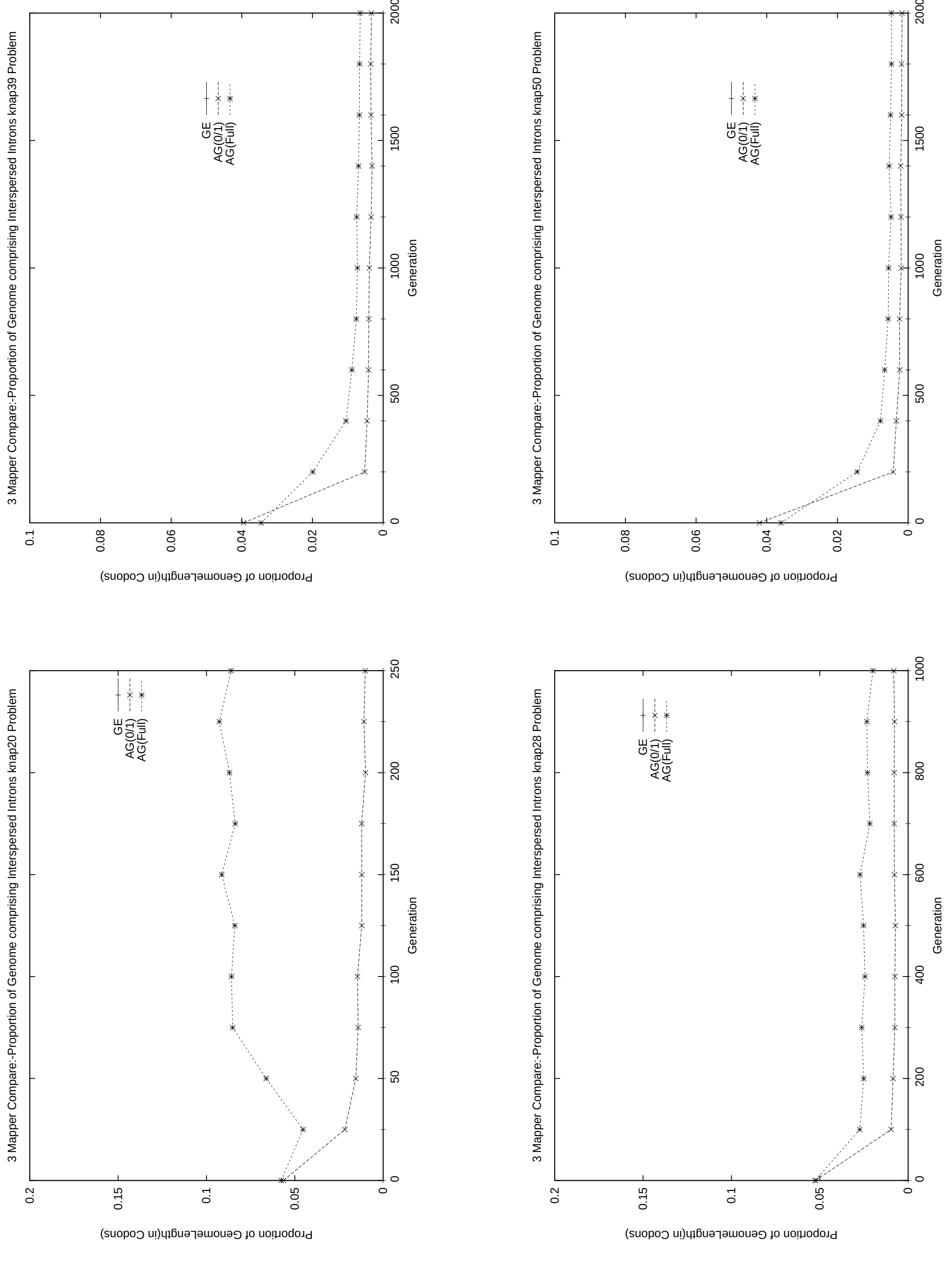

Figure 5.6: Plotting the proportion of the genome (averaged over 30 runs), comprising of interspersed-introns (IIs), for *knap* problems.

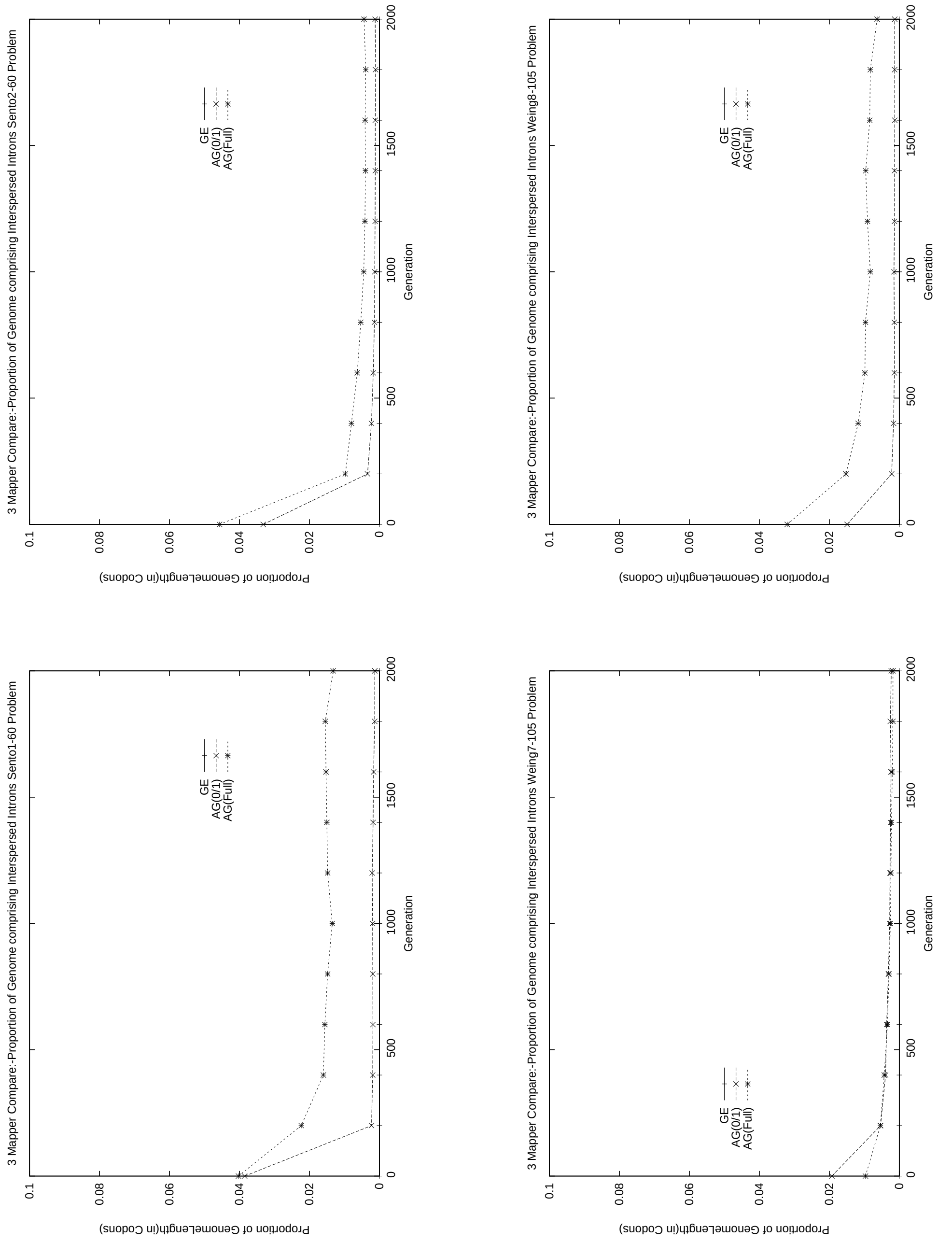

Figure 5.7: Plotting the proportion of the genome (averaged over 30 runs), comprising of interspersed-introns (IIs), for *Sento & Weing* problems.

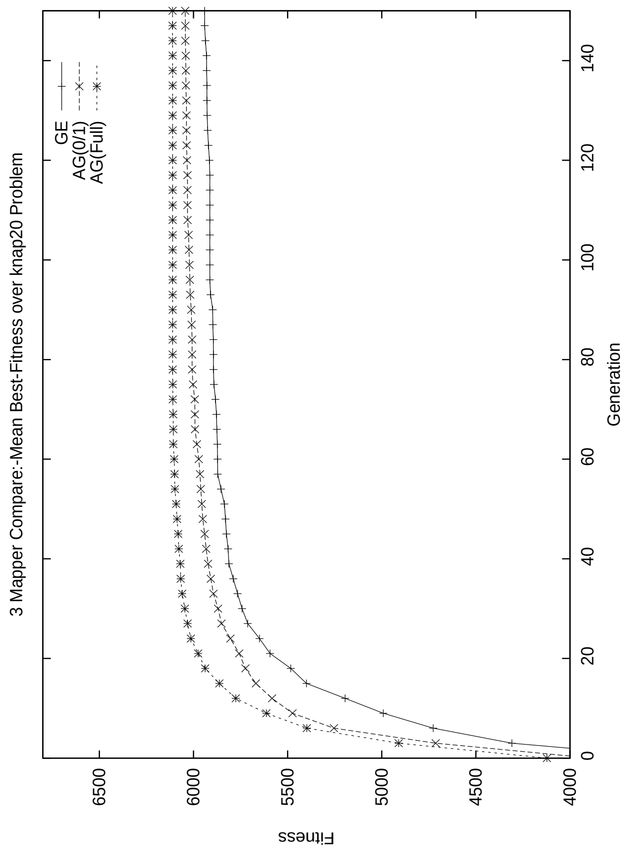

Figure 5.8: Plotting the mean-best fitness for the 3 systems (averaged over 30 runs). AG(Full) system is shown to converge to the optimal fitness of 6120 after approximately 30 generations: this correlates to the observed stagnation of it's genome's tail growth; as seen in Figure 5.4.

5.3.2 A Focus on Interspersed-Introns

Realising by analysis, that the evolving genomes are largely comprised of residual-introns, did not serve to provide any real insight into the difference (in terms of the structures evolving), which one system provides over the other. Early analysis showed the fully attributed system (AG(Full)) to be the best performer; our wish at this point, is to examine the structural dynamics along the evolving genome which may be contributing to this behavior.

As such, there follows a focused analysis of the 3 Systems, in terms of the II structures found (on average), along the genome. Again, standard GE shows a flat line in all graphs; this, as it's mapping process doesn't permit the introduction of IIs. Figures 5.9 and 5.10 show the average number of *II-regions* occurring (a contiguous sequence of IIs within the effective-length); with Figures 5.11 and 5.12, showing the average length of such regions. As previously discussed, it is not surprising to see the AG(Full) system exhibiting more of these II-regions; and with larger average size. However, one problem again breaks the consistent behavior exhibited over all others. The very restrictive problem of *Weing7* - for which we fail to report an optimal solution - shows the two attribute grammar systems exhibiting much the same structure; in terms of IIs. This is thought to be a result of premature convergence; as this is a highly restrictive problem (the feasible subset of the absolute search-space is very small), both systems seem to be unable to escape local optima and exhibit the same - highly intronated - structural evolution. This owing to relatively equal constraint-clashes for both systems.

What is interesting, and more significant in terms of information-gain to our analysis - is the possible positive effect these entities have on evolution. The following section seeks to examine to effect on behavior, when such introns are removed from the evolutionary process.

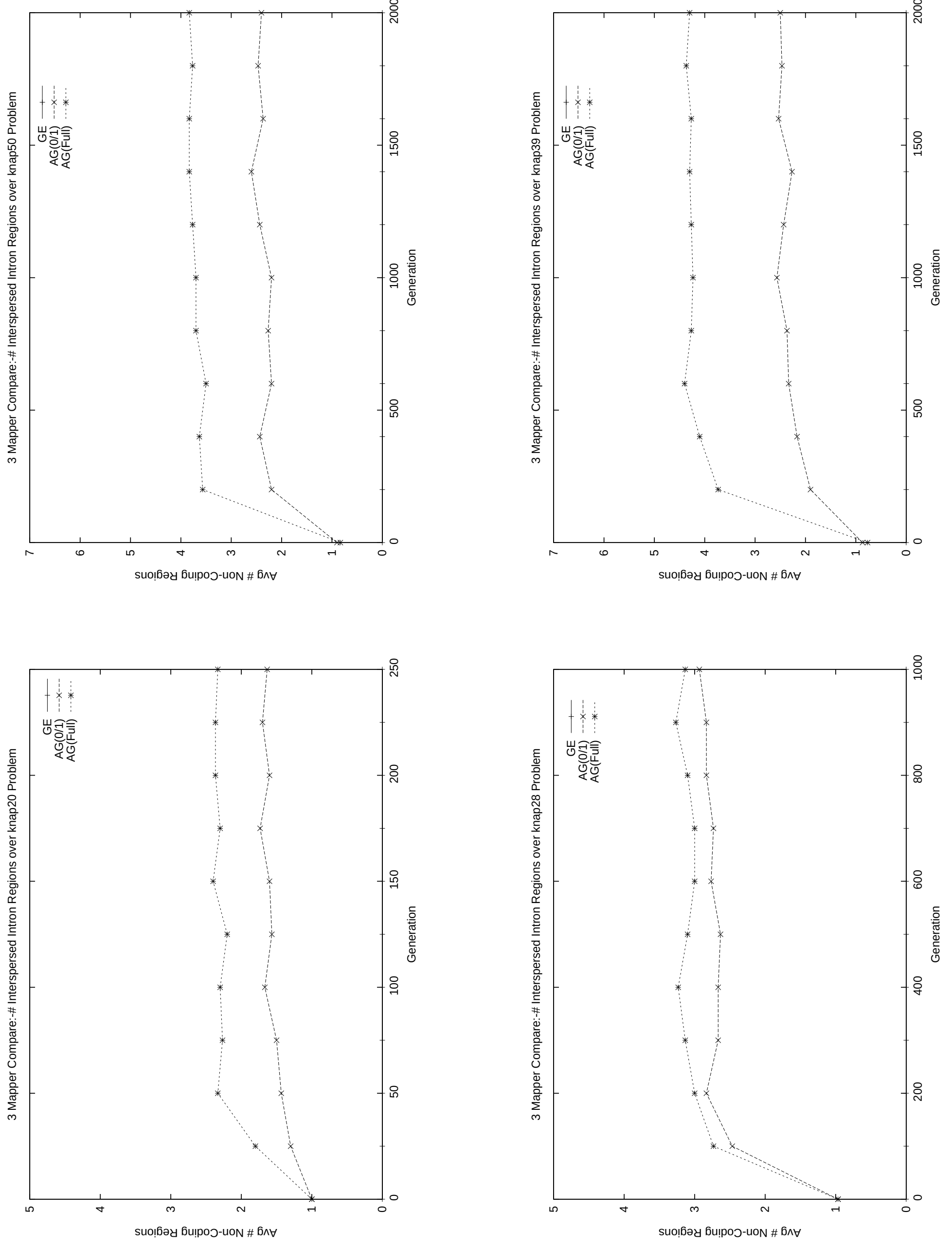

Figure 5.9: Plotting the average number of II Regions (averaged over 30 runs), occurring within the effective-region of the genome, for *knap* problems.

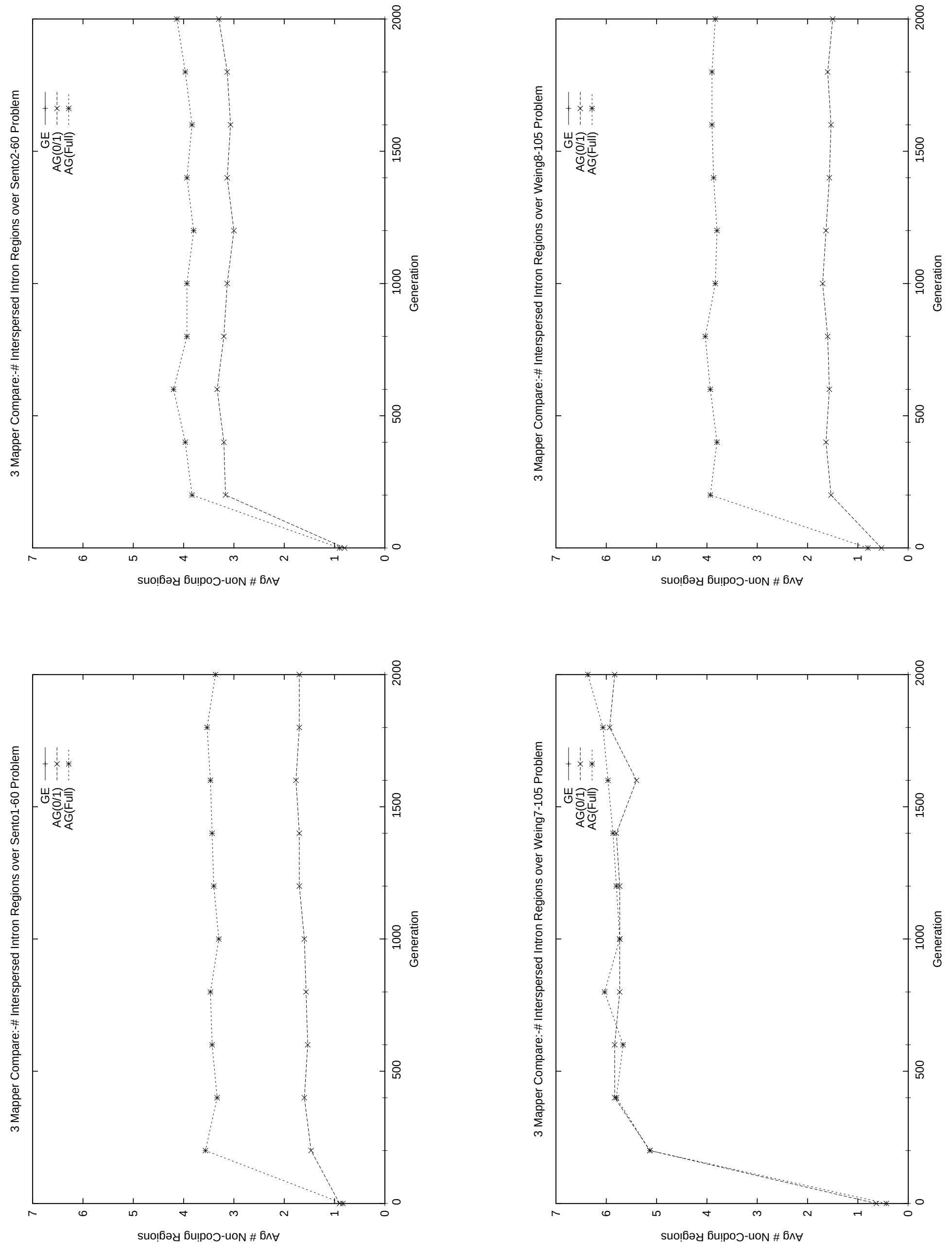

Figure 5.10: Plotting the average number of II Regions (averaged over 30 runs), occurring within the effective-region of the genome, for *Sento & Weing* problems.

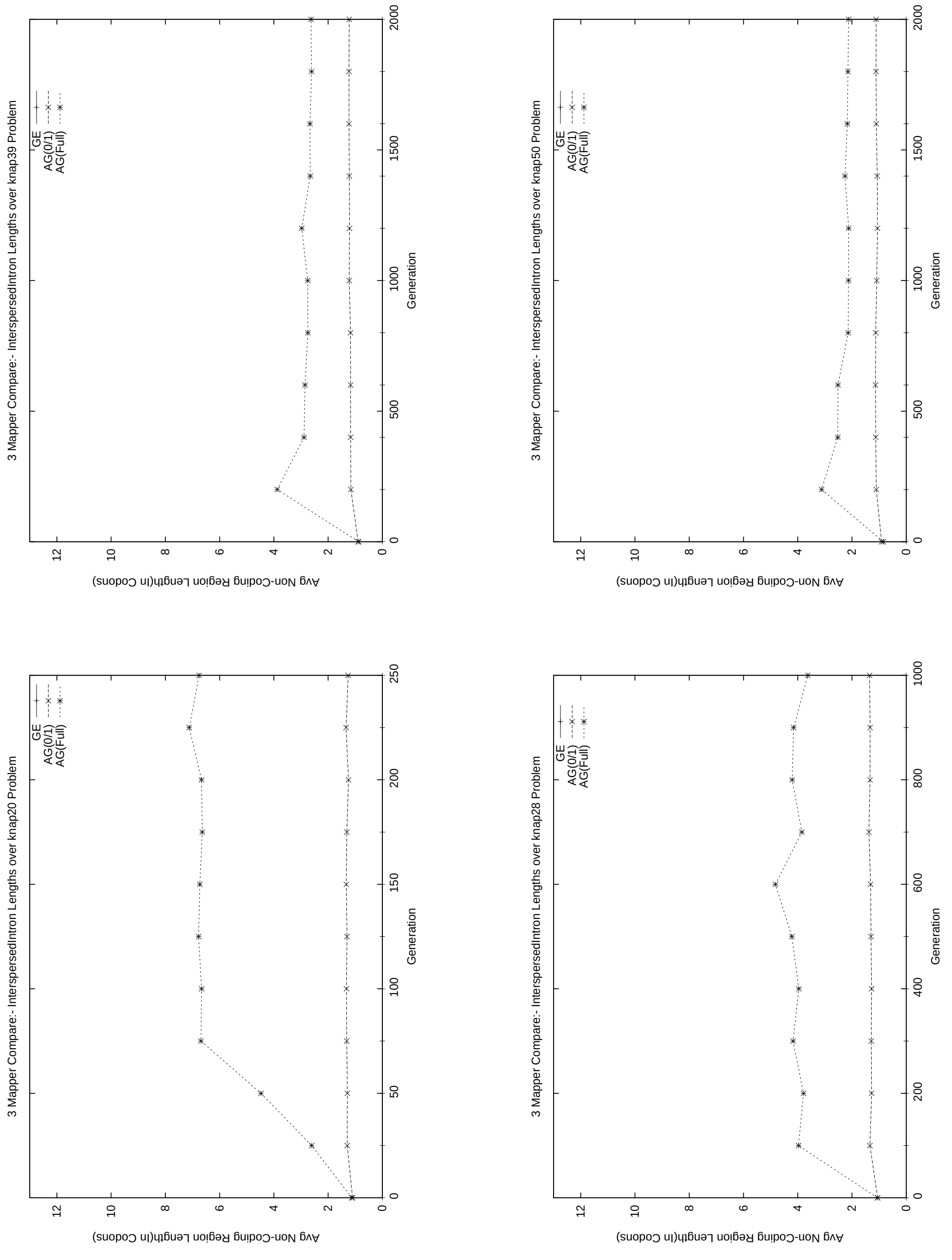

Figure 5.11: Plotting the average length of II-regions within the genome (averaged over 30 runs), for *knap* problems.

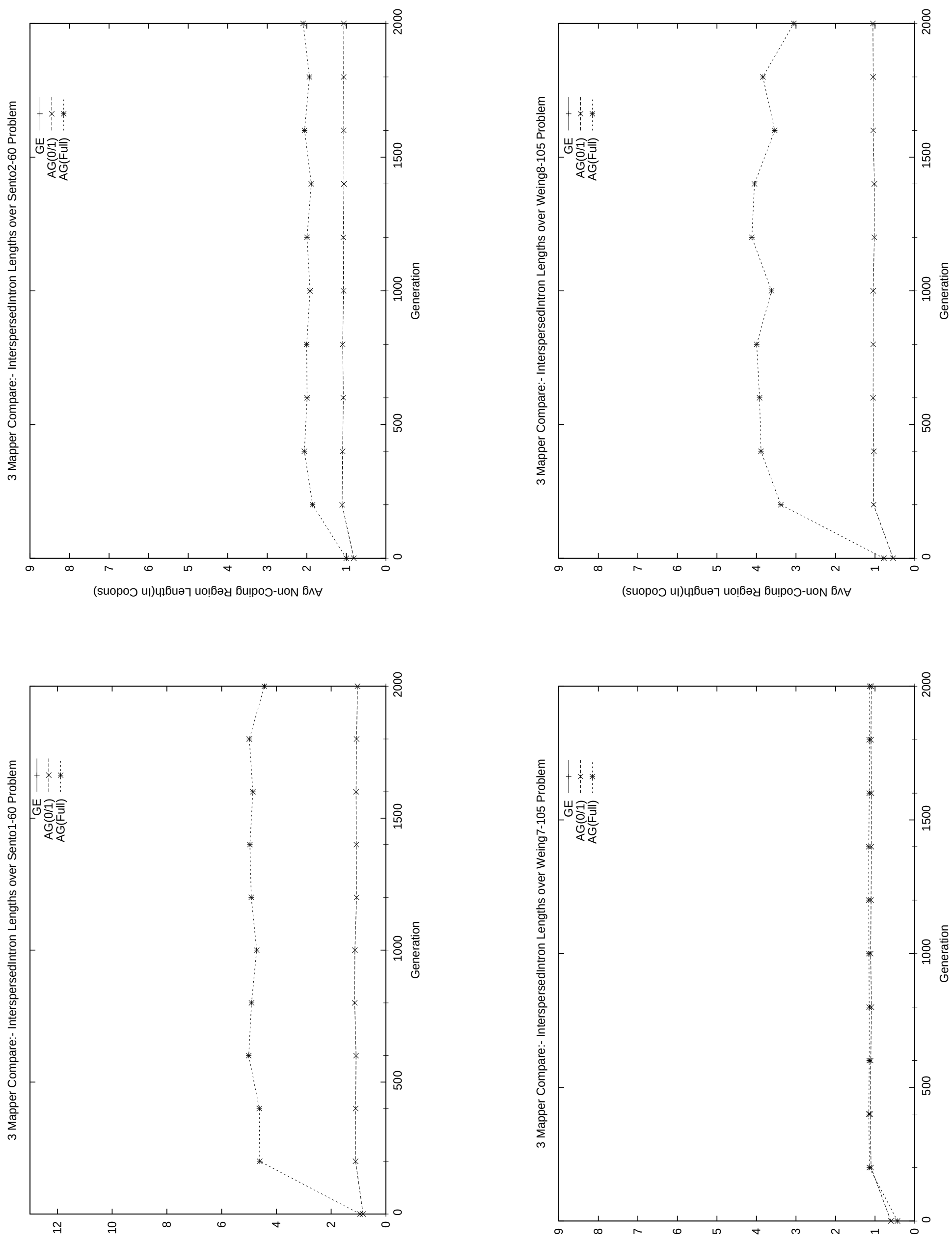

Figure 5.12: Plotting the average length of II-regions within the genome (averaged over 30 runs), for *Sento & Weing* problems.

5.4 Intron Removal Strategies

We enter this section having conducted the initial analysis of the systems under empirical investigation (Section 4.3.2). At this point, and in light of the initial analysis - we reduce further investigations to (the most promising) AG(Full) system, and attempt to discover the role of the various non-coding components of the genome therein. We wish to discover if there is anything structural, giving this system a comparatively superior performance over the others. Primarily, we wish to study if the presence of introns provides any positive effect to the evolutionary process.

In this examination, we introduce a set of proprietary genetic-operators which allow various combinations of intron removal. The purpose and goal of which - is to allow us to discover the effect of their presence, on this particular system's performance. Each operator tested, carries out intron-removal prior to replacement; an evolving population of *altered* genetic structures, is subsequently maintained. In this way, the operators work in a form of Lamarkian evolution. To better describe the operators, each section will provide a graphical example to illustrate how they operate.

5.4.1 Pruning

Results from previous analysis in this chapter, identified the propagation of individuals with exceptionally large genome-lengths (Section 5.2). Further analysis showed that genomes were predominantly comprised of *residual-introns.* That is, this redundant non-coding material at the end of the genome, commonly known as the *GE-tail* [O'Neill and Ryan, 2003b] was being propagated from generation to generation resulting in the phenomenon of bloat. Nordin and Banzhaf [Nordin et al., 1997], in the context of a linear

GP system, suggest that this particular type of intron-propagation is suggestive of an emergent protection-mechanism, against any destructive-effects exhibited by the evolutionary operators; predominantly crossover. Similarly, Streeter also takes this view, summarising several other works which propose comparable theories based on the same premise [Streeter, 2003].

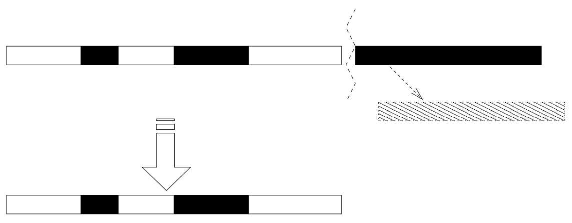

Figure 5.13: Diagram illustrating the operation of the pruning operator.

Born out of this work, we attempt to analyse - the effect to evolution - when we remove these *tails* of residual-introns, by a genetic *pruning* operator similar to that described in [O'Neill and Ryan, 1999]. Figure 5.13, illustrates the operation of the pruning technique; again, the operator takes effect immediately after the evaluation process - prior to replacement. The population is thus maintained such that evolution occurs over genomes which have had their tails "pruned" or cut away. interspersed-introns (IIs) which emerge via the AG(Full) algorithm remain. This is essentially a form of genetic *clean-up*, removing the genetic material which was unused (i.e. the *residue*) in the mapping of genotype - to phenotypic solution.

In terms of evolution, this operator has the effect of magnifying the effective-region; forcing crossover and mutation to occur within this space. In terms of the defacto AG(Full) algorithm, free from pruning - this oper-

Problem	knap15	knap20	knap28	knap39	knap50
AG(Full)	83.33%	76.66%	40%	36.66%	3.33%
Pruning	66.66%	60%	10%	10%	6.66%
Problem	**Sento1-60**	**Sento2-60**	**Weing7-105**	**Weing8-105**	
AG(Full)	10%	3.33%	0%	6.66%	
Pruning	6.66%	0%	0%	6.66%	

Table 5.1: Comparing problem success-rate of the *pruning* strategy over the AG(Full) control: the effect of pruning GE-tails. Table shows the % of successful runs (i.e. *achieving an optimal solution*) over 30 independent runs

ator provides an increase in probability of crossover occurring within the effective-region; the increase being directly proportional to the size of the pruned tails. As mutation is bitwise, the probability of mutation remains unaffected. Based on the observations of Nordin and Banzhaf, removal of this redundant genetic material, decreases what they term as the effective-region's "compression", and we would expect a direct decrease in performance as a result [Nordin et al., 1999] [Nordin et al., 1997]. Indeed the results support this, and give rise to the suggestion that our standard genetic operators may well be of destructive or predominantly neutral influence. Further analysis is required in order to further substantiate this observation.

Results Table 5.1 shows the comparative performance of the AG(Full) system with pruning, compared to the AG(Full) system, without; results presented are an average of 30 independent runs. Employing the pruning strategy shows a degradation of performance over most problems; with the exception of the *knap50* problem - where an increase of 1 more successful run is achieved. For the more difficult *Weing* problems, however, there is no reported difference between the pruning system and the standard prune-free.

5.4.2 Splicing

The splicing operator has the effect of removing the interspersed-introns introduced by the mapping process. Again, removal occurs immediately after evaluation of the genome, and prior to replacement in the population. That is, the population is maintained such that evolution occurs over genomes which have had their interspersed-introns "spliced". The GE-tail or residual intron sequence remains. If you like; this operator enforces a form of genetic

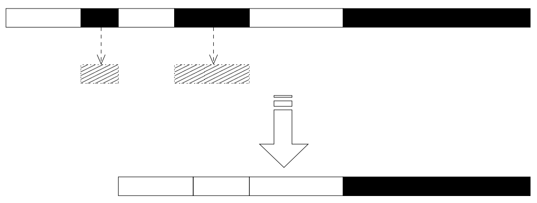
Figure 5.14: Diagram illustrating the operation of the splicing operator.

repair; translating the genome into a sequence of codons, which subsequently represent a contextually-correct derivation sequence (an uninterrupted sequence of expressed codons (or *exons*)).

In effect, this operator shortens the *effective-region*, increasing it's compression against the absolute length of the genome. This has the effect of reducing the probability that crossover will occur in this area. As can be seen in Figure 5.14, this reduction in the probability corresponds to the proportion of the entire genome which had formerly been comprised of interspersed-introns.

Problem	knap15	knap20	knap28	knap39	knap50
AG(Full)	83.33%	76.66%	40%	36.66%	3.33%
Splicing	86.66%	80%	30%	33.33%	3.33%
Problem	Sento1-60	Sento2-60	Weing7-105	Weing8-105	
AG(Full)	10%	3.33%	0%	6.66%	
Splicing	40%	13.33%	0%	33.33%	

Table 5.2: Comparing problem success-rate of the *splicing* strategy over the AG(Full) control: the effect of IIs. Table shows the % of successful runs (i.e. *achieving an optimal solution*) over 30 independent runs

Results Table 5.2 shows the comparative performance of how the AG(Full) system *with* this splicing strategy compares to the standard AG(Full) system which is untouched; again, results presented are an average of 30 independent runs. With this strategy we see a slight increase in success for the easy problems of knap15 and knap20, a slight decrease in performance for the problems with up to 50 items; but a marked increase in performance is recorded for the larger, and more difficult, Sento and Weing problems. Alternatively, Table 5.3 shows how the splicing strategy measures-up against the previous *pruning* strategy. On the whole, splicing shows a marked improvement in performance over the pruning strategy - making it the best-to-date (results are significantly better for all but the *knap50* problem, for which pruning produces only one more optimal run to splicing).

In all, the behavior depicted in Table 5.2 and 5.3 serve to suggest that - in the presence of a GE-tail - IIs occurring along the genomes' effective-region are of negative effect to performance (i.e. splicing them, improves performance). Looking again at figure 5.14, it can be seen that evolution occurs over sequences of un-interrupted, contextually-correct codons (a contiguous sequence of codons, which result in a contextually-correct derivation-step

Problem	knap15	knap20	knap28	knap39	knap50
Pruning	66.66%	60%	10%	10%	6.66%
Splicing	86.66%	80%	30%	33.33%	3.33%
Problem	Sento1-60	Sento2-60	Weing7-105	Weing8-105	
Pruning	6.66%	0%	0%	6.66%	
Splicing	40%	13.33%	0%	33.33%	

Table 5.3: Comparing problem success-rate of the *splicing* strategy over the previous *pruning* strategy. Table shows the % of successful runs (i.e. *achieving an optimal solution*) over 30 independent runs

- at each mapping to a terminal-symbol or item) where the presence of a GE-tail of residual-introns exists. This removal of interspersed-introns, and consequent increase in compression - of the effective-region - over the absolute length of the genome, would appear to be favourable for the creation and exchange of sub-solutions or building blocks. A spiral in tail-growth remains; and this would seem to support the suggestions of Nordin and Banzhaf [Nordin et al., 1997], that this residual-intron propagation acts as a form of self-regulation to the probability of crossover.

5.4.3 Splicing & Pruning

This system setup, combines the two operators previously described in Section 5.4.2 and Section 5.4.1, having the effect of removing all introns from the genotypic material. Again, operators work immediately after evaluation of the genome, and prior to replacement. With this setup, splicing removes any IIs introduced by the attribute grammar's mapping process; and pruning removes the residual intron sequence of the GE-tail. Figure 5.15 shows the effect of this system setup in operation; it is akin to both genetic repair and clean-up. This strategy effectively operates by removing all introns so that

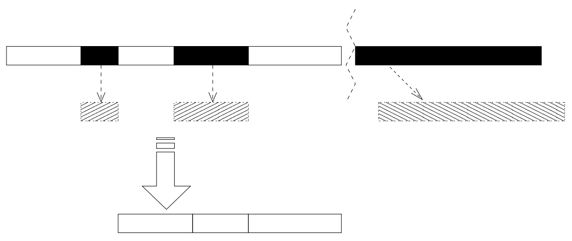

Figure 5.15: Diagram illustrating the effect on the genome of the *splice+prune* strategy.

evolution occurs over genotypes, which subsequently represent contextually-correct derivation sequences. In terms of evolution, this setup is essentially an extension of the pruning strategy (in which, the evolutionary operators of crossover and mutation *must* occur in the effective-region); the major difference here being, that crossovers occurring at interspersed intron (II) sites can no longer occur. It was hoped that this measure would help to provide us with an observation either in support or contradiction to the findings of Nordin and Banzhaf's research - suggesting *a*) that introns exist to guard against any destructive effects of crossover, and *b*) that IIs act as a form of sub-solution protector [Nordin et al., 1997].

Results As can be seen from the results presented in Table 5.4, this strategy shows a degradation in performance over all problem instances, and gives the worst success rate of all the intron-removal strategies documented in this chapter. In terms of extending the pruning strategy - removing interspersed-introns (IIs) (as well as tails), we see a marked decrease in success-rate over all but the knap15 problem; who's success improvement is negligible. This would seem to suggest, that the difference to pruning (i.e. the presence of interspersed-introns) is of benefit to search[5]; Figure 5.16 serves to demonstrate the difference visually. In this context then, the observed results for the

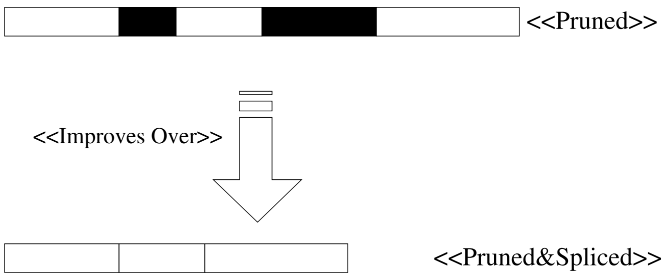

Figure 5.16: Diagram illustrating the difference between evolving structures for pruning strategy (Top), against splice+prune (Bottom).

(beneficial) presence of IIs in pruning, gives credence and supports the argument of Nordin and Banzhaf, who describe IIs to provide a "structural protection" of building-blocks or sub-solutions. That is, their presence may guard against any sub-solution disruption which the genetic operators can cause. Where splicing-alone was seen to further the compression of the effective-

[5]It is worth noting, that this, positive observation as to the presence of IIs, can only be made in the absence of a GE-tail; if a tail is present, the splicing strategy has illustrated that II-removal leads to improved performance.

Problem	knap15	knap20	knap28	knap39	knap50
AG(Full)	83.33%	76.66%	40%	36.66%	3.33%
Splice+Prune	73.33%	40%	3.33%	3.33%	0%
Pruning	66.66%	60%	10%	10%	6.66%
Splicing	86.66%	80%	30%	33.33%	3.33%
Problem	**Sento1-60**	**Sento2-60**	**Weing7-105**	**Weing8-105**	
AG(Full)	10%	3.33%	0%	6.66%	
Splice+Prune	6.66%	0%	0%	0%	
Pruning	6.66%	0%	0%	6.66%	
Splicing	40%	13.33%	0%	33.33%	

Table 5.4: Splice+Prune results: comparison to the AG(Full) system, and all previous systems examined. Table shows the % of successful runs (i.e. *achieving an optimal solution*) over 30 independent runs

region against the tail of residual-introns, Table 5.4 also shows how eradicating this compression (adding pruning to splicing - removing the GE-tail), results in an extreme drop in performance.

5.5 Conclusion

This chapter has essentially presented an analysis of the genetic structures evolving (the variable-length binary genomes of codons). We have examined these genomes in terms of their make-up, focusing on the relations between coding-regions (exons) and non-coding regions (introns). A common-sense, preliminary analysis led to the naming of two different classes of introns; *interspersed* and *residual* respectively (Figure 5.1). A logical trail of analysis, stemming from the initial results of what is commonly referred to as code-growth (or bloat) - led to a focused analysis of the genome make-up in terms of introns.

An exceptional behavior observed for the AG(Full) system, over one particular problem-instance - led us to deduce that a stagnation in tail-growth can be indicative of convergence to the optimal solution; and a consistent rise in tail-proportion for all systems in the early generations gives credence to the notion that these residual-introns are being selected-for in highly fit individuals.

A look at the length and structure of IIs enforced our belief that the AG(Full) system brings about the insurgence of more IIs; with II-regions (a contiguous sequence of IIs within the effective-length) being generally longer - owing to the greater number of problem-constraint clashes (clashes occurring as a consequence, of the more tightly constrained search space explored by the attribute grammar system).

This observation, gave rise to an analysis, with a mind to examining - whether or not the presence of these introns, was providing positive support to search - and if so, in what role. Previous work from the literature, provided the impetus to examine intron removal strategies [O'Neill and Ryan, 1999] [Nordin et al., 1997, Nordin et al., 1996] and how to subsequently examine their contribution to search. Bourne out of this study of the literature, a set of proprietary intron removal operators allowed a comparative *with/without* study, to examine the effect of introns. The result of this analysis - leading us to believe (and in support of Nordin and Banzhaf's argument) - that both forms of intron identified act in a role of sub-solution, and indeed possibly full-solution protection.

The *splicing* strategy emerged as the best performer; early analysis suggested that, this is primarily due to a compression of the effective-region; with the splicing of IIs, effectively acting as a form of what Nordin and Banzhaf describe as a "global-protection" mechanism. In this context, "global-protection" eluding to a preserving behavior - where large tails of residual-introns serve to protect the current effective-region (a whole solution) - from disruption through crossover[6].

Two other strategies of *pruning* and *splice+prune* (a combination), provided results to suggest that in the absence of the previously observed compression, IIs act as a form of sub-solution protector. The combined observation of the *splice* and *splice+prune* strategies, allowed further analysis of the evidence. We see that our results to-date provide evidence to support a two-prong argument *a*) that the GE-tail, is essential to the performance improvements observed; and also, *b*) that IIs (in the absence of a tail) provide a form of protection against any disruption effected by the evolutionary operators.

The former argument could be viewed, as by Nordin and Banzhaf - that the GE-tail provides a compression of the effective-region; this compression results in a self-adaptation of the probability of crossover: as yet, however, we feel there is not enough evidence to make the claim that this is the case. Similarly, the latter argument - suggesting that IIs play a role in structural protection of building blocks or sub-solutions; also requires further research.

As it stands, we cannot deny that Nordin and Banzhaf's observations (over a linear GP system) seem to hold true for our attribute grammar extension of the standard GE. The subsequent analysis will attempt to strengthen or weaken this analysis : looking more closely at the evolutionary process and it's core operating components.

[6]*Note:* as mutation is bitwise by probability, the observed compression will have no bearing to any of the disruptive forces to which *it* may exhibit

Chapter 6

A Closer Look at Evolution

6.1 Introduction

The analysis to-date, has identified the need to examine the process of artificial-evolution, more closely. Standing back from, and looking again at the attribute grammar system under examination, it is easily seen - that what actually drives evolution (or learning), is that which is essentially, a variable-length genetic algorithm (GA). Changing the state of the population, from one generation to the next are - selection (*the Darwinian principle of artificial-evolution*); the replacement policy - effectively controlling the introduction of new genetic material into the population of the GA; and the core evolutionary operators of crossover and mutation.

We turn then, to a closer look at the evolutionary process; this, in order to add weight to our current suspicions, and help to ultimately affirm our thesis conclusions. Although the outset research-philosophy of this thesis, is to make as few assumptions as possible - we must note here, that we are following the line of *continued-research*; and as such, the following findings are to be made with respect of the underlying evolutionary setup. That is, in respect of the research which has gone before we choose the configuration for artificial-evolution, thereby recommended [O'Neill and Ryan, 2003a, O'Neill et al., 2003, O'Neill, 2001] (See Section 4.3.2).

As outlined above, the setup is governed by the three key processes driving artificial-evolution. The standard evolutionary setup for GE incorporates selection, as a fitness-proportionate roulette-wheel algorithm; and replacement, as a steady-state algorithm - serving to maintain the incumbent (best-fitness) individual, and ensure the removal of poor genetic-material (in terms of fitness)[1]. In respect of O'Neill et al.'s research findings [O'Neill and Ryan, 2003a, O'Neill, 2001], we choose to maintain these operators (or evolutionary subprocesses) as fixed; we do not change the setup, but view the algorithm for what it is; and attempt to figure out the behavior it exhibits

Turning then to the third of the three key processes - We view the core evolutionary operators of crossover and mutation as the main turning-cogs of the artificial-evolution algorithm, and choose to utilise the operators which research has shown to work well (i.e. bitwise mutation by probability, and variable-length one-point crossover) [O'Neill and Ryan, 2003a] [O'Neill et al., 2003, O'Neill et al., 2001]. Further, we present an argument - reasoning that the variable length one-point crossover operator, is a viable metric of analysis for our study. We look predominantly at crossover; viewing this as paramount in the EA's struggle for balance between exploration and exploitation of the search-space: this is the looking-glass through which our analysis observes the attribute grammar system's behavior.

[1] *Note:* Owing to the absence of wrapping (wrapping turned-off) steady-state replacement also serves to eradicate new offspring which under-specify a solution (*See Section 6.4*)

Owing then, to it's importance, we begin with a brief overview of crossover in GE - discussing some of the important considerations to which one *should* be aware when analysing crossover in GE. Following from this, we outline our crossover-focused approach to system-analysis, and as this analysis progresses, we discuss some experiments describing the effect of the GE-tail to search. Some correlations are drawn with our previous findings and in light of these (and the findings of this chapter), we present our thesis on the observed behavior of the attribute grammar system.

6.2 Crossover & Approach to Analysis

Our continued-research (or extension) of the standard GE algorithm essentially chooses to, *a*) redefine the mapping process with an attribute grammar, and *b*) carry-out an analysis of the resulting genetic-structures being evolved. From the former's constraint-based mapping process, we discovered the insurgence of interspersed-introns (IIs), and consequently, intron-removal strategies served as an initial metric of analysis to present our research findings. Following from these initial findings, and as stated, we choose to maintain the standard GE setup, tried-and-tested in the field of research [O'Neill and Ryan, 2003a], [O'Neill et al., 2003], and use the core-component: *crossover*, as the next metric of analysis. Now, it could be viewed that a study exchanging different types of operator should be carried out, or indeed, that crossover experiments should be carried out in the absence of mutation - our view, is that this is a valid argument; just not the approach best suited to meet the goals of this research.

We wish to examine the behavior of the current system under examination, such that we can learn more about GE, and (more specifically) the

AG(Full) GE system. As such, we do not change the crossover operator; instead viewing it as a metric of behavioral insight into the workings of the un-altered system's behavior.

Guided by the works of O'Neill et al. [O'Neill et al., 2003], Nordin and Banzhaf [Nordin et al., 1997, Nordin et al., 1996], and Gottlieb and Raidl [Raidl and Gottlieb, 1999a, Raidl and Gottlieb, 1999b]; we take the view that crossover can be used as a logical metric of analysis; to access the behavior of the artificial-evolution algorithm. Reference material for the forthcoming presentation can be reduced predominantly, to the work of O'Neill et al. [O'Neill et al., 2003], and Nordin and Banzhaf's 1997 conference paper [Nordin et al., 1997][2].

O'Neill et al.'s study of crossover in the context of GE, shows the variable-length one-point operator - in comparison to, two variants of homologous crossover; two variants of two-point crossover; and a uniform crossover operator. The one-point crossover operator, essentially emerges as the operator which:

1. Provides the best cumulative frequency of success for a quartic-polynomial symbolic regression problem;

2. Provides a non-erratic (as with the homologous crossover's) behavior, when examined over the metric of - the number of newly-evolved individuals which propagate to the next generation; and

3. Shows again, consistent (non-erratic) results, when examined under a metric to measure - the amount of genetic-information being exchanged at each crossover event.

[2]Although these are the main sources, we recommend the interested reader to view all cited material.

The one-point crossover emerges as providing (on average) a 50% exchange of information throughout the runs recorded; that is, approximately half the genetic-material of each parent is exchanged. In this way, and as pointed out, GE with one-point crossover demonstrates operation as a global search algorithm *throughout* an evolutionary run. This point is emphasised in comparison to Poli's findings for GP crossover [Poli and Langdon, 1998], which state that crossover (in a GP context), starts out as a global search operator, but rapidly declines to local-search as the run progresses.

In their study, O'Neill et al.'s metric, of the number of newly-evolved-individuals which propagate to the next generation - although, not providing a model of crossover - does serve to illustrate that at least $\approx$ 35% of the time, crossover is not working as a negative search operator (for the symbolic-regression problem - that is).

Nordin and Banzhaf also study crossover to examine the behavior of their system [Nordin et al., 1997, Nordin et al., 1996]. In these studies, a full model of crossover is presented; where we can see the number of crossovers resulting in a positive, or negative fitness-change (with respect to their parents). Their study is also carried out in light of initial observations of intron propagation, and it is this work in particular, which gave rise to our initial interest in the use of crossover to examine the EA behavior.

Inspired by these two surveys of crossover, for differing GP-based systems, we present a positive model of crossover: choosing to analyse only the positive effects which the crossover operator bears to evolution. From O'Neill et al.'s study we can assume that one-point crossover with the standard GE setup will result predominantly in a non-negative fitness-change ($\approx$ 35% of the time for the symbolic-regression problem): this assumption presumes a translation to the knapsack problem in question. In any case, our approach is to access the number of crossover operations which result in a newly-evolved offspring having greater fitness than a parent. This is similar, but not to be confused with the term used by Gottlieb and Raidl [Raidl and Gottlieb, 1999a, Raidl and Gottlieb, 1999b]; when they describe their *crossover-innovation* (CI) metric: there, *innovation* describing the measure of phenotypic change - or evolved-solution difference, from parent to offspring.

6.2.1 Crossover: A GE-Specific Study

We have presented an argument for the use of crossover as a metric of analysis for the system under evaluation. Before delving further into this analysis, there are some GE-specific considerations for crossover which should be discussed; understanding these, will provide further insight from which one can gain a greater appreciation for the results and analysis.

The crossover operator, and it's role in the effective search of the GE algorithm, has undergone heavy research in recent years [O'Neill et al., 2003, O'Neill et al., 2001, O'Neill and Ryan, 2003a]: one outcome of this research, pointing out that crossover considerations for a GP-system - do not map directly to GE. Primarily, this is due to GE's separation of search and solution space - which affords the variable-length one-point crossover that is the defacto standard for a typical GE run[3]. Furthermore, owing to it's genotype-phenotype mapping process; GE's codons - or tokens of genetic material - are seen to exhibit *intrinsic polymorphism* [O'Neill et al., 2003]. That is - a codon, when applied to the same non-terminal grammar-symbol - will always derive the same production; applied however in a different context (i.e. to a

[3]This one-point crossover is verbatim to any variable-length GA one-point crossover - except to say, that typically GE restricts cross-sites to be between codons.

different non-terminal), it will result in an entirely different phenotypic effect. At the phenotypic level, then, a codon applied to two different terminal-producing productions, will result in a different phenotypic form. Atomically, codons exhibit polymorphism: there are many phenotypic forms depending on the production to which they are applied; and intrinsic-polymorphism is thus the first point of importance to a discussion of crossover in GE.

The diagram in Figure 6.1 shows two GE individuals, and their resulting derivation-sequences. Considering the diagram: we can see, firstly that different codons (*codon4* in each case), result in a different phenotypic effect for the same non terminal[4]. Secondly, it can be seen that the emergence of interspersed-introns (IIs) are particularly context-sensitive: in both individuals *codon4* emerges as an II, only because the attempted production of i_4 and i_2 violate constraints of the problem. That is, they are introns here - only in the context that an i_2 has previously been derived.

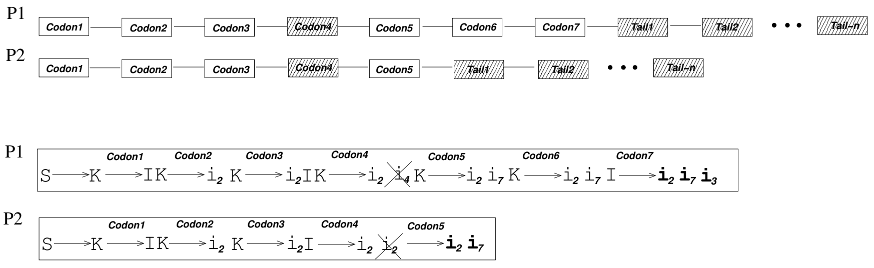

Figure 6.1: Diagram describing two example GE Individuals (for the attribute grammar system). The diagram depicts individuals as codon streams - and as corresponding derivation sequences. Introns are shaded, with "Tail" denoting a residual-intron: i.e. the preceding codon denotes the last codon used to completely specify a solution.

Looking at GE from this current perspective, we note an important discovery. It can be seen that what is an II in one context (*i.e. an i_2 being*

[4]There are various arrangements of outcome - for codons of different size, value, and applied-production; this is relatively irrelevant in this context, so we refer the reader to O'Neill's research for further information [O'Neill and Ryan, 2003a].

previously derived); may not necessarily be an II in another context. In order to understand this, let us turn then to examine a simple crossover operation. A crossover at any point in the codon-stream of a GE individual will result in the remainder of the individual being applied in a different context in the receiving individual. As such we get what is termed a *ripple-crossover* [O'Neill et al., 2003]. To give an example, consider the occasion that the parent cross-site in Figure 6.1, is prior to *codon5* in the first parent; and again prior to *codon4* in the second. In the original context, both codon's emerge as IIs - swapped however, the first will result in a continued legal derivation (II free); with the second emerging as a second II for $P1$, with a *ripple* effect thereafter. Ripple-crossover is thus the second point of importance to a discussion of crossover in GE.

This observation is of particular importance to both - the observed behavior of IIs in pruning systems, where they possibly act as sub-solution protectors (for which we are currently inconclusive); and also, in understanding the danger of transferring observations from a GP-system to GE. In Nordin and Banzhaf's work with a linear GP-system [Nordin et al., 1996, Nordin et al., 1997], they find that IIs and more specifically *explicitly defined* introns (EDIs) are observed to behave as sub-solution, or building block protectors. It can easily be seen, that this is not the case with the AG(Full) GE system. With our system, and as depicted in Figure 6.1, IIs arise through a conflict with a problem constraint. As such, the emergence of a codon as an II is inherently dependent on the context: it is an extremely context-sensitive behavior. Owing to this intrinsic polymorphism of GE codons; crossovers at II-regions do not necessarily result in non-disruption of sub-solutions, as indeed is the case with the GP-system[5].

[5]Introns in the GP-system translate to code-fragments of redundant effect to the overall

This understanding settles our previously inconclusive affirmations of IIs as sub-solution protectors for GE. The improved results of the splicing strategy had contradicted this to be the case, but we still noted the apparent utility of IIs; in the absence of the GE-Tail. In any case, and as can be seen through the example, crossovers at II-regions will more likely result in ripple crossovers (owing to the change of context); and will very rarely, as the case may be, result in non-disruption of sub-solutions.

In the same way, an understanding of this can lead us to better understand the degradation of performance exhibited by systems which prune away the GE-tail. As outlined in the experimental setup (Section 4.3.2), our GE's wrapping operator is turned off. As such, the operation of the resulting mapping process is possibly best described by O'Neill et al. [O'Neill et al., 2003]:

> "If, during the decoding process the generative string runs out of genetic material, the individual is killed (i.e., gets worst fitness)."

Given our understanding of ripple crossover, it can be seen that a cross-site in the latter half of the effective-region will most probably result in, the use of the GE-tail (of residual introns) to grow the newly-emerging ripple-trees to completion [O'Neill et al., 2003]. Pruning of this redundant-store of genetic material will therefore result in an inhibition of ripple-crossovers generative capabilities. Subsequent sections will provide evidence in support of this claim.

Finally, let us consider one more facet of crossover in GE. This is what we define as a *tail-crossover.* A tail-crossover occurs when the cross-site of an individual is selected in the tail. What can be seen, and is depicted in Figure 6.4 of Section 6.3 (which also serves to demonstrate an effective-region

solution: they are redundant in *any* context. Thus, coupled with GP's sub-tree crossover will result in a crossover with no disruption to sub-solutions.

crossover), is that the resulting offspring will be phenotypically indifferent to the parent (See individual *C2* of Figure 6.4). It is important to note here, that this suggestion can be made only from a perspective of crossover alone. One must realise, that subsequent to crossover - a mutation in the effective-region *could* result in what is effectively identical to a ripple-crossover.

Alternatively, such a mutation may be neutral with respect to change - such mutations commonly known as *neutral-mutations* provide a random walk within a fitness plateau, where this process may discover a useful phenotypic change: as such, the term used to describe this behavior is *genetic-drift*; this has been one of the attributed powers of GE's separation of search and solution space (See Section 1.3.1). Owing to a redundant many-to-one genotype-phenotype mapping process - it is speculated that a set of linked neutral-mutations (generally referred to as a *neutral-network*) can result in an escape from local-optima [O'Neill and Ryan, 2003a, O'Neill, 2001]. This is of particular interest to problems of constrained-optimisation - where *saddle-surfaces* frequently arise at the point that the cutting-planes intersect the fitness-landscape (See Section 1.2).

6.3 Analysis by a Model of Crossover

The following graphs of Figure 6.2 and Figure 6.3, present a model of - the operation of the crossover operator, in the described artificial-evolution system. Two graphs for each of the intron-removal strategies, seen thus far, are presented; with the graph on the left showing a model of crossover, and the graph on the right attempting to demonstrate it's subsequent effect to the populations best-fitness achieved. Results presented are an average of 30 independent runs.

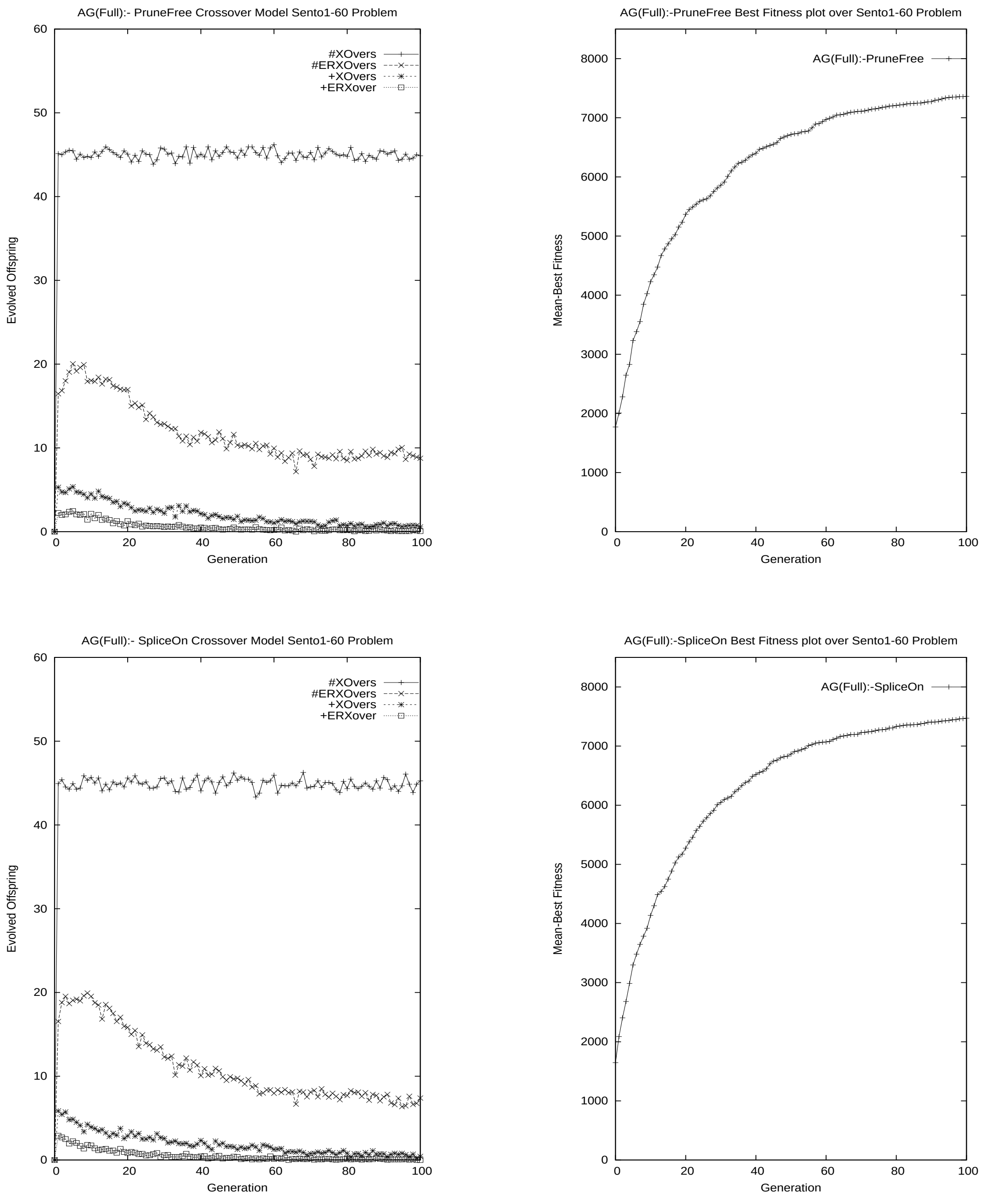

Figure 6.2: Crossover models and associated mean-best fitness plots, for the untouched AG(Full) (*PruneFree*), and AG(Full)-with-splicing (*SpliceOn*), systems. This figure shows the crossover models for systems where the GE-tail remains. Results presented, are averaged over 30 independent runs.

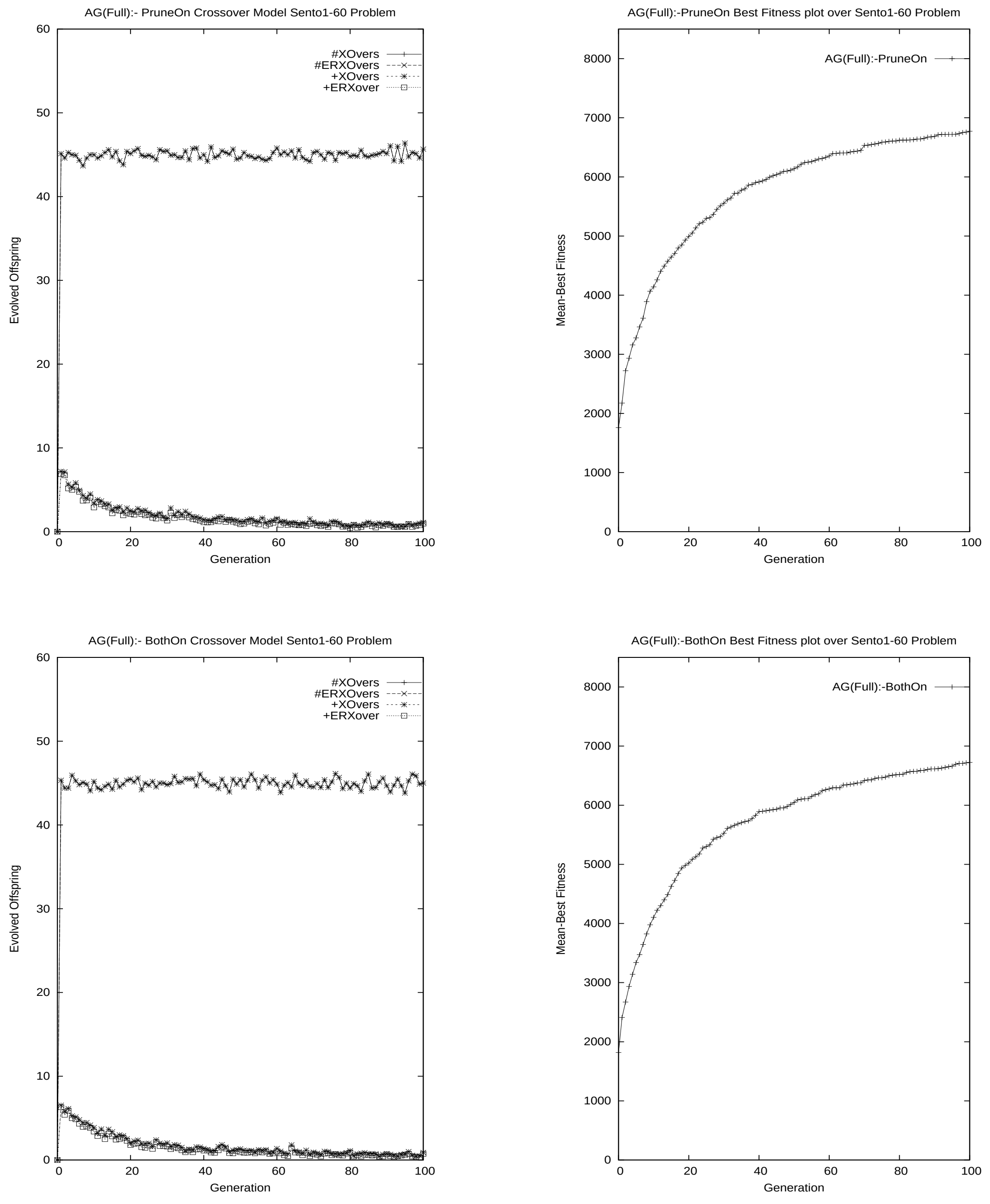

Figure 6.3: Crossover models and associated mean-best fitness plots, for the AG(Full-with-pruning (*PruneOn*), and AG(Full)-with-splice+prune (*BothOn*), systems. This figure shows the crossover models for systems where the GE-tail has been removed. Results presented are averaged over 30 independent runs.

The Figures are displayed, such that only the intron-removal strategies seen thus far, are presented and as will become evident, we have chosen to group these systems together according to their allowance or exclusion of individuals with a GE-tail, to evolve. Figure 6.2 thus shows the untouched AG(Full) system, coupled with the system using the *splicing* operator (*below it*). Similarly, Figure 6.3, shows the *pruning* system, coupled with the *splice+prune* system (*below it*); again here, but in direct contrast to the former - these systems are grouped according to the *absence* of the GE-tail. Plots are shown only for the first 100 generations, as each metric of analysis is seen to plateau thereafter; in this way, one can view the changing - and therefore, more important system behavior, at a closer perspective. In order to unify the behavior exhibited by crossover, we examined all problem-instances tested, and highlighted the problem instance of *Sento1-60* as being most representative of the overall behavior. No other instances showed a deviation from the behavior displayed forthwith.

The model of crossover presented, shows plots for four independent metrics of analysis described as follows:

#XOvers The number of crossovers metric: shows the number of newly evolved offspring (conceived by crossover), that occur at each generation. For example, as described in the experimental setup, we choose a population size of $\mu = 50$, and as such, this plot effectively shows $\mu * P_c$; as occurring for each steady-state generation - where P_c is the probability of crossover; i.e. 0.9 (Section 4.3.2).

#ERXOvers The number of newly-evolved individuals, conceived as a result of a cross-site in the effective-region. Figure 6.4, serves to clarify this definition: $P1$ and $P2$ depict parents, with $C1$ and $C2$ their respective offspring - the figure shows that $C2$ fails to register as an

ERXOver as it was conceived with a cross-site in the GE-tail, and is thus phenotypically identical with it's first contributing parent ($P2$).

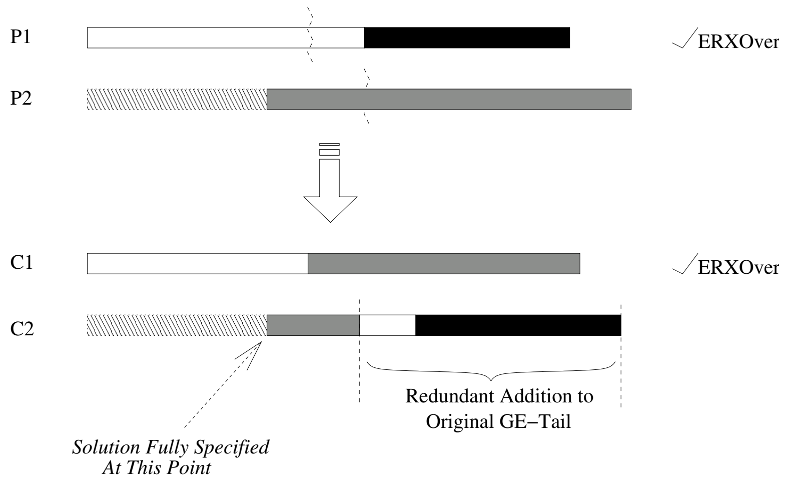
P1
P2
√ERXOver
C1
C2
√ERXOver
Redundant Addition to Original GE–Tail
Solution Fully Specified At This Point

Figure 6.4: Diagram defining an effective-region crossover (ERXOver): specific to GE's one-point crossover - a child becoming an ERXOver, only, if a parent cross-site is selected within the effective region.

+XOvers The number of newly evolved individuals contributing to *fitness-innovation.* As depicted in Table 6.1, we define this *fitness-innovation*, according to an offspring's parent-relative fitness: a fitness-innovation has occurred when a child's fitness exceeds it's parents' best (i.e. parent upper-bound) - or exceeds the parents' worst (i.e. lower-bound). A crossover event produces 2 offspring, and as such we mark the respective child's parent-relative fitness with an x in the corresponding row (*rows depicting parent-relative fitness brackets - above, in-between, below*).

+ERXOvers This metric captures the newly-evolved individuals which are fitness-innovative *and* have been conceived via an ERXOver. This plot attempts to illustrate the benefit of ERXOvers to the fitness-innovation.

As depicted in Figure 6.4 and eluded-to in the accompanying text - offspring resulting from a tail-crossover (tail-crossover offspring), can never produce

		Innovation Score	2	2	1	1	1	0
		C_1	x	xx		x		
Parent-upper								
		C_2	x		xx		x	
Parent-lower								
		C_3				x	x	xx

Table 6.1: The *fitness-innovation* metric: $Parent-upper$ and $Parent-lower$ show the upper and lower bounds covered by both parents' fitness; C_1, C_2 and C_3, denote a possible offspring's parent-relative fitness position; with a single x showing only 1 evolved offspring in this fitness-range, and an xx denoting both. $InnovationScore$ shows the weighted contribution to the fitness-innovation for each possibility.

a fitness-innovation or an innovative phenotypic-effect. As such, all fitness-innovation should result from ERXOver's. It is important to note, however - that a tail-crossover *can* result in a fitness-innovation; in the rare occasion that it is followed by a mutation in the effective-region. (As such and as will be seen, we do report innovative tail-crossovers; although very-few).

Looking then, for the first time - at the crossover-model graphs in Figure 6.2, we can clearly see that the #ERXovers is quite low with respect to the total (i.e. #XOvers). For the untouched AG(Full) system, we begin with $\approx 44\%$ of crossovers being ERXOvers (by generation 5), and this is halved to $\approx 22\%$ by generation [40-50]; the plots plateau thereafter. In general, this shows that less than half of the newly-evolved individuals involve a parent cross-site in the effective-region; and as evolution progresses, this measure can be seen to further reduce by half again. This serves to illustrate that crossover is predominantly operating via *tail-crossovers* (the space above the plot for #ERXOvers shows this). Through the eyes of crossover alone, and as depicted in Figure 6.4, these will always result in zero innovation (newly emerging phenotypic information), and are always neutral with respect to fitness increase. As previously stated - the exception to the case being, a subsequent innovative-mutation in the effective-region.

Looking at Figure 6.2 from a more comparative perspective, we see confirmation of our suspicions that the splicing strategy increases compression (splicing decreases the #ERXOvers to $\approx 17\%$ whereas the straight AG(Full) system remains at $\approx 22\%$): more specifically, we observe this to occur in the latter generations - presumably increasing the number of neutral crossovers, and giving support to the Nordin and Banzhaf argument; that compression of the effective-length acts as a form of self adaptation to the probability of crossover.

Looking more closely at the graphs, and with emphasis again on our model's goal (to analyse the positive effects of crossover), we can see that initially - there is approximately 13% fitness-innovation (+XOver), with approximately 6% (half) occurring in the effective-region (+ERXOvers). The gap between $+XOver$ and $+ERXOver$, therefore shows, the number of innovative tail-crossovers; as pointed out, this signals the number of fitness-innovations occurring as a result of positive mutations, subsequent to the crossover operation. In essence, what is happening here - is that, mutations occur after a tail-crossover and it is these mutations which result in a fitness-innovation.

One should now continue to view the model in this way: the space above the #ERXOvers represents the number of tail crossovers and the space between #ERXOvers to $+XOvers$, represents effective-region crossovers which show no record of fitness-innovation. The space from +XOvers to +ERXOvers represents the number of innovative tail-crossovers (by mutation) and finally the space below the $+ERXOvers$; illustrates just that. Viewed therefore, from top to bottom - we get, tail-crossover offspring; effective-region crossover offspring; innovative tail-crossover offspring, and innovative effective-region crossover offspring, respectively: innovative-crossovers, (it can be seen) account for only a very small percentage of the overall.

In respect of this explanation, and looking again at Figure 6.2, we can see that after approximately 100 generations, the crossovers are predominantly - if not all - of neutral or negative effect, so no longer are sub-solutions combining; what we see is that search is effectively reduced to random exploration by mutation, operating at a very low probability of application. The gap between *+XOver* and *+ERXOver* further demonstrates that as evolution progresses - mutation also has little or no effect: the population has converged, and search is effectively discontinued.

The included mean-best fitness plots (also averaged over 30 independent runs), serve to reflect the effects of either strategy on the populations mean-best fitness. As with the earlier presented results, it can be seen that the splicing strategy achieves a higher fitness plot - particularly in the first 50 generations; this can be seen in the more pronounced belly of the plot. Furthermore, the plot shows the greater end-fitness achieved which is not surprising owing to the previous results for a 40%-10% success rate in favour of the splicing stragegy (Section 5.4.2, Table 5.2). On analysis of the crossover model - we can only assume (at this point), that the slightly higher plot for splicing's #ERXOvers and +XOvers alike - in the early generations - (which incidentally are maintained for slightly longer), coupled with the already-noted increase in compression from generation 50 on; can be the observed cause for the improvement in performance.

Presumably, owing to the increase in duration of #ERXOvers, this creates a population with more effective-diversity in the early phase of evolution. This could provide a greater genetic-store for exploration. Further, the compression exhibited by splicing (*i.e. reducing ERXOvers to* $\approx 17\%$) minimises disruption of this diversity and results in a better balance between exploration and exploitation of the populations pool of genetic material. Future analysis will seek to confirm these suspicions.

As previously stated, Figure 6.3 shows graphs plotted for the two systems employing the pruning operator: *pruning* and *prune+splice*. For these, and subsequent graphs of systems which do not permit the presence of the GE-tail; we must take a slightly altered - if not more simplistic - view of the crossover model. In these systems, tail-crossover offspring cannot occur, and so all crossover events occur within the effective-region only. As such, #ERXOvers and #XOvers show the same plot. Similarly, plots for the metrics of +XOvers (crossover innovations) and +ERXOvers (innovations resulting from ERXOvers) match; as all newly-evolved individuals, must now result from an effective-region crossover.

Effectively in these graphs, we get - one plot showing the number of offspring evolved by crossover (always ERXOvers now), and one plot showing the number of innovative-offspring resulting thereof. As there is no significant difference between the fitness-innovation occurring in these and the non-pruned systems; (crossover is resulting in much the same fitness-innovation for pruned and non-pruned systems) - we turn to an examination of the GE-tail (the difference) in an attempt to gain further insight.

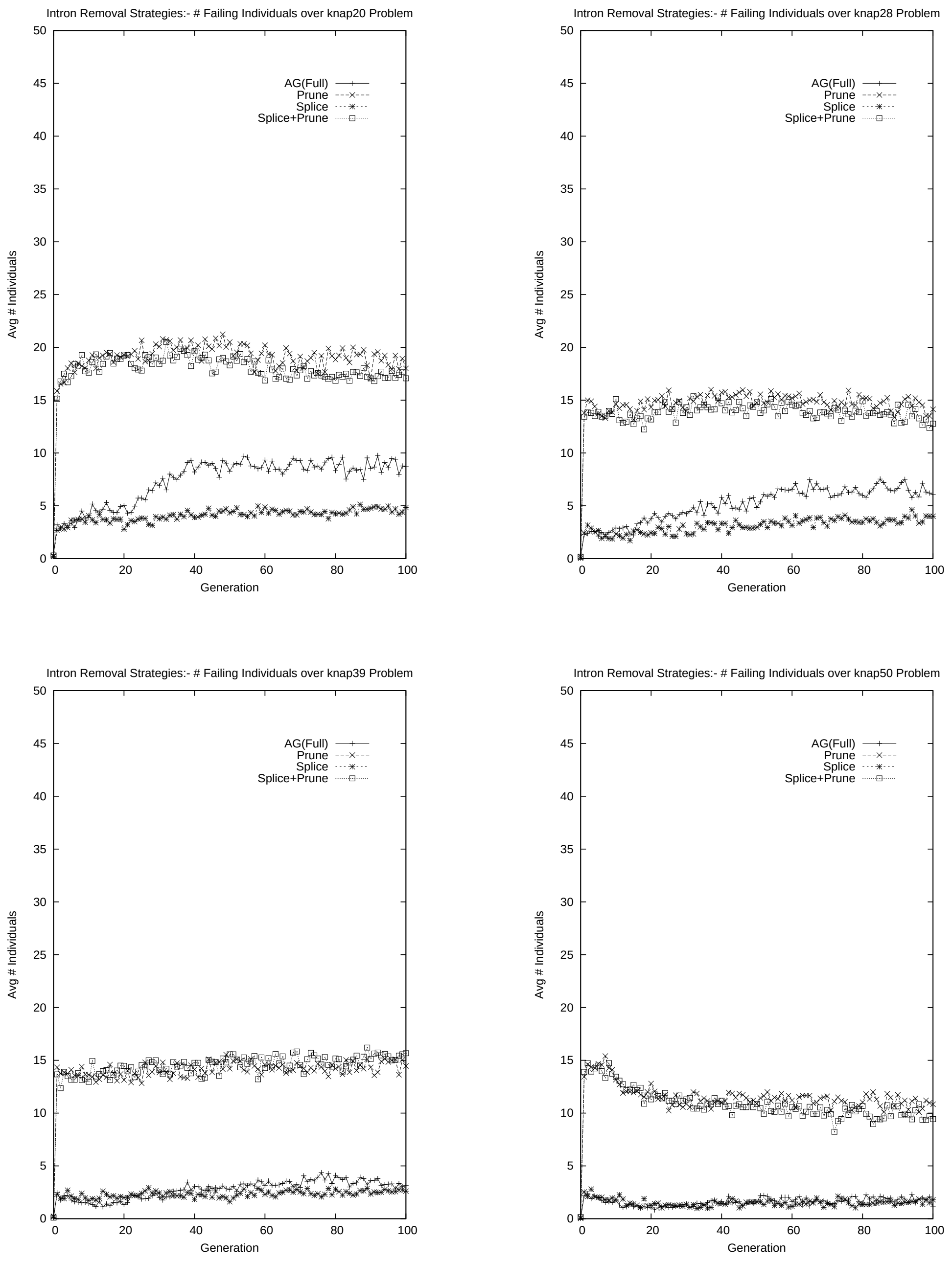

Figure 6.5: The effect of pruning on underspecified-offspring (Failing Individuals). Plotting the number of individuals who fail for *knap* problems: individuals which do not have enough codons to completely specify a solution. GE's wrapping operator is turned off, and all results presented are an average of 30 independent runs.

6.4 The *Tail* of Ripple-Crossover

The purpose of this section is to follow the lead of the analysis which has gone before, and attempt to evaluate the effect that the presence of a GE-tail has on the evolving systems, in question. The results to-date have shown pruning to degrade performance and we now suspect that this is the case - owing to the absence of the GE-tail. In order to assess this, we have developed a set of experiments which restrict the crossover operator: the purpose of this being, to allow us to emulate the exact behavior of the pruning system - *only* where a tail of residual-introns remain. In this way, it is hoped that we can assess whether the absence of a GE-tail in pruning systems, is indeed a hindrance to the generative capabilities of ripple crossover.

Figures 6.5 and 6.6, show graphs for a metric measuring the number of offspring failing to result in a complete derivation (*underspecified offspring*). A clear divide between the plots for different intron-removal techniques can be seen: systems which involve pruning show-up to produce approximately three times the number of underspecified offspring. (The graphs of Figures 6.5 and 6.6 are shown for all eight problem instances; behavior is consistent throughout, and again results presented are an average of 30 independent runs).

It is suspected that this behavior is due to the absence of the residual store of genetic material found in the GE-tail (for pruned systems); therein, we suspect that the previously described ripple-crossover cannot complete it's ripple-trees and an underspecified offspring results. It is hoped that the experimentation of this section, will serve to affirm our current two-part thesis: *a*) that the removal of the GE-tail is indeed an inhibition on ripple crossover; and also, *b*) that the *compression* of the effective-region - affords a form of protection against the disruptive-effects exhibited by crossover.

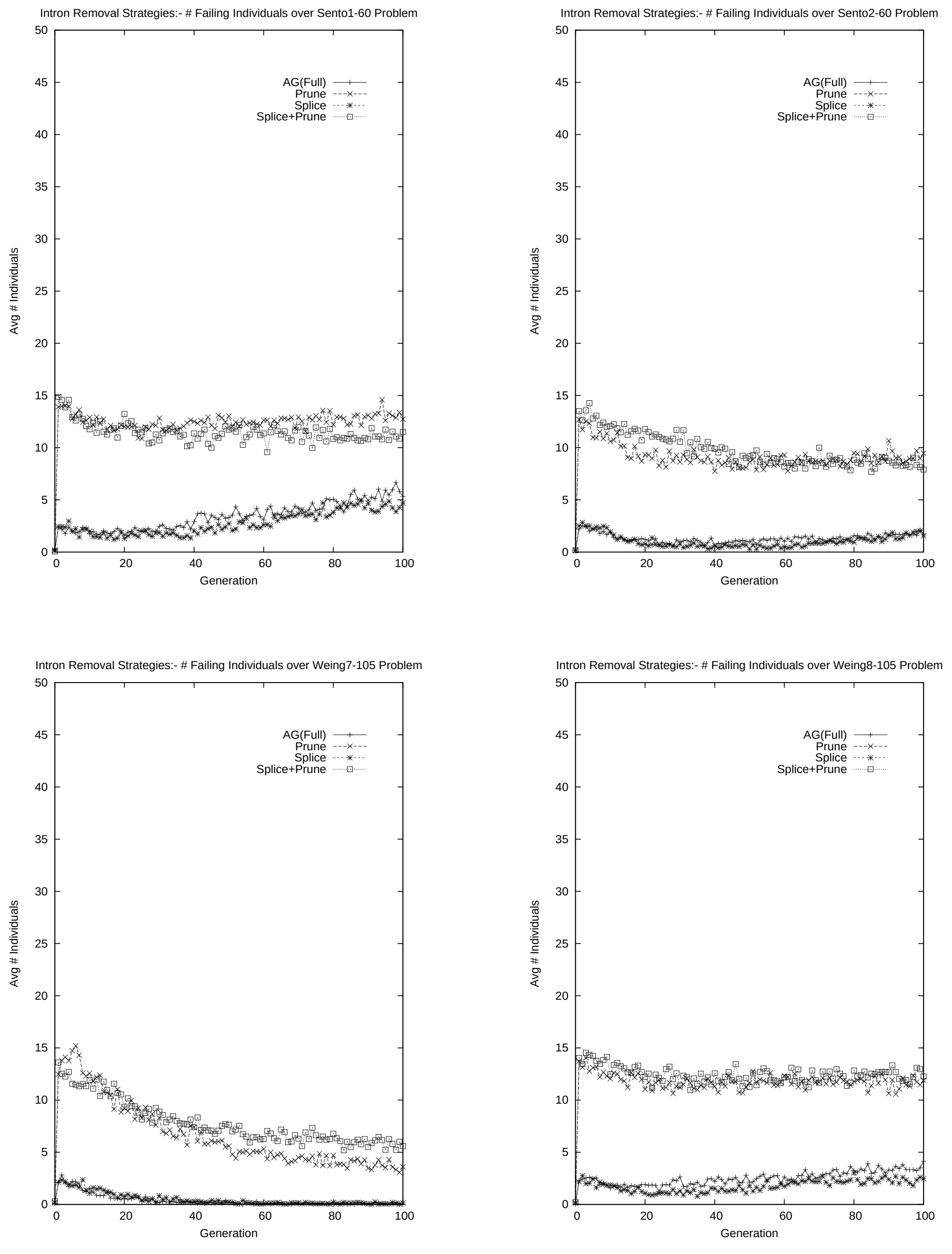

Figure 6.6: The effect of pruning on underspecified-offspring (Failing Individuals). Plotting the number of individuals who fail for *Sento & Weing* problems: individuals which do not have enough codons to completely specify a solution. GE's wrapping operator is turned off, and all results presented are an average of 30 independent runs.

6.4.1 Experiments with a Restrictive Crossover

In an attempt to capture the utility, if any, of the GE-tail to ripple crossover, we choose to experiment with a restriction on the standard GE one-point crossover. As depicted in Figure 6.7, this amendment to crossover regulates, that cross-sites be chosen within the bounds of the effective-region (emulating the pruning strategy's lack of tail). With this strategy however, a GE-tail exists; capable of feeding the growth of expansive ripple-trees where necessary.

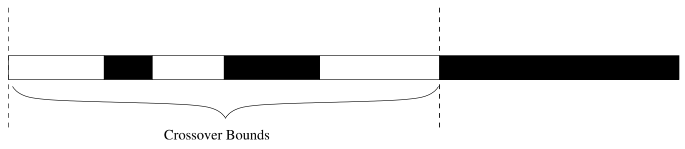

Figure 6.7: Diagram illustrating the restriction enforced by the restrictive crossover operator.

Results The restrictive-crossover setup is compared against the pruning strategy. Table 6.2 presents the core findings of the study: restrictive-crossover is found to significantly outperform it's pruning counterpart. This direct comparison implies, that the presence of a GE-tail is beneficial to search. Table 6.3 shows the results relative to the untouched AG(Full) system, with Table 6.4 showing that restrictive-crossover, although a significant improvement over the AG(Full) and pruning systems; does not measure up to the splicing strategy.

As the difference between the two systems being compared is the presence of the GE-tail - we can strongly believe that the first part of our thesis is confirmed: removal of the GE-tail (by pruning) is inhibiting the generative-power of ripple-crossover.

Problem	knap15	knap20	knap28	knap39	knap50
Pruning	66.66%	60%	10%	10%	6.66%
RXOver	86.66%	93.33%	43.33%	23.33%	0%
Problem	Sento1-60	Sento2-60	Weing7-105	Weing8-105	
Pruning	6.66%	0%	0%	6.66%	
RXOver	13.33%	0%	0%	10%	

Table 6.2: Core Results for restrictive crossover as a simulation of the pruning strategy; but where a GE-tail exists. Restrictive crossover is compared to pruning, showing a substantial improvement. Results presented are an average of 30 independent runs.

Problem	knap15	knap20	knap28	knap39	knap50
AG(Full)	83.33%	76.66%	40%	36.66%	3.33%
RXOver	86.66%	93.33%	43.33%	23.33%	0%
Problem	Sento1-60	Sento2-60	Weing7-105	Weing8-105	
AG(Full)	10%	3.33%	0%	6.66%	
RXOver	13.33%	0%	0%	10%	

Table 6.3: Results for restrictive crossover against the standard AG(Full) system: a general increase in improvement is observed, particularly over the more difficult problems. Results presented are an average of 30 independent runs.

Problem	knap15	knap20	knap28	knap39	knap50
Splicing	86.66%	80%	30%	33.33%	3.33%
RXOver	86.66%	93.33%	43.33%	23.33%	0%
Problem	Sento1-60	Sento2-60	Weing7-105	Weing8-105	
Splicing	40%	13.33%	0%	33.33%	
RXOver	13.33%	0%	0%	10%	

Table 6.4: Results for restrictive crossover against the best-performing splicing strategy: a improvement is observed over the easy problems, but a general degradation is observed over all difficult problems. Results presented are an average of 30 independent runs.

6.4.2 Splicing & Restrictive Crossover

This further experiment was developed - essentially, to examine the effect of interspersed-introns (IIs) in the previously described restrictive-crossover setup (*See again, Figure 6.7*). Figure 6.8 shows the effect-to-evolution of the combined splicing and restrictive-crossover system proposed.

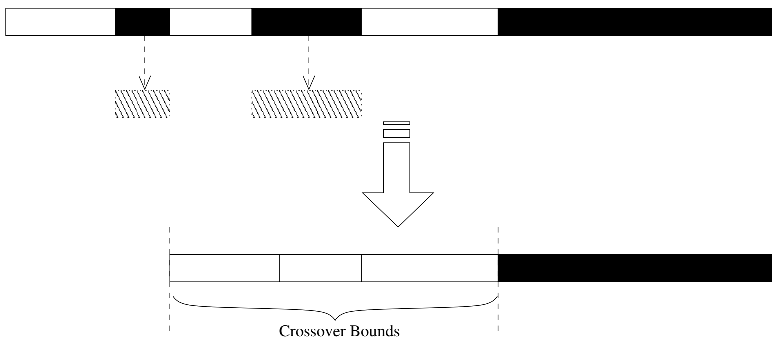

Figure 6.8: Diagram illustrating the effect to the evolving genome, of the splice and restrictive crossover combination.

Results It can be seen that there are two ways one can attempt to analyse the results for this setup. Firstly , as *restrictive-crossover without IIs* - Table 6.5 provides an initial examination, showing that the removal of IIs (increase in compression) does indeed demonstrate an improvement in results. Alternatively, we could interpret the results as - *splicing-alone, with crossover restricted to the bounds of the effective-region* (i.e. compression by splicing, but no tail-crossovers). This perspective allows us to question the benefit of tail-crossovers in the splicing strategy. Table 6.6 shows that the absence of tail-crossover in $Splice + RXOver$ result in a general degradation in performance; with respect to splicing-alone where tail-crossovers are permitted.

In order to confirm the second-part of our two-part thesis then, the first perspective - i.e. the absence of IIs (and furthered compression of the

Problem	knap15	knap20	knap28	knap39	knap50
RXOver	86.66%	93.33%	43.33%	23.33%	0%
Splice+RXOver	86.66%	90%	13.33%	16.66%	3.33%
Problem	Sento1-60	Sento2-60	Weing7-105	Weing8-105	
RXOver	13.33%	0%	0%	10%	
Splice+RXOver	13.33%	6.66%	0%	26.66%	

Table 6.5: Splice+RXOver Results: effect of removing IIs from restrictive crossover. Removal of IIs is seen to show an improvement in performance for more difficult problems. Results presented are and average of 30 independent runs.

effective-region thereof) - would have us expect to see an improvement, and we do; similarly, tail-crossovers (we suspect) emerge as a protection-mechanism to the destructive effects of crossover: as such we can confirm that their improvement (in *Splicing*) over *Splice* + *RXOver*, suggests that they are of merit, but as yet we cannot conclusively confirm our hypothesis.

In a more global context, Table 6.7 shows how the *Splice*+*RXOver* strategy measures up to the other two. We can see that although an improvement over the standard AG(Full) algorithm and RXOver alike; it doesn't measure-up to the *Splicing* strategy alone.

As such, we are left with the following suspicions: that either tail-crossovers are innovative with respect to the exploration/exploitation balance of search; or, that their existence affords a greater number of neutral crossovers. The latter case drawing from the previous assertion - that bloat, with an increased probability of tail-crossovers, can be interpreted as a self-adaptive behavior in response to disruptive crossovers.

Problem	knap15	knap20	knap28	knap39	knap50
Splicing	86.66%	80%	30%	33.33%	3.33%
Splice+RXOver	86.66%	90%	13.33%	16.66%	3.33%
Problem	Sento1-60	Sento2-60	Weing7-105	Weing8-105	
Splicing	40%	13.33%	0%	33.33%	
Splice+RXOver	13.33%	6.66%	0%	26.66%	

Table 6.6: Comparing Splice+RXOver to Splicing: effect of restricting crossover to the splicing setup: tail-crossovers in splicing are responsible for the improvement. Results presented are and average of 30 independent runs.

Problem	knap15	knap20	knap28	knap39	knap50
AG(Full)	83.33%	76.66%	40%	36.66%	3.33%
Splicing	86.66%	80%	30%	**33.33%**	3.33%
RXOver	86.66%	**93.33%**	**43.33%**	23.33%	0%
Splice+RXOver	86.66%	90%	13.33%	16.66%	3.33%
Problem	Sento1-60	Sento2-60	Weing7-105	Weing8-105	
AG(Full)	10%	3.33%	0%	6.66%	
Splicing	**40%**	**13.33%**	0%	**33.33%**	
RXOver	13.33%	0%	0%	10%	
Splice+RXOver	13.33%	6.66%	0%	26.66%	

Table 6.7: Splice+RXOver Results: a comparison to all related systems. The table shows the incumbent (best-performing) strategy for each problem-instance. Results presented are an average of 30 independent runs.

6.4.3 Analysis of Restrictive Crossover

Up to this point, the analysis of this chapter has essentially utilised two metrics of examination to observe the behavior of intron-removal in the AG(Full) system. The initial intron-removal techniques have been studied by *a*) a model of the behavior of crossover, and *b*) by examining their respective effects on the number of newly-evolved individuals, failing to grow ripple-trees to completion.

In order to maintain consistency and complete the analysis, this section presents a corresponding study for the restrictive-crossover experiments. Figure 6.9 shows the crossover models for the two respective systems; again here these models can be viewed in the more simplistic way - as having one plot for the number of offspring evolved by crossover (always ERXOver here, owing to the restriction), and one plot for the number of innovative-offspring resulting thereof (See Section 6.3).

Examining the graphs for the model of *restrictive* crossover (i.e. presence of GE-tail) in Figure 6.9, and referring back to that of the pruning technique (Figure 6.3) - it can be seen that particularly from generation [0-20]; restrictive-crossover shows a higher plot for the number of innovative-offspring. This is further, reflected in the corresponding graphs for mean-best fitness - after $\approx$ 20 generations, the restrictive-crossover has achieved a fitness-difference of, $\approx$ 1000 fitness units[6], to that of pruning. It is suspected that the observed rise in innovative-offspring, is a direct result of the restoration of ripple-crossover - through the reinstatement of the GE-tail, and thus ripple-crossover's store of residual genetic-information.

We have seen that splicing demonstrates an improved compression of the effective-region, and further - that forbidding tail-crossovers (*Splice* +

[6]Raw-fitness of the knapsack-profit for the evolved solution is shown.

RXOver), disimproves performance. Looking at the comparative crossover models of Figure 6.9 for *Splice* + *RXOver*, with plain splicing in Figure 6.2; it can be seen that the addition of RXOver provides an initial *increase in fitness-innovation* - but a stronger plateau thereafter; again after ≈ 20 generations. This can be seen in both, the crossover models and accompanying mean-best fitness plots. We suspect that the following explanation describes this behavior.

As solution sizes increase, they become closer to a weight-constraint violation. Restricting crossover to the effective-region will result in more ripple crossovers - these facts remain plain-to-see. We believe that early in the evolutionary process - this increase in ripple-crossovers (for the restrictive-crossover setup) improves exploration (hence the higher initial innovation and fitness increase); later in evolution however, as solution sizes increase (nearing optimum), ripple-crossover operates as a disruptive influence to search - presumably causing large changes at a phenotypic level. As such, it is here that plain splicing outperforms *Splice* + *RXOver*. As evolution progresses - compression increases, and the evolving population effectively self-adapts to the disruption being caused by the ripple-effect to these good solutions: self adaptation occurring through residual-intron propagation. The resulting increase in the probability of tail-crossovers, effectively renders crossover in-operational. We can thus confirm the second-part of our thesis to the strength this suggestive-evidence.

This claim of self adaptation via residual-intron propagation remains consistent with our earlier findings: where it was observed that, for the *knap20* problem-instance, a stagnation in tail-growth resulted - as a consequence of convergence to the optimal solution (Section 5.3.1). This further supports the notion that selection is acting in favour of highly-fit individuals with a high tail-proportion. It appears that residual-intron propagation results in direct response to the mid-evolutionary change-of-role for ripple-crossover: from constructive to disruptive. These findings, together with the observed exhibition of a continually reducing innovation in the crossover models, oblige us to observe that - for these problems - crossover is effectively cancelled-out as a search operator after approximately 20 generations. That is, referring to the crossover-models for all systems examined - it can be seen that - each exhibits a short, successful, explorative-phase (up to ≈ 20 generations); before a large dip results thereafter; we attribute this dip to a disruptive change-of-role exhibited by ripple-crossover as solutions near the optimal solution. It appears that O'Neill et al.'s findings for standard GE over the symbolic-regression and Santa-Fe ant, trail problems - do not translate to the attribute grammar system over constrained knapsack problems: the power of ripple-crossover's global-search is effectively nullified by this self-adaptive behavior, protecting against disruption of solutions.

Furthermore, the graphs of Figure 6.10 and 6.11 serve to show a direct comparison of pruning and restrictive-crossover; in terms of, the number of underspecified offspring. Pruning emerges once again as comparatively detrimental to search with approximately twice the number of offspring having inability to specify a solution. This again, furthering the evidence to show that ripple-crossover relies on the GE-tail to fulfill it's ripple-trees.

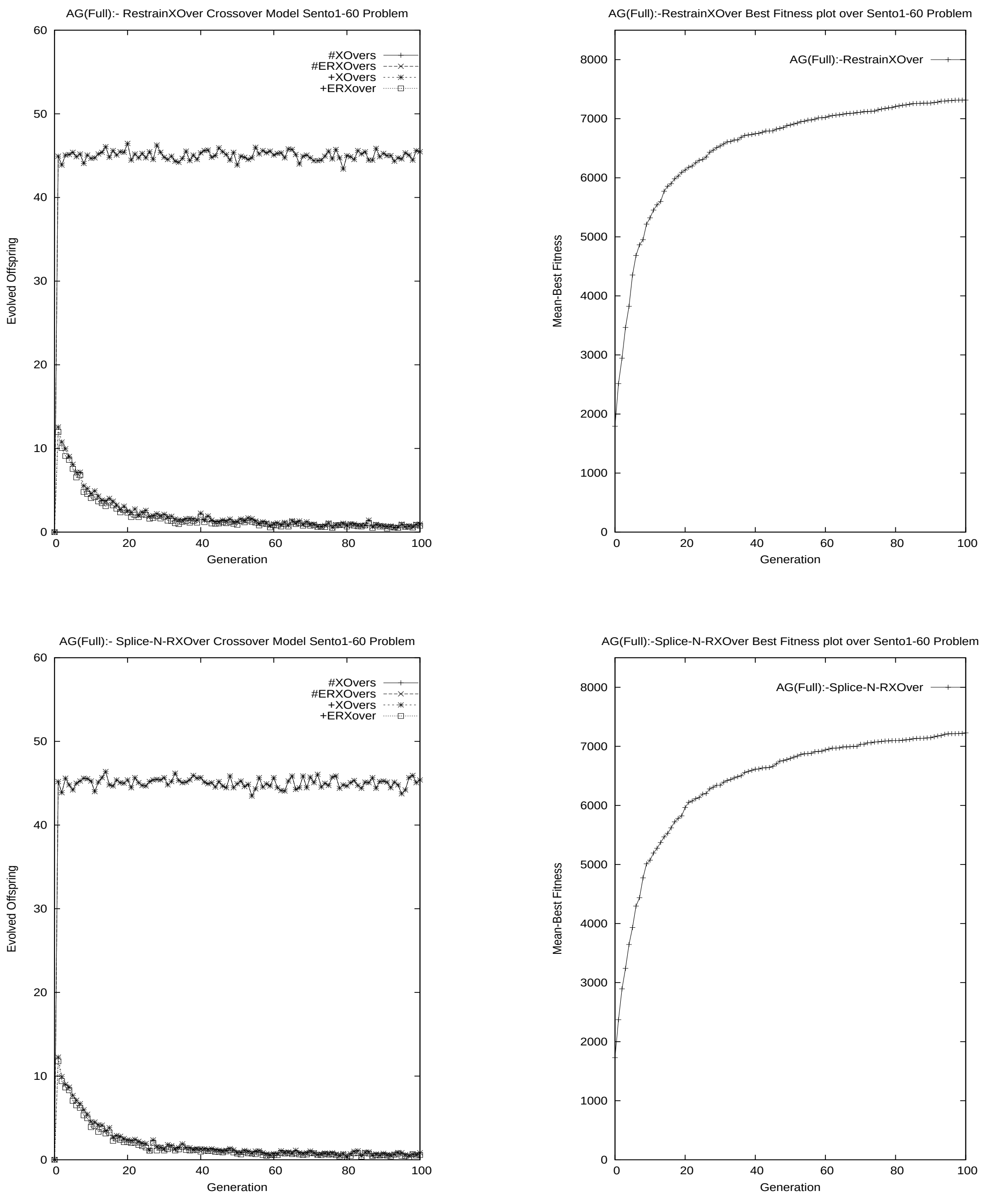

Figure 6.9: Crossover models and associated mean-best fitness plots, for the AG(Full)-with-restrictive crossover (*RestrainXover*), and AG(Full)-with-splice+restrictive crossover (*Splice-N-RXOver*), systems. Results presented are averaged over 30 independent runs.

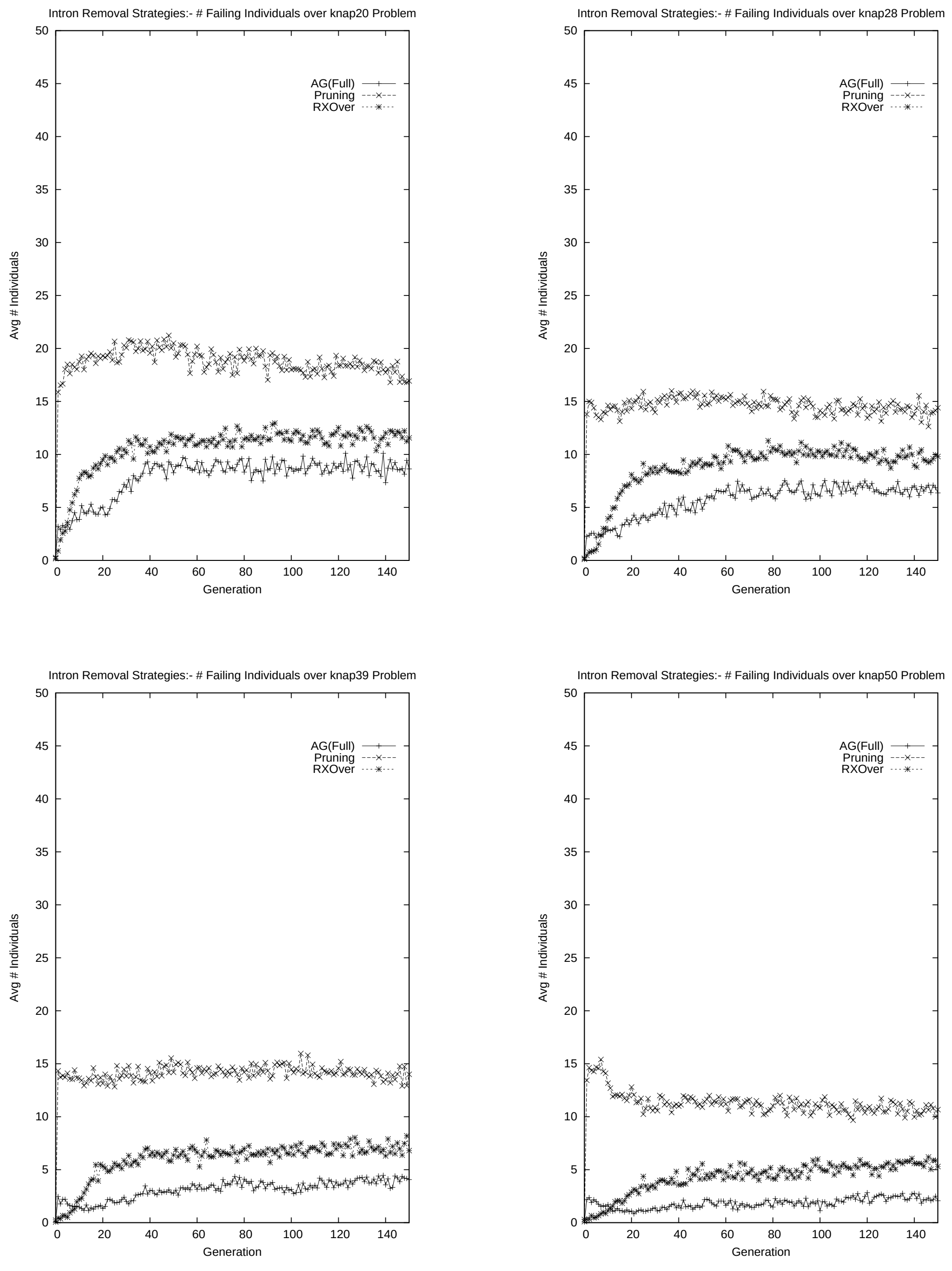

Figure 6.10: The effect of pruning on underspecified-offspring (Failing Individuals). Plotting the number of individuals who fail for *knap* problems: individuals which do not have enough codons to completely specify a solution. GE's wrapping operator is turned off, and all results presented are an average of 30 independent runs.

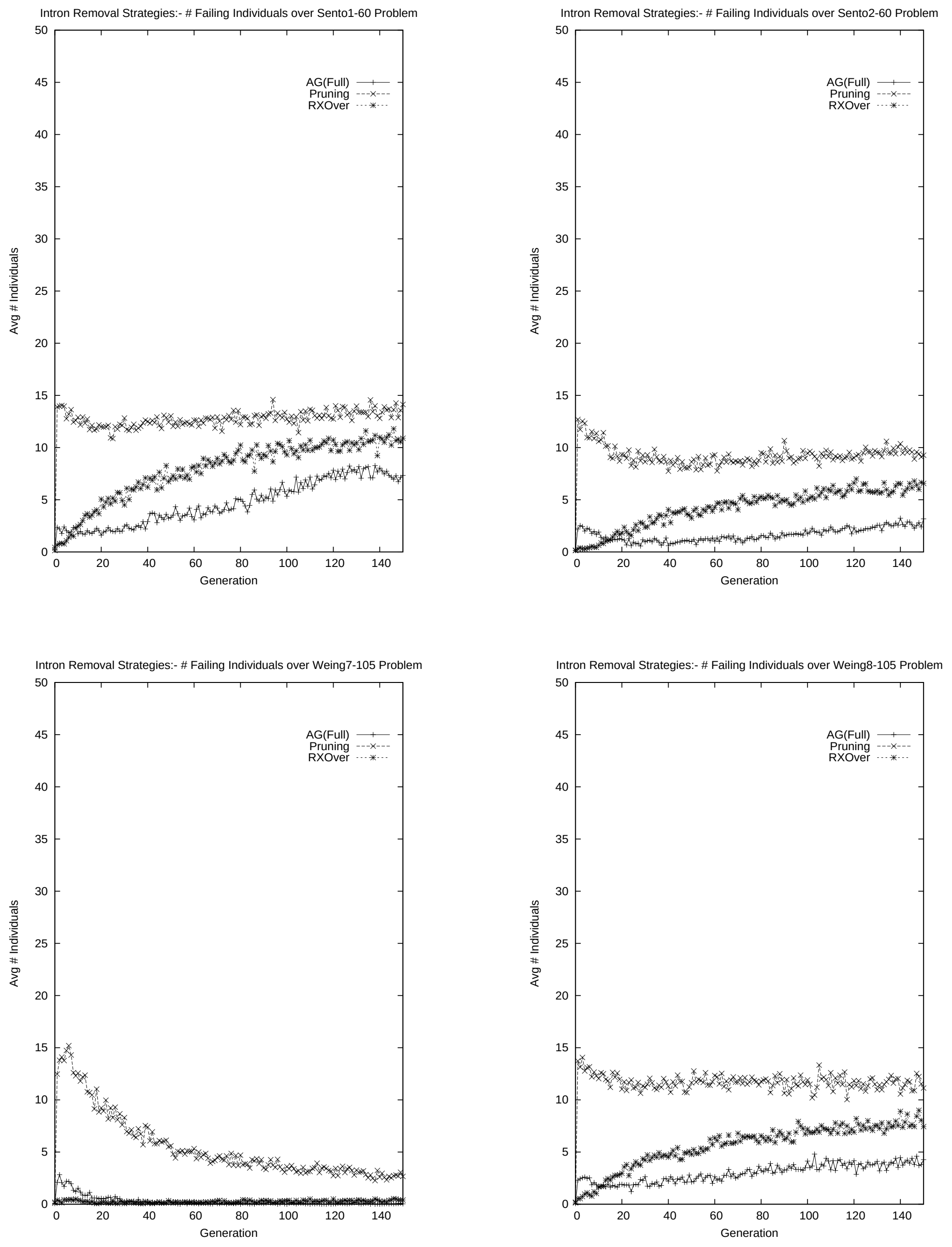

Figure 6.11: The effect of pruning on underspecified-offspring (Failing Individuals). Plotting the number of individuals who fail for *Sento & Weing* problems: individuals which do not have enough codons to completely specify a solution. GE's wrapping operator is turned off, and all results presented are an average of 30 independent runs.

6.5 Summary and Conclusion

In this chapter, we have identified the attribute grammar system as an extension to the standard GE. Standing back from the algorithm, we realised that we are essentially dealing with a search algorithm: in this case that search algorithm is a variable-length GA. As such, we identify the three key sub-processes which underpin the GA (and, in-fact most EAs): selection, replacement, and the core operators of - crossover and mutation.

An argument is presented for the use of the crossover-operator as an analysis-tool (akin to a looking-glass over the black-box operation of the GA) - we do not choose to augment the natural behavior of the attribute grammar systems in any way - but instead, build a model of the behavior of crossover. The defined model, focused on examining crossover's benefit to search; defining *fitness-innovation* as a metric to measure the utility of the newly-evolved offspring with respect to improvements in fitness.

Prior to the observation of results, we present a brief study of GE-specific crossover considerations - pointing out the behavior of *intrinsic-polymorphism* (exhibited by GE's codons) and the consequent phenomena of *ripple-crossover*. An understanding of these - GE behavioral traits - led us to believe that contrary to Nordin and Banzhaf's linear GP-system - interspersed-introns (IIs) in GE are very-much less likely to act in a role of sub-solution protector [Nordin et al., 1997]; the attribute grammar system's II emergence was seen to be a highly context-sensitive issue.

Analysis of the systems by the model of crossover showed interesting results - with crossover showing-up to be *a*) of little benefit to fitness-innovation, and *b*) predominantly producing offspring by means of tail-crossovers. Mutations within the effective-region of these such offspring, were observed to contribute to fitness-innovation - this observation holding true for only a very small percentage of the evolved offspring. For systems eliminating the GE-tail by pruning - the crossover model became more simplistic; in this case, showing just the number of evolved crossover-offspring; and the fitness-innovative offspring thereof.

Results from analysis of the model of crossover gave the impetus for a set of experiments describing a restricted-crossover (working now within the bounds of the effective-region). Initial results for these experiments, served as confirmation that the absence of the GE-tail in pruning gave rise to an inhibition of the ripple-effect of GE's crossover. Further experiments into the removal of IIs in this - restricted-crossover environment - lead to confirm that tail-crossovers exist as a self-adaptive response to the destruction of sub-solutions. These findings correlate to Nordin and Banzhaf's GP discovery [Nordin et al., 1997].

Finally and most conclusively, the analysis of this chapter has led us to an understanding of the attribute grammar system's behavior. We view crossover, as paramount to the observed behavior: the results and analysis presented provide evidence to suggest that ripple-crossover is the key player in the behavior of the system. We believe that - early in the evolutionary process, ripple-crossover causes a diverse search of the solutions in the search-space; but as evolution progresses, and solutions approach the maximum allowable length - ripple-crossover acts in a destructive role. This, we believe is the reason that splicing-alone exhibits a better performance than splicing with restrictive crossover: tail-crossovers in the former allowing a form of self-adaptation to the mid-evolutionary disruption caused by crossover. Unfortunately, this has the effect of crippling the core operators of search, and consequently we observe that search is effectively discontinued.

Chapter 7

Phenotypic-Duplicate Elimination

7.1 Introduction

Results and analysis from the previous chapter led us to propose that a mid-evolutionary *change-of-role* is observed for ripple-crossover in the attribute grammar system. This change-of-role seems to effect a change from an early constructive-phase (exploration), to an observed dip in fitness-innovation thereafter (*from approximately* 20 *generations*). We propose, that this dip - is a consequence of a protective-behavior of the system; effectively self-adapting against the disruptive effects of the said ripple-crossover. Experimental-results comparing splicing with-and-without a restricted crossover, identified that *tail-crossovers* (Section 6.3) were responsible for this protection; providing a mechanism to reduce the probability of disruptive-crossovers (crossovers in the effective-region) . Observations from earlier results (which identified a correlation between - the stagnation of tail-growth, and convergence to the optimum-solution) supported the notion that tail-growth was being selected-for in high-fitness individuals; upon finding the optimum - growth stops. This gave credence to our argument that the subsequent rise in tail-crossovers effectively canceled-out the operation of crossover; analysis of the crossover-models from Section 6.3 confirming an increase in tail-crossovers.

The evidence, up to this point, has essentially served to support the findings of Nordin and Banzhaf [Nordin et al., 1996, Nordin et al., 1997], which suggests that the observed intron-propagation (tail-growth) is a protective behavior against the destructive effects of crossover. Splicing was seen to increase the compression of the effective-region (thereby reducing the probability of disruptive-crossover); enacting a form of "global-protection" to solutions of high quality. To support this claim, an example showed tail-crossovers to be neutral with respect to fitness; and their offspring to be of phenotypic identity to their parents (Section 6.3). Owing to the phenotypic-identity of tail-crossover offspring, and their observed insurgence - *phenotypic-duplicate elimination*, is examined with a brief review detailing it's genesis.

7.2 Investigating Phenotypic-Duplicate Elimination

We have noted that results from the previous chapter, raised a question concerning the maintenance of population-diversity. From knowledge of GE's mapping process we were aware of the redundant-encoding inherent in the system. That is, for the standard 8-bit codon, there are a possible 256 decimal values transcribable. Now, it has been seen that GE's mapping process - at each derivation-step - uses this (transcribed) decimal-value, to choose a production-rule for the current non-terminal; this, by way of a modulus operation (Section 1.5.1). It can easily be seen (that for most production-rules), many different codon-values will encode for the same production[1]. As such, it can be seen that many genotypicly-different individuals, will commonly derive the same phenotypic-solution - this especially being the case where grammar-rules contain few choices.

Cleary and O'Neill [Cleary and O'Neill, 2005], propose that the attribute grammar system, can be viewed as a decoder for legal-knapsacks - effectively confining search to the feasible-subset F of the search-space ($F \subset S = \{0, 1\}^n$). In this way, our approach can be seen as an indirect-decoder, which effectively introduces what has been termed a *heuristic-bias* (See Section 3.5.2). This adds to the already redundant-mapping of GE - further channeling multiple-genotypes into the same phenotypic output. It is worth noting, that the more restrictive the problem - the greater the heuristic-bias imposed by the attribute grammar (a higher number of constraint-clashes and subsequent IIs causing many more genotypes to decode to the same phenotype). In any case, what results, is a heavily redundant-encoding and there has been much work from the literature documenting research into the applications of such systems to knapsack-problems. As such, a short review (surveying, only the conclusions of these works), highlights the suggested importance of population-diversity, before introducing the best-known technique for it's maintenance in *phenotypic-duplicate elimination*.

It should be noted that there are many techniques for the maintainence of population diversity. Burkes et al.'s work provides a good review of such methods [Burke et al., 2004]; phenotypic-duplicate elimination has been chosen owing to it's particular success in the field of EAs for knapsack problems [Raidl and Gottlieb, 1999c, Hinterding, 1999, Hinterding, 1994].

[1]We refer the reader to the canonical text on GE for further explanation of this [O'Neill and Ryan, 2003a, O'Neill, 2001].

7.2.1 Diversity and Redundant-Encoding

We attribute Hinterding [Hinterding, 1994], in his study of decoder approaches, as the first author to comment on the importance of population-diversity for the success of an EA approach to knapsack-problems. In this work, he comments on the use of a group-mutation operator (which strongly supports the introduction of diversity), as being *essential* to the decoder's effective operation. Furthermore, and in respect of his earlier observations, he finds that specialising the evolutionary process to promote diversity significantly improves results. This is achieved by a process which discards newly-created offspring, already represented in the current population (See Section 7.2.2).

In the same work, the effect of population-size on diversity is examined. Varyious population-sizes are examined, showing that for the proposed decoder (with a redundant-mapping) - a large population is required for high success; increased diversity being pointed-out as the source of success.

Similarly, in a later work [Hinterding, 1999], Hinterding attributes the comparatively poor performance of a random-key encoding (a *numeric-real* representation (*See Section 3.4*)), to be a consequence of the very-strong redundancy inherent in it's representation and decoder-coupling; premature-convergence results, and it is observed that an overly-strong redundant mapping can be detrimental to search. This owing to an extreme number of genotypes mapping to the same phenotype.

Raidl in his improved GA [Raidl, 1998], comments on the importance of a diverse *initial* population. His special pre-optimised initialisation procedure, utilises a stochastic implementation to ensure the probability of high-diversity in this initial population. As noted in the discussion of Section 3.7, it is this non-determinism and improved diversity which is shown to provide the improvement over the earlier GA of Chu and Beasley.

7.2.2 Phenotypic-Duplicate Elimination

While the rejection of duplicate-offspring had been previously recognised as beneficial to search [Hinterding, 1994] [Hinterding, 1999] [Raidl, 1998] [Raidl, 1999b] [Chu and Beasley, 1998] [Raidl and Gottlieb, 1999a] [Raidl and Gottlieb, 1999b]; Raidl and Gottlieb present an empirical analysis into the effect, in terms of premature convergence and maintaining diversity in this way [Raidl and Gottlieb, 1999c]. They study the differences between maintaining genotypic-diversity and maintaining phenotypic-diversity. That is, in the context of decoders which provide redundant *many-to-one* genotype-phenotype mapping process, they find that ensuring the population represents a *phenotypically* diverse set of solutions - is most important to the success of the EA[2]. Similarly, they find that maintaining a genotypically diverse population fails to ensure that distinct areas of the solution space are being searched.

Building on their earlier work, statistical measures are put in place such that the black-box process of evolution, can be examined in terms of - the effect of the evolutionary-operators on phenotypic-diversity. Four decoders are examined under these statistical measures: once using no duplicate-elimination, once using genotypic-duplicate elimination and once using phenotypic-duplicate elimination. Results of the study showed phenotypic-duplicate elimination to be very important for good performance. Without it, the crossover is reported to be unreliable in maintaining an explorative search, and as such the EA's get stuck in very bad local optima[3].

[2]It is important to note, that duplicate-elimination (when carried out over a direct encoding) implicitly signifies *phenotypic*-duplicate elimination.

[3]At this point, we note the observed results for crossover over the attribute grammar system (Section 6.3); although finding optimal-solutions, and not exhibiting an overly-premature convergence - our results observe an interesting correlation to these findings

The use of duplicate-elimination at the genotypic level failed to prevent premature-convergence to highly-fit regions of the phenotypic solution-space. As such the redundant-mapping is not controlled and is overpowering (an overly-strong heuristic-bias exists). Only the strategy of phenotypic duplicate elimination was able to work in synergy with the redundancy of the mapping to prevent premature-convergence and consistently produce highly-fit near optimal solutions

The importance of duplicate-elimination in the up-keep of population-diversity, seems not to be understated. Raidl, while comparing three GA's - each with a many-to-one redundant-encoding points out that: "In all three GAs it proved to be essential to avoid duplicates in the population. Otherwise the population diversity gets lost very soon, and only few *super-individuals* survive" [Raidl, 1998]. Again, when examining problem-space search approaches [Raidl, 1999b], he notes it "to be essential to disallow duplicates in the population". Similarly, O'Neill et al., while not explicitly stating that they use the strategy of duplicate-elimination, do state - that in one of their experiments - in the creation of the initial-population a check is carried out to ensure that a diverse genetic-population exists: i.e. genotypic-duplicate elimination is employed (this, they note, is owing to experimentation with a closed-grammar which introduced significant heuristic-bias) [O'Neill et al., 2003].

7.2.3 Results for Phenotypic-Duplicate Elimination

Bourne from the recognition of this increased-redundancy of GE's encoding (i.e. the heuristic-bias imposed by the attribute-grammar), and the above-outlined work from the literature, we implemented the following study of the attribute-grammar system; examining the effect to performance of phenotypic-duplicate elimination. Table 7.1 documents the results. Results for the at-

Problem	**Khuri**	**Cotta**	**AG(Full)**	**AG(Full) +Splice**	**AG(Full)+DE**	**AG(Full)+DE +Splice**
knap15	83%	100%	83.33%	86.66%	96.6%	100%
knap20	33%	94%	76.66%	80%	100%	100%
knap28	33%	100%	40%	30%	90%	80%
knap39	4%	60%	36.66%	33.33%	43.33%	60%
knap50	1%	46%	3.33%	3.33%	16.66%	20%
Sento1	5%	75%	10%	40%	66.66%	90%
Sento2	2%	39%	3.33%	13.33%	30%	20%
Weing7	0%	40%	0%	0%	0%	0%
Weing8	6%	29%	6.66%	33.33%	36.66%	76%

Table 7.1: Table shows results for the attribute grammar system: alone; with splicing; with phenotypic-duplicate elimination(DE); and with a combination of splicing and DE. Results are again compared to [Khuri et al., 1994] - with the addition of Cotta's problem-space search approach gives comparison against a more capable algorithm. Results presented are an average of 30 independent runs.

tribute grammar system and it's splicing variant are replicated from before, with results for the addition of phenotypic-duplicate elimination (PhenoDE) being displayed for each, in direct comparison. As can be seen and in support of suggestions from the literature, PhenoDE provides a significant advantage over systems without such a diversity-maintenance measure. All results improve, with the exception of one-result for the $Sento2$ problem.

With respect to our research from the literature - the results are not surprising. Phenotypic-duplicate elimination provides a method to ensure a minimum level of diversity within the population of the EA. Conclusions from the latter chapter pointed-out that (in the context of knapsack-solutions), ripple-crossover bears too much of a disruptive force to evolving solutions, when approaching the optimum-length (i.e. crossover is only constructive in the early generations). As such we would expect to see the same disruptive change-of-role exhibited by crossover, early in evolution. Indeed, looking at the crossover-model of Figure 7.1 our suspicions are confirmed. It can be seen that for systems using PhenoDE, there is a reduction in tail-crossovers, and a slight increase in fitness-innovation being demonstrated only, in the early generations (tail-crossovers again, being represented by the gap between #XOvers and #ERXOvers). The observed increase in effective-region crossover (ERXOvers) can be seen as a direct-consequence of PhenoDE. As previously stated (See Section 6.3), offspring created from tail-crossovers were seen to be (predominantly) of phenotypic-identity to their parents. They are *phenotypic* duplicates, and as a consequence - in systems running PhenoDE, they are eliminated. The higher plot for ERXOvers is accounted for by this removal process.

Table 7.1's improved results for PhenoDE, demonstrates the better ability of the system to preserve diversity; that is, the gain in success rate can be attributable to greater diversity in the earlier generations - giving rise to more effective-diversity and therein providing a greater probability that search will find the optimal-solution. This can be seen in the slight increase in the PhenoDE plots for fitness-innovation (+XOvers and +ERXOvers) in the early generations.

Tail-crossover offspring are still reported - but as mentioned - subsequent mutations may allow them to observe a ripple-crossover effect and become phenotypically different. The remaining - low plot - for fitness-innovation would suggest that such mutations (although providing phenotypic-exploration) are not of any remarkable benefit to fitness.

From the difference observed with the crossover-models for their counterpart systems in Figure 7.1 (without PhenoDE), it can be seen that mutation is significant; working to create a phenotypic effect; but without major impact on the fitness-innovation. We must assume from this, that such mutations show more of a resemblance with random-search as opposed to a balanced exploration/exploitation trade-off. It would appear that crossover is incapable of complimenting the now, highly redundant encoding; and as such we observe a random-search effect through mutation alone.

7.3 Conclusion

If all work and no play makes Jack a dull boy - then all genotypic-search and no phenotypic-search makes an indirect-decoder ineffective. This chapter has detailed experiments over the minimum-diversity maintenance technique of *phenotypic-duplicate elimination*. Research from the literature allowed us to identify that for all works with a redundant many-to-one genotype-phenotype mapping process (or high heuristic-bias) - diversity, and particularly it's maintenance at a phenotypic level was reported to be of major importance to search.

Experiments were carried out for it's addition to the most prominent attribute grammar systems; and a significant increase in results reported in all cases (with the exception of one problem instance). A look at the model of crossover for these systems, identified that a greater number of effective-region crossovers (ERXOvers) were observed as a direct consequence. This was seen to be owing to the removal of offspring of phenotypic-identity to their parents. A significant number of tail-crossovers remain - and we observe that their existence holds evidence that mutation is working as an exploratory operator; but owing to it's previously identified ripple-effect and the lack of any significant increase in fitness-innovation, it would appear that it is working more like random-search as opposed to locally exploring the solution-space. The evolutionary operators (it is suggested), are inca-

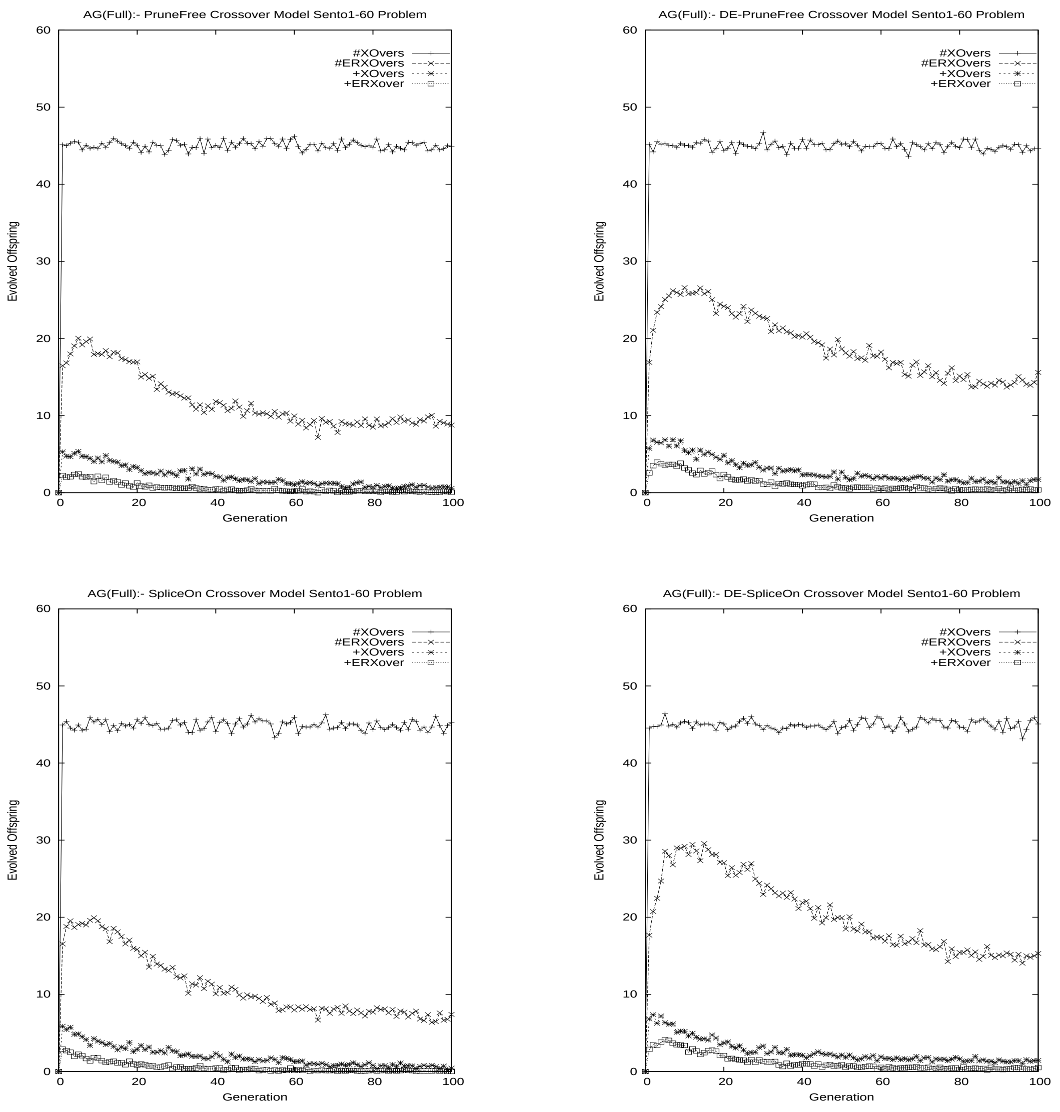

Figure 7.1: Crossover models and associated mean-best fitness plots, for systems with phenotypic-duplicate elimination (PhenoDE) and without. AG(Full), and AG(Full)-with-splicing, now compared against their equivalent with PhenoDE (marked DE). Again, crossover models are for systems where the GE-tail remains. Results presented, are averaged over 30 independent runs.

pable of conducting a balance between exploration and exploitation as they are unable to control the highly redundant mapping imposed by the attribute grammar's increase in heuristic-bias.

Chapter 8

Conclusions and Future Work

8.1 Summary and Conclusion

The grammar-based evolutionary algorithm of Grammatical Evolution (GE) has been proposed for application to problems of constrained-optimisation. More generally, this thesis sought to examine it's general applicability to the generation of problem solutions - which are context-sensitive in nature. In it's standard form, the generative-power of a context-free grammar (lying at the core of GE) was shown to be insufficient in this role - and inferior to an extended system which uses an *attribute grammar* to encode the context-sensitivity of the problems examined. Attribute grammars have been shown as a viable method to successfully describe context-sensitive problem-solutions, and by extension - show a significant performance improvement over the standard context-free mapping process.

Grammars are described under the canonical guise of Chomsky's hierarchy - with a focus on the context-free (CFGs); they are highlighted as a method to describe a confined search-space. Combining work from the field of grammar-based GP [Whigham, 1996] and EA approaches to knapsack-problems [Raidl and Gottlieb, 1999a] [Raidl and Gottlieb, 1999b] [Gottlieb, 1999a] [Gottlieb, 1999b], the standard CFG mapping process is seen to provide a *language-bias* constraining the space of search-able solutions to that of the confines of the language defined therein. Attribute grammars were seen as a method to increase the search or *heuristic-bias* - further constraining the space of search-able solutions. Work from the literature, suggests that this is an optimal approach to solving problems of a static-nature [Gottlieb, 1999b, Gottlieb, 1999a]. It was noted that - from a GE perspective - this is seen as furthering the redundancy of the genotype-phenotype mapping.

A new method has been presented, by which to classify feasibility algorithms. We acknowledge Gottlieb's method of classifying approaches to constrained-optimisation by the search-space explorable [Gottlieb, 1999b], but identify the distinguishing feature between feasibility-algorithms as their approach to evolutionary-learning: *decoders* promoting a model of Baldwinian evolutionary learning, with *repair* encompassing a Lamarkian approach. Upon application to a series of knapsack problems, it is found that the attribute grammar system - enabling GE to act as a feasible-only *decoder* for knapsack problems - significantly outperforms the standard CFG-based algorithm; which allowed the generation of infeasible candidate-solutions.

A closer look at the evolving genetic-structures allowed us to identify two classes of intron occurring along the genome: *interspersed-introns* (IIs) and *residual-introns* were identified respectively. We identified a sequence of un-interrupted IIs as an *II-region*; whereas a sequence of the latter (we noted) were previously referred to as a *GE-tail* [O'Neill, 2001]. The bounds of actively-coding expressed codons (or exons) was identified as the *effective-region*. Following from this analysis, a logical-trail of experimentation led-us to the initial findings of a *code-growth* like, propagation of residual-introns. Further investigation, led-us to believe that these large *tails* of residual-introns were being selected for in high-fitness individuals. As such a set of experiments describing *with/without* intron-removal techniques realised the finding, that reduction of the - effective-region to absolute-size ratio - (*compression of the effective-region* [Nordin et al., 1997]) through a novel *splicing* operator; significantly improved performance. The GE-tail was found to be essential to the reported improvement in performance. Questions were raised as to the benefit of IIs, where (in the absence of a GE-tail), they were seen to improve performance - whereas in it's presence, the reverse was true.

A positive model of the *fitness-innovation* emerging from GE's one-point crossover was presented and analysed. A discovery as to the context-sensitive nature of II emergence was proposed, outlining the *intrinsic-polymorphism* of GE codons and their subsequent *ripple-crossover* effect. GE-tails of residual-introns were found to be essential to the workings of this ripple-crossover acting as a buffer of residual genetic-material upon which it relies to fill it's ripple-trees. Results and analysis suggested that (for the examined problem test-suite) after ≈ 20 generations - ripple-crossover exhibits a *change-of-role* from a constructive, to disruptive force. It is suggested that in support of the findings of Nordin and Banzhaf (over a linear-GP system), that the attribute grammar system, thereafter exhibits a self-adaptive behavior, whereby individuals with a high tail-proportion propagate to subsequent generations. This effectively cancels-out crossover as a search operator.

Results for *phenotypic-duplicate elimination* showed that, maintaining a diverse population provides improvement to results. It is suggested that this diversity helps search in the early-generations and ultimately leads to a better chance of an optimum-solution being the result of an EA run. In conclusion, we find that (in the context of the problems examined) the core evolutionary operator (of crossover) is destructive in nature. As a result, a self-adaptive protection mechanism is observed by the evolving population: genomes with higher tail-proportions are seen to propogate and survive - as they have less chance of producing offspring which will be disrupted. In this way, the genetic-encoding of strong phenotypic traits can survive. This is a perfect example of evolution working as the Darwinian theory of biological-evolution would have us believe.

As recommendation of future work - we suggest that the investigations into tailoring the attribute grammar system specifically for knapsack problems; be undertaken. In this regard (and as seen in previous approaches from the literature [Gottlieb, 2000]), a look at order-preserving crossover operators is recommended (See Section 3.4.2). We suspect that they may result in a lower likelyhood of destruction as evolution progresses.

Finally, in conclusion - we reflect on the work of this thesis from an evolutionary-learning point-of-view. It was commented that the best EA approaches for knapsack-problems to-date use a Lamarkian repair strategy; similarly, in the work of this thesis - we note that the best results came as a result of the *splicing* (intron-removal) strategy. Effecting a form of genetic-repair we can postulate that the experimental evidence (combined with that of the literature) would suggest that there is some merit in the Lamarkian philosophy for evolutionary-learning. Indeed it is the opinion of the author, that a hybrid-approach to evolutionary-learning may have benefit; in the work of this thesis, for example; the attribute grammar decoder was seen to effect a form of Baldwinian learning - whilst it's coupling with a Lamarkian genetic-repair yielded the best results. This we feel is also an area for future research.

Appendix A

Publications

Some of the work reported in this thesis has already been published. The details are given below.

Conference Papers

- O'Neill, M., Cleary, R., and Nikolov, N. (2004) Solving knapsack problems with attribute grammars. In Proceedings of the Grammatical Evolution Workshop 2004.

- Cleary, R., O'Neill, M. (2005) An attribute grammar decoder for the 01 multiconstrained knapsack problem. In 5th European Conference on Evolutionary Computation in Combinatorial Optimization.

Bibliography

[Aho et al., 1986] Aho, A., Sethi, R., and Ullman, J. (1986). *Compilers: Principles, Techniques, and Tools.* Addison-Wesley.

[Angeline and Kinnear, 1996] Angeline, P. J. and Kinnear, J. K. E. (1996). *Advances in Genetic Programming 2.* MIT Press, Cambridge, MA, USA.

[Avci et al., 2003] Avci, S., Akturk, S. M., and Storer, R. H. (2003). A problem space algorithm for single machine weighted tardiness problems. In *IIE Transactions.*

[Azad, 2003] Azad, R. M. A. (2003). *A Position Independent Representation for Evolutionary Automatic Programming Algorithms - The Chorus System.* PhD thesis, University Of Limerick.

[Banzhaf, 1994] Banzhaf, W. (1994). Genotype-phenotype mapping and neutral variation - a case study in genetic programming. In *Parallel Problem Solving from Nature III.*

[Banzhaf et al., 1998] Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *An Introduction; On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann, dpunkt.verlag.

[Beasley, 1990] Beasley, J. (1990). Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

[Bruhn and Geyer-Schulz, 2002] Bruhn, P. and Geyer-Schulz, A. (2002). Genetic programming over context-free languages with linear constraints for the knapsack problem: First results. *Evolutionary Computation*.

[Burke et al., 2004] Burke, E. K., Gustafson, S., and Kendall, G. (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62.

[Chinneck, 2000] Chinneck, J. W. (2000). Practical optimization: a gentle introduction. Online Textbook at http://www.sce.carleton.ca/faculty/chinneck/po.html.

[Chomsky, 1956] Chomsky, N. (1956). Three models for the description of language. In *IRE Transactions on Information Theory*, pages 113–124.

[Chomsky, 1959] Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, pages 137–167.

[Chu and Beasley, 1998] Chu, P. and Beasley, J. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86.

[Cleary and O'Neill, 2005] Cleary, R. and O'Neill, M. (2005). An attribute grammar decoder for the 01 multiconstrained knapsack problem. In *5th European Conference on Evolutionary Computation in Combinatorial Optimization*.

[Cotta and Troya, 1998] Cotta, C. and Troya, J. M. (1998). A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In *Artificial Neural Nets and Genetic Algorithms 3*.

[CPLEX, 2005] CPLEX (2005). Cplex 9.0., paragon decision technology b.v. http://www.ilog.com/products/cplex.

[Dantzig, 1957] Dantzig, G. B. (1957). Discrete variable extremum problems. *Operations Research*, 5:266–277.

[Darwin, 1859] Darwin, C. (1859). On the origins of the species by means of natural selection, or the preservation of favoured races in the struggle for life.

[De Jong, 1999] De Jong, K. (1999). *Evolutionary Computation: Recent Developments and Open Issues*, chapter Evolutionary Algorithms in Engineering and Computer Science, pages 43–54. Wiley.

[Fischer and LeBlanc, 1991] Fischer, C. and LeBlanc, R. (1991). *Crafting a compiler with C*. Benjamin/Cummings.

[Freeman, 1998] Freeman, J. J. (1998). A linear representation for gp using context free grammars. In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., and Riolo, R., editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 72–77. Morgan Kaufmann.

[Freville and Plateau, 1994] Freville, A. and Plateau, G. (1994). An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. *Discrete Applied Mathematics*.

[Geyer-Schulz, 1995] Geyer-Schulz, A. (1995). *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, volume 3 of *Studies in Fuzziness.* Physica-Verlag, Heidelberg.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimisation and Machine Learning.* Addison-Wesley.

[Gottlieb, 1999a] Gottlieb, J. (1999a). Evolutionary algorithms for multidimensional knapsack problems: the relevance of the boundary of the feasible region. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 787. Morgan Kaufman.

[Gottlieb, 1999b] Gottlieb, J. (1999b). On the effectivity of evolutionary algorithms for the multidimensional knapsack problems. In *Proc. of Artificial Evolution.* Springer LNCS.

[Gottlieb, 2000] Gottlieb, J. (2000). Permutation-based evolutionary algorithms for multidimensional knapsack problem. In *Proceedings of ACM Symposium on Applied Computing.*

[Grefenstette et al., 1985] Grefenstette, J. J., Gopal, R., Rosmaita, B., and Van Gucht, D. (1985). Genetic algorithms for the traveling salesman problem. In *Proceedings of the 1st Int. Conf. on Genetic Algorithms*, pages 160–168.

[Hinterding, 1994] Hinterding, R. (1994). Mapping, order-independant genes and the knapsack problem. In *Proc. of 1st IEEE Int. Conf. on Evolutionary Computation.*

[Hinterding, 1999] Hinterding, R. (1999). Representation, constraint satisfaction and the knapsack problem. In *Proc. of 1999 IEEE Congress on Evolutionary Computation.*

[Hinterding and Michalewicz, 1998] Hinterding, R. and Michalewicz, Z. (1998). Your brains and my beauty: Parent matching for constrained optimisation. In *Proc. of 1998 IEEE Conference on Evolutionary Computation*. IEEE Press.

[Hoai et al., 2003] Hoai, N. X., McKay, R. I., and Abbass, H. A. (2003). Tree adjoining grammars, language bias, and genetic programming. In Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., and Costa, E., editors, *Genetic Programming, Proceedings of EuroGP 2003*, volume 2610, pages 340–349, Essex. Springer-Verlag.

[Hoai et al., 2002] Hoai, N. X., McKay, R. I., and Essam, D. (2002). Some experimental results with tree adjunct grammar guided genetic programming. In Foster, J. A., Lutton, E., Miller, J., Ryan, C., and Tettamanzi, A. G. B., editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278, pages 228–237, Kinsale, Ireland. Springer-Verlag.

[Joshi and Schabes, 1997] Joshi, A. and Schabes, Y. (1997). *Handbook of Formal Languages*, volume 3, chapter Tree-adjoining grammars, pages 69–124. Springer, Berlin, New York.

[Keijzer, 2002] Keijzer, M. (2002). *Scientific Discovery Using Genetic Programming*. PhD thesis, Danish Technical University, Lyngby, Denmark.

[Keller and Banzhaf, 1996] Keller, R. E. and Banzhaf, W. (1996). Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In *Genetic Programming 1996: Proceedings of the First Annual Conference*.

[Khuri et al., 1994] Khuri, S., Back, T., and Heitkotter, J. (1994). The zero/one multiple knapsack problem and genetic algorithms. In *Proceedings of the 1994 ACM symposium of Applied Computation*.

[Kimura, 1983] Kimura, M. (1983). *Neutral theory of molecular evolution*. Cambridge University Press.

[Kinnear, 1994] Kinnear, J. K. E. (1994). *Advances in Genetic Programming*. MIT Press, Cambridge, MA.

[Knuth, 1968] Knuth, D. (1968). Semantics of context-free languages. *Mathematical Systems Theory*, 2(2).

[Koza, 1989] Koza, J. (1989). Hierarchical genetic algorithms operating on populations of computer programs. In Sridharan, N., editor, *Proceedings of the 11th international conference on Artificial Intelligence*, pages 768–774. Morgan Kaufman.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[Koza, 1994] Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts.

[Koza et al., 1999] Koza, J. R., Andre, D., Bennett III, F. H., and Keane, M. (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman.

[Langdon and Poli, 2002] Langdon, W. B. and Poli, R. (2002). *Foundations of genetic programming*. Springer.

[Levenick, 1991] Levenick, J. R. (1991). Inserting introns improves genetic algorithm success rate: Taking a cue from biology. In Belew, R. and Booker, L., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 123–127, San Mateo, CA. Morgan Kaufman.

[Magazine and Oguz, 1984] Magazine, M. and Oguz, O. (1984). A heuristic algorithm for the multidimensional zero-one knapsack problem. *Operational Research.*

[Martello and Toth, 1990] Martello, S. and Toth, P. (1990). *Knapsack Problems.* J. Wiley & Sons, Chicester.

[Michalewicz, 1995] Michalewicz, Z. (1995). A survey of constraint handling techniques in evolutionary computation methods. In *Proceedings of the 4th Annual Conference on Evolutionary Programming.*

[Michalewicz, 1996] Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs, Third Edition.* Springer Verlag, third edition.

[Michalewicz and Nazhiyath, 1995] Michalewicz, Z. and Nazhiyath, G. (1995). Genocop 3: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Proc. of 2nd IEEE International Conf. on Evolutionary Computation.*

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning.* McGraw Hill, New York.

[Montana, 1994] Montana, D. J. (1994). Strongly typed genetic programming. BBN Technical Report 7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA.

[Muggleton and Raedt, 1994] Muggleton, S. and Raedt, L. D. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679.

[Naur, 1963] Naur, P. (1963). Revised report on the algorithmic language algol 60. *Communications of the ACM, 6.1*, pages 1,20.

[Nordin et al., 1999] Nordin, P., Banzhaf, W., and Francone, F. (1999). Compression of effective size in genetic programming. In Haynes, T., Langdon, W. B., O'Reilly, U.-M., Poli, R., and Rosca, J., editors, *Foundations of Genetic Programming*, Orlando, Florida, USA.

[Nordin et al., 1997] Nordin, P., Banzhaf, W., and Francone, F. D. (1997). Introns in nature and in simulated structure evolution. In Lundh, D., Olsson, B., and Narayanan, A., editors, *Bio-Computation and Emergent Computation*, Skovde, Sweden. World Scientific Publishing.

[Nordin et al., 1996] Nordin, P., Francone, F., and Banzhaf, W. (1996). Explicitly defined introns and destructive crossover in genetic programming. In Angeline, P. J. and Kinnear, Jr., K. E., editors, *Advances in Genetic Programming 2*, chapter 6, pages 111–134. MIT Press, Cambridge, MA, USA.

[Olsen, 1994] Olsen, A. L. (1994). Penalty functions and the knapsack problems. In *Proc. of the 1st Int. Conf. on Evolutionary Computation.*

[O'Neill, 2001] O'Neill, M. (2001). *Automatic Programming in an Arbitrary Language: Evolving Programs with Grammatical Evolution.* PhD thesis, University Of Limerick.

[O'Neill et al., 2004a] O'Neill, M., Brabazon, A., Nicolau, M., McGarraghy, S., and Keenan, P. (2004a). π grammatical evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference.*

[O'Neill et al., 2004b] O'Neill, M., Cleary, R., and Nikolov, N. (2004b). Solving knapsack problems with attribute grammars. In *Proceedings of the Grammatical Evolution Workshop 2004.*

[O'Neill et al., 2001] O'Neill, M., Keijzer, M., Ryan, C., Cattolico, M., and Babovic, V. (2001). Ripple crossover in genetic programming. In *Proceedings of EuroGP 2001.*

[O'Neill and Ryan, 1999] O'Neill, M. and Ryan, C. (1999). Under the hood of grammatical evolution. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, Florida USA. San Francisco, CA. Morgan Kaufmann.

[O'Neill and Ryan, 2001] O'Neill, M. and Ryan, C. (2001). Grammatical evolution. In *IEEE Transaction on Evolutionary Compuation.*

[O'Neill and Ryan, 2003a] O'Neill, M. and Ryan, C. (2003a). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language.* Kluwer Academic Publishers.

[O'Neill and Ryan, 2003b] O'Neill, M. and Ryan, C. (2003b). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*, chapter 7. Kluwer Academic Publishers.

[O'Neill et al., 2003] O'Neill, M., Ryan, C., Keijzer, M., and Cattolico, M. (2003). Crossover in grammatical evolution. *Genetic Programming and Evolable Machines*, 4(1).

[Orvosh and Davis, 1993] Orvosh, D. and Davis, L. (1993). Shall we repair? genetic algorithms, combinatorial optimization, and feasibility constraints. In *Proceedings of the 5th International Conference on Genetic Algorithms*.

[Paterson, 2002] Paterson, N. (2002). *Genetic programming with context-sensitive grammars*. PhD thesis, Saint Andrew's University.

[Paterson and Livesey, 1997] Paterson, N. and Livesey, M. (1997). Evolving caching algorithms in c by genetic programming. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., and Riolo, R. L., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*.

[Pirkul, 1987] Pirkul, H. (1987). A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics*.

[Pisinger, 1995] Pisinger, D. (1995). *Algorithms for Knapsack Problems*. PhD thesis, University of Copenhagen.

[Poli and Langdon, 1998] Poli, R. and Langdon, W. B. (1998). On the search properties of different crossover operators in genetic programming. In Koza, J., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, M. H., Garzon, D. E., Goldberg, D. E., Iba, H., and Riolo, R., editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 293–301, San Francisco, CA. Morgan Kaufmann.

[Powell and Skolnick, 1993] Powell, E. and Skolnick, M. M. (1993). Using genetic algorithms in engineering design optimisation, with non-linear constraints. In *Proceedings of the Fifth IEEE Conference on Evolutionary Computation*.

[Raidl, 1998] Raidl, G. R. (1998). An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In *Proc of 1998 IEEE Congress on Evolutionary Computation*, pages 207 – 211.

[Raidl, 1999a] Raidl, G. R. (1999a). A weight-coded genetic algorithm for the multiple container packing problem. In *Proc of the 14th ACM Symposium on Applied Computing*, pages 596–603.

[Raidl, 1999b] Raidl, G. R. (1999b). Weight-codings in a genetic algorithm for the multiconstraint knapsack problem. In *Proc of 1999 IEEE Congress on Evolutionary Computation*, pages 596–603.

[Raidl and Gottlieb, 1999a] Raidl, G. R. and Gottlieb, J. (1999a). Characterizing locality in decoder-based EAs for the multidimensional knapsack problem. In *4th European Conference on Artificial Evolution*, pages 38–52. Springer-Verlag.

[Raidl and Gottlieb, 1999b] Raidl, G. R. and Gottlieb, J. (1999b). The effects of locality on the dynamics of decoder-based evolutionary search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 787. Morgan Kaufmann.

[Raidl and Gottlieb, 1999c] Raidl, G. R. and Gottlieb, J. (1999c). On the importance of phenotypic duplicate elimination in decoder-based evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 204–211. Late-Breaking Papers.

[Ramsey et al., 1998] Ramsey, C. L., Jong, K. A. D., Grefenstette, J. J., Wu, A. S., and Burke, D. S. (1998). Genome length as an evolutionary self-adaptation. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*.

[Richardson et al., 1989] Richardson, J. T., R., P. M., G., L., and M., H. (1989). Some guidelines for genetic algorithms with penalty functions. In *Proc. of the 3rd Int. Conf. on Genetic Algorithms*.

[Ronald, 1995] Ronald, E. (1995). When selection meets seduction. In *Proc. of the 6th Int. Conf. on Genetic Algorithms*.

[Slonneger and Kurtz, 1995] Slonneger, K. and Kurtz, B. (1995). *Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach*. Addison-Wesley Publishing Company.

[Spector et al., 1999] Spector, L., Langdon, W. B., O Reilly, U.-M., and Angeline, P. J. (1999). *Advances in Genetic Programming 3*. MIT Press, Cambridge, MA, USA.

[Spencer, 1864] Spencer, H. (1864). *The Principles of Biology*. Williams and Norgate, London and Edinburgh.

[Storer et al., 1992] Storer, R., Wu, S., and Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management Science*.

[Storer et al., 1995] Storer, R., Wu, S., and Vaccari, R. (1995). Local search in problem and heuristic space for job shop scheduling. In *in Artificial Neural Nets and Genetic Algorithms 3, ORSA Journal on Computing*.

[Streeter, 2003] Streeter, M. J. (2003). The root causes of code growth in genetic programming. In *Genetic Programming, Proceedings of EuroGP 2003*, volume 2610 of *LNCS*, pages 449–448, Essex. Springer-Verlag.

[Watt and Brown, 2000] Watt, D. A. and Brown, D. F. (2000). *Programming language processors in Java : compilers and interpreters.* Harlow:Prentice Hall.

[Whigham, 1995] Whigham, P. A. (1995). Inductive bias and genetic programming. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications.*

[Whigham, 1996] Whigham, P. A. (1996). Search bias, language bias, and genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 230–237.

[Whitley et al., 1994] Whitley, D., Gordon, S., and Mathias, K. (1994). Lamarkian evolution, the baldwin effect and function optimization. In *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature.*

[Wong, 1995] Wong, M. L. (1995). *Evolutionary program induction directed by logic grammars.* PhD thesis, Department of Computer Science and Engineering. The Chinese University of Hong Kong.

[Wong and S., 1997] Wong, M. L. and S., L. K. (1997). Evolutionary program induction directed by logic grammars. *Evolutionary Computation.*

[Yu and Bentley, 1998] Yu, T. and Bentley, P. (1998). Methods to evolve legal phenotypes. In *Proceedings of the Fifth International Conference on Parallel Problem Solving From Nature*, pages 280–291.

[Zitzler, 1999] Zitzler, E. (1999). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications.* PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.