CONTROLLED GENETIC PROGRAMMING SEARCH FOR SOLVING DECEPTIVE PROBLEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMİN ERKAN KORKMAZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

MARCH 2003

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Tayfur Öztürk
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Göktürk Üçoluk
Supervisor

Examining Committee Members

Prof. Dr. Halil Altay Güvenir

Prof. Dr. Faruk Polat

Assoc. Prof. Dr. Hakkı Toroslu

Assoc. Prof. Dr. Göktürk Üçoluk

Prof. Dr. Fatoş Yarman Vural

ABSTRACT

CONTROLLED GENETIC PROGRAMMING SEARCH FOR SOLVING DECEPTIVE PROBLEMS

Korkmaz, Emin Erkan
Ph.D., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Göktürk Üçoluk

March 2003, 77 pages

Traditional Genetic Programming randomly combines subtrees by applying crossover. There is a growing interest in methods that can control such recombination operations. In this thesis, a new approach is presented for guiding the recombination process for *Genetic Programming*. The method is based on extracting the *global information* of the promising solutions that appear during the genetic search. The aim is to use this information to control the crossover operation afterwards. A separate control module is used to process the collected information. This module guides the search process by sending feedback to the genetic engine about the consequences of possible recombination alternatives.

Keywords: genetic programming, recombination, epistasis, deception

ÖZ

ALDATICI PROBLEMLERİ ÇÖZEBİLMEK İÇİN KONTROLLÜ EVRİMSEL PROGRAMLAMA

Korkmaz, Emin Erkan
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Göktürk Üçoluk

Mart 2003, 77 sayfa

Geleneksel Evrimsel Programlama kromozom parcalarını çaprazlama işlemi sırasında rastgele değiştirir. Bu tür rastgele gerçekleştirilen işlemleri kontrol altına alabilecek yöntemlere artan bir ilgi vardır. Bu tez çalışmasında sözedilen işlemleri yönlendirecek yeni bir yaklaşım sunulmaktadır. Öne sürülen metod, genetik arama sırasında ortaya çıkan umut vadeden kromozomların *bütünsel yapısını* incelemeye dayanmaktadır. Amaç elde edilen bilgiyi daha sonra çaprazlama işlemini yönlendirmekte kullanmaktır. Toplanılan bilgiyi işlemek için ayrı bir kontrol modülü tasarlanmıştır. Bu modül genetik motoruna gerçekleştirilecek işlemlerin olası sonuçlarını bildirerek genetik arama işlemini kontrol etmektedir.

Anahtar Kelimeler: evrimsel programlama, yeniden birleştirim, içsel bağıntı, aldatıcılık

ACKNOWLEDGMENTS

I would like to thank Dr. Göktürk Üçoluk for his warm friendship and support. I want to thank all my friends, especially Dr. Onur Tolga Şehitoğlu for his continuous help. It would be impossible to complete this study without his technical support.

To love and rebellion

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF TABLES

TABLE

CHAPTER 1

INTRODUCTION

Evolutionary computation is the general name used for the group of search algorithms which use Darwinian-like evolutionary processes to solve problems. The Darwinian principle of survival and reproduction of the fittest is the underlying approach of these algorithms. *Genetic Programming (GP)* is an important evolutionary method where the process is based on the adaptation of structured-trees. The adaptation is carried out by various recombination operators.

A population of chromosomes is used for the search process. Each chromosome represents a point in the search space. The search proceeds in a parallel manner by transforming different chromosomes simultaneously. The chromosomes of GP are constructed by using the elements of specific terminal and function sets. The search space includes all possible trees that can be built by using the elements of the specified sets. Hence, GP provides a mechanism to carry out the search among hierarchical structures with varying length and sizes.

The reason why genetic programming works is based on the notion of *Building Block*. The performance of a chromosome might depend on the existence of some critical terminal-functional elements and their configuration in the chromosome. The schemata formed by these critical elements is named as a building block. Note that chromosomes with higher fitness values have more chance to survive and to be chosen for the recombination process. Hence, it is more probable for the high performance building blocks to be spread among the population and combine with each other. The *Building Block Hypothesis* denotes that the genetic search seeks for the solution through the juxtaposition of small and high performance building blocks [21].

There is a class of problems where the genetic search is easily misled to a local optimum by the building blocks. These are called *deceptive* problems. The interaction among the partial solutions is high for these problems. In other words the contribution of a subpart of a chromosome to the overall fitness depends on the configuration of other parts. Such a strong relation among the subparts of a chromosome is denoted with the term *Epistasis*. Hence, considering only small building blocks turns out to be a deceptive choice. The global meaning of finding a possible solution goes beyond determining isolated, non-interacting building blocks and bringing them together. This is the underlying reason why genetic programming becomes inefficient and unsuccessful for deceptive problems.

The focus of the thesis is to control recombination process in GP in order to improve the performance for deceptive problems. In order to achieve this goal, a genetic search system which can deduce beneficial knowledge from its own experience, is designed. This is named as a *Self-Referential Genetic System*. The system would use the collected information to control recombination afterwards.

There are other researchers trying to control the recombination operation [26, 66]. Their focus is usually on determining the beneficial building blocks and preventing them to be disturbed during the recombination process. However, as mentioned above it is difficult to base the solution on building blocks for deceptive class of problems. Therefore, this study focuses on a new kind of abstraction on the structure of the chromosomes. A new representation is used for this scheme. The proposed representation reflects the important characteristics which would denote the overall organization of a chromosome. The information obtained by using the new representation is named as *Global Information*.

The self referential information collected during the search is based on the global information of the chromosomes. The aim is to built prototypes of the promising solutions by extracting the knowledge of what it is to be good globally. These prototypes are used to perform the right crossover operations which would keep the search among the localization of well-fit elements afterwards. The proposed representation holds two important characteristics of the chromosomes. These are the frequency information of the elements used and their position in the chromosome. These two characteristics are used to trace the critical global aspects of the high-fit chromosomes. The details of the representation used can be found in Chapter 3.

The proposed system has a dual character. The genetic engine performs the standard genetic search. On the other side, a control module has been used in order to observe the genetic search. This module collects information about the consequences of various recombination operations and performs a meta-level learning at certain periods to determine what it is to be good globally. Once the first learning process takes place, the control module starts guiding the search process by sending feedback to the genetic engine about the consequences of possible crossover operations.

The proposed method has been applied to two different real-world domains namely the *Context-Free Grammar Induction* and the *N-Parity Problem*. Both domains can be considered as highly deceptive. Their search spaces are discontinuous and the interdependency among the subparts of a possible solution is high. Traditional GP has exhibited quite a low performance for both of the problems. Hence, they are quite suitable for the research purposes of the thesis. Furthermore, a tunable benchmark problem is defined and our approach is tested on different instances of this problem. The aim is to show that the significance of the method increases as the problem is made more deceptive. Furthermore, statistical tests are carried out for the verification of the performance increase obtained.

1.1 Scope of the Thesis

Chapter 2 starts with an overview of GP and the Schemata theorem. Then, examples for various application areas of GP are presented. The significant attempts to increase the performance of GP and the previous attempts to control recombination operation are analyzed in this chapter. Also, the philosophical base of our work has been presented in the light of the ideas and discussions of various philosophers and scientists in the history.

In Chapter 3, our approach is presented. The new representation used to capture the global information of the chromosomes and the details of the control module designed are given in this chapter.

In Chapter 4, the application of our approach on CFG induction is given. The description of the experimental setup used and the results obtained for the problem are presented in this chapter.

In Chapter 5, the N-Parity problem is analyzed in the light of our approach. Again the experimental results are given for the problem.

In Chapter 6, the learning period which is an important parameter of our approach is analyzed in detail. Then the performance of our approach is compared with traditional genetic programming by using real CPU time statistics of the two methods.

In Chapter 7, a new benchmark problem is designed for genetic programming. Then the proposed method is tested on different instances of this benchmark problem.

In Chapter 8, statistical tests are carried out in order to verify the performance increase obtained.

In Chapter 9, the dissertation is summarized and conclusions are presented.

CHAPTER 2

BACKGROUND

Evolutionary computation can be considered as a part of a general movement aiming to use biological ideas in computer science. The pioneers of this movement are scientists such as Von Neumann [60, 61], Alan Turing [59] and Nobert Wiener [65]. The movement still continues today with the research on evolutionary computation, Neural Networks, artificial life and other methods inspired by the biological systems.

A wide range of methods have been proposed by the researchers in the area of evolutionary computation. All of these methods share some common features. As mentioned in the previous chapter, the usage of Darwinian-like evolutionary processes forms the core of an evolutionary method. Also some basic elements like a population of individuals, a birth-death cycle and the notions of fitness and inheritance can be considered as other common features shared by the evolutionary computation methods.

Evolutionary Programming(EP) and *Evolutionary Strategies(ES)* form the historical roots of evolutionary computation. EP was developed by Fogel in 1960s [16]. The aim is to evolve intelligent behavior by the adaptation of finite state machines. Only mutation is used for the adaptation process. On the other side ES was developed again in 1960s by Rechenberg and Schwefel [54]. The method is used for the optimization of real-valued parameters. Hence, the individuals of the population are parameter vectors and the search is carried out by mutating the parameters.

Genetic Algorithms(GA) is the most well-known and widely used method in the area. Holland [24] developed the approach with the aim of achieving a robust, adaptive system. Holland used a genetic encoding for the points in the search space and the adaptation is carried out by using different genetic operations like mutation, crossover and reproduction.

Genetic Programming is the latest method that appeared in the area. The approach can be considered as a variant of genetic algorithms where the search is carried out on hierarchical structures. Koza is the researcher who developed the method in the 1980s [33].

2.1 Overview of Genetic Programming

Representation is an important issue in traditional genetic algorithms. Genetic algorithms work on fixed-length encoded character strings. It is possible to solve various problems with the representation used. However drawbacks exist. Encoding and decoding processes give rise to the separation of the search and the solution spaces. This is a situation where issues like validity, coherence and diversity become more critical. On the other hand, fixed string length necessitates the predetermination of the shape and the size of the solutions.

It can be claimed that the natural solution for many problems is structured composition of functions. Genetic programming provides a framework to carry out the search by adapting hierarchical structures with varying length and sizes.

The chromosomes in genetic programming are structured-trees. These trees are constructed by using the elements of specific terminal and function sets. Thus, the search space includes all possible trees that can be built by the elements of the specified sets.

The general life-cycle of a genetic programming search can be summarized as follows [32].

1 Create an initial random population by using the function and the terminal set elements.

2 Breed new populations from the existing population by using the following steps, until termination criteria is met.

2.1 Assign a fitness value to each chromosome in the population. This fitness value would be a measure of how well the chromosome solves the problem.

2.2 Use a selection criteria to select chromosomes from the population and create the individuals of the next population by using the following operations.

2.2.1 Directly copy the individual to the next generation.

2.2.2 Create offsprings by using recombination or mutation.

The termination criteria mentioned in item (2) can either be finding the solution or reaching a predetermined limit on the number of generations. The selection criteria mentioned in item (2.2) could vary. The most widely used criteria are *fitness proportional selection* and *tournament selection*. As the name implies, chromosomes have chance to be selected proportional to their fitness values when fitness proportional selection is used. With tournament selection, two chromosomes are chosen randomly and the one with better fitness value is selected. The tournament may be repeated more than once to achieve a more elitist selection method.

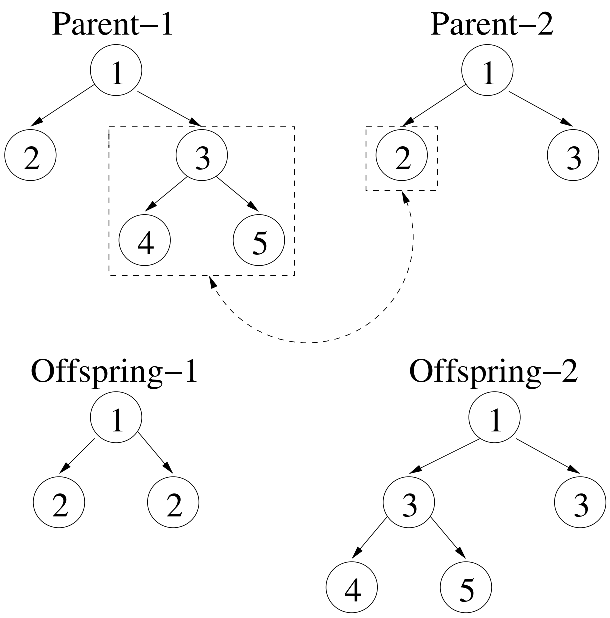

Figure 2.1: A sample crossover operation. Crossover fragments are denoted with dashed lines.

New populations are bred by either using reproduction (2.2.1) or recombination. The most widely used recombination operation is crossover. Crossover is carried out by exchanging randomly chosen subtrees of two parents. Two offsprings are obtained after the operation. In Figure 2.1 an example for the operation is given. Also, mutation causes random changes on the chromosomes. This is obtained by replacing a randomly chosen subtree of a chromosome by a new randomly created subtree.

2.2 Why Genetic Programming Works: Schema Theorem

Different approaches have been proposed in order to explain the theoretical foundations of evolutionary computation. Defining the general laws that would describe the behavior of evolutionary computation, explaining the affect of genetic operations on the search process and determining the set of suitable problems for the evolutionary approach can be considered as the core of this research area.

The schema theorem developed by John Holland [24] in mid seventies is often used to explain why genetic algorithms work. Schema can be defined as a similarity template representing a group of chromosomes. However, it is not straightforward to propose a definition of schema for genetic programming. Hence, the research carried out to explain the theoretical foundations of genetic programming can not be considered as complete yet.

Several definitions are proposed for GP schema in the literature. The research carried out by [32, 44, 45] and [63] can be considered as the initial attempts in the area. The common feature of the definitions proposed by these researchers is that their schemata are *non-rooted*. Thus, a schema can appear more than once in the same chromosome. This leads to some complications in calculating the probabilities of schema-disruption.

The approach proposed by [49] can be considered as the most comprehensive analysis of the problem. In [49], the difficulty of formulating the affect of standard crossover and mutation in GP is admitted. However, a schema theorem for one-point crossover and point mutation is proposed by [49]. These operations can be considered as restricted versions of standard crossover and mutation in GP. Point-mutation is the analogue of bit-flip mutation where a function can be substituted only with an other function of the same arity and a terminal can be substituted with an other terminal.

In order to carry out one-point crossover, the parents are traversed starting from the root and the parts with the same shape and arity are identified. Then, a random crossover point is determined in the parts shared by both of the parents and the subtrees are swapped as in standard crossover.

In the following paragraphs an overview of the schema theorem, developed in [49] is presented.

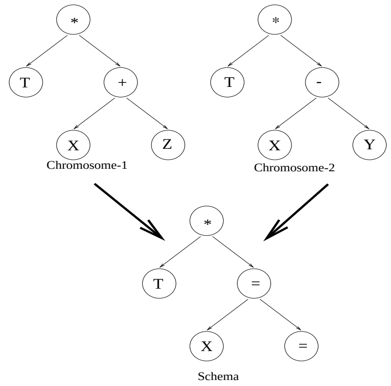

Figure 2.2: Two chromosomes and the schema preserved by them.

The schema theorem for GAs introduces a *don't care* symbol "*" in the schema vectors in addition to the binary alphabet $\{0,\ 1\}$. Hence it is possible to obtain a schema vector contained in a string by replacing the necessary symbols with the "don't care" symbol. Similarly, a schema contained in a program can be obtained by replacing the nodes with the "don't care" node denoted by the symbol "=" [49]. The "don't care" symbol would represent a single function or terminal. In Figure 2.2 the schema preserved in two different chromosomes is presented.

The necessary definitions needed for the theory are as follows.

Definition 1 (GP Schema). *A GP schema is a rooted tree composed from the elements of the set $F \cup T \cup \{=\}$. F is the function and T is the terminal set used. The element $=$ is a polymorphic function which has as many arities as the number of different arities of the function and terminal set elements.*

Definition 2 (Order). *The order $O(H)$ of a schema H is the number of non-$=$ symbols in H.*

Definition 3 (Length). *The length $N(H)$ of a schema H is the number of nodes in H.*

Definition 4 (Defining Length). *The defining length $L(H)$ of a schema H is* *the number of links in the minimum fragment including all the non-$=$ symbols in H.*

Definition 5 (Hyperspaces and Hyperplanes). *A schema H is a hyperspace if $O(H) = 0$ and hyperplane otherwise. The hyperspace corresponding to a hyperplane H can be obtained by replacing all the non-$=$ symbols of H with the "don't care" symbol $=$.*

Having the above definitions it is possible to formulate how a certain schema H would propagate throughout the generations. This propagation is in fact effected by three factors; selection process, crossover and mutation operations. The formulation is based on determining the probability that a schema H will not be disrupted at a certain generation. The expected number of chromosomes sampling a schema H at generation $t+1$ can be calculated with the following equation

$$E[m(H,t+1)] = MPr\{h \in H\}(1 - Pr\{D_m(H)\})(1 - p_c Pr\{D_c(H)\}). \tag{2.1}$$

In the above equation, $m(H,t+1)$ is the number of chromosomes having schema H at generation $t+1$ and M is the population size. The expected number is calculated by multiplying the population size with three probabilities. These are, the probability that a chromosome h sampling a schema H is selected for the mating pool ($Pr\{h \in H\}$), the probability that H is not disrupted when a chromosome h sampling H is mutated ($1 - Pr\{D_m(H)\}$) and the probability that H is not disrupted when a chromosome h sampling H is crossed over with another chromosome ($1 - p_c Pr\{D_c(H)\}$). Note that D_m and D_c denote the events that H is disrupted during mutation or crossover and p_c is the crossover probability.

Assuming that fitness proportionate selection is used, the selection probability can be formulated as

$$Pr\{h \in H\} = \frac{m(H,t)f(H,t)}{M\overline{f}(t)}, \tag{2.2}$$

where $f(H,t)$ is the mean fitness of the chromosomes holding schema H and $\overline{f}(t)$ is the mean fitness of the whole population.

Note that, point mutation changes the label of a node into a different one with a certain probability p_m. A schema H will survive during mutation only if all of its $O(H)$ non-$=$ symbols are unchanged. Hence, a schema can be disrupted by mutation with the following probability

$$Pr\{D_m(H)\} = 1 - (1 - p_m)^{O(H)}. \tag{2.3}$$

Proposing the formulation for crossover is not straightforward. There are two ways H can be disrupted. The first case, denoted by $D_{c1}(H)$, is when h is crossed over with $\hat{h}$ having a different structure. If $G(H)$ is the hyperspace associated to H, then the probability of the event D_{c1} would be as follows

$$Pr\{D_{c1}(H)\} = Pr\{D_c(H) \mid \hat{h} \notin G(H)\} Pr\{\hat{h} \notin G(H)\}. \tag{2.4}$$

In this equation, $Pr\{\hat{h} \notin G(H)\}$ denotes the probability of selecting a chromosome with different structure. Again assuming fitness proportionate selection is used, this probability can be formulated as

$$Pr\{\hat{h} \notin G(H)\} = 1 - \frac{m(G(H),t) f(G(H),t)}{M\overline{f}(t)}. \tag{2.5}$$

However, it is difficult to determine the probability $Pr\{D_c(H) \mid \hat{h} \notin G(H)\}$ since not all crossovers between parents of different structure would produce offsprings which do not sample H. However it is possible to make a simplistic assumption and consider this probability to be equal to one.

The second case where H can be disrupted is when h is crossed over with a chromosome $\hat{h}$ having the same structure with h but which does not sample H. This can be expressed as the event $D_{c2}(H) = \{D_c(H), \hat{h} \notin H, \hat{h} \in G(H)\}$. Note that the necessary condition for H to be disrupted is that crossover point should be chosen between the non-= nodes of H. The probability for such a case would be $\frac{L(H)}{N(H)-1}$. Since this is only the necessary condition, the actual probability would be lower than this value. This time the selection probability would be based on the number and mean fitness of the elements which have the same structure with h, ($\hat{h} \notin H$), but which do not sample h, ($\hat{h} \in G(H)$). Hence, the probability for the event D_{c2} can be formulated as

$$Pr\{D_{c2}(H)\} \leq \frac{L(H)}{N(H)-1} \frac{m(G(H),t) f(G(H),t) - m(H,t) f(H,t)}{M\overline{f}(t)}. \tag{2.6}$$

By considering all of the three effects discussed we can write

$$E[m(H,t+1)] \geq m(H,t)\frac{f(H,t)}{M\overline{f}(t)}.(1-p_m)^{O(H)}.$$
$$\left\{1-p_c.\left[\left(1-\frac{m(G(H),t)f(G(H),t)}{M\overline{f}(t)}\right)+\left(\frac{L(H)}{N(H)-1}\frac{m(G(H),t)f(G(H),t)-m(H,t)f(H,t)}{M\overline{f}(t)}\right)\right]\right\} \tag{2.7}$$

The above equation provides a theoretical explanation for the schema propagation during a GP search. Considering the formulation, it can be claimed that a schema H with above average fitness and short defining length will tend to survive better.

2.3 Research on GP

The idea of inducing computer programs using evolutionary techniques is not new. Even in the early days of genetic algorithms attempts have been carried out to evolve computer programs. However, the focus was more on production languages, rather then traditional computer languages [37]. The research carried out by [17, 9, 19] are the initial attempts in the area. John Koza [32] is the first researcher who has been able to come up with a convenient method to evolve computer programs. The technique proposed by Koza was named as *Genetic Programming*.

The technique have been successfully applied to a variety of real world domains. Prediction, classification, image and signal processing, optimization, financial trading, robots and autonomous agents, artificial life, neural networks and art seem to be the main areas where genetic programming applications draw attention [37]. For instance, [23] uses GP on protein geometry problems. A successful application of the method on optical character recognition is presented in [2]. [57] extracts objects from noisy pictures with the help of GP. The standard metrics are extracted by using the image processing techniques and then a genetic program is used to find out further details. [35] proposes a method to combine hand-coding and genetic programming to create control code for modular robots. It is possible to encounter even some artistic applications of the method. [10] uses genetic programming to create virtual reality environment. The genetic programs are used to produce sounds and three dimensional shapes. Certainly these are presented to a user and a manual fitness function is used for the evolution process.

The main concern of the researchers mentioned above is the application of the GP method. Certainly research on GP techniques and theory also exists in the area. In section 2.2 a summary of the research on GP theory is presented. The research on GP techniques is mostly based on the critical decision points that would effect the GP search. Deciding on the breeding policy, choosing a suitable fitness function and the usage of genetic operations are such critical points in solving a problem with GP. Thus, it is possible to encounter intensive research for proposing different breeding policies, fitness functions and new genetic operations. Also a group of researchers have focussed on the syntax of GP since representation is an important phenomena for all evolutionary methods. This has resulted obtaining different variations of the standard GP method like Pedestrian GP [5], Strongly Typed GP [40], Stack Based GP [30] and Machine Code GP [42].

Note that abstraction is an important phenomena for solving complex problems. However, standard GP has no mechanism to support data abstraction for the control of growing complexity and to facilitate code reuse. Hence research on the notion of abstraction exists in the area. For instance [3] and [53] focussed on module acquisition during the GP search. The aim is to convert beneficial building blocks into modules so that the complexity of the evolving programs would be under control. However, the most notable work on abstraction in GP is certainly the *Automatically Defined Functions*(ADFs) of John Koza [32]. ADFs are functions which can be called by the evolved genetic programs. The recombination operations rearrange the use of ADFs in the main routine. ADFs can also be evolved but in order to preserve the overall format of the program, the recombination operators are allowed to operate only within each ADF. Hence code exchange between the ADFs and the main program is prevented. This abstraction method has successfully been applied to many difficult problems and the new approach has improved the performance of GP significantly.

2.4 Controlling Genetic Programming

Various techniques have been proposed in order to increase the performance of GP. But a significant group of researchers focus on controlling and guiding the search process in GP. Note that recombination in traditional GP is random. Therefore, researchers have been mainly interested in controlling this operation. The aim is to perform more intelligent recombination that would increase the performance.

The approach proposed by [25] tries to dynamically determine the appropriate type and ratio of the recombination operators for the problem at hand. In traditional GP, the type of the operations and their usage ratios are predetermined. The researcher is expected to make the suitable decision based on her personal experience. In [25], it is tried to embed this task in the search process with an unsupervised approach. Their method is based on tracking the average progress values of different recombination operations during the search and adjusting the ratios of the candidate operators depending on their performance. Hence appropriate operators turn out to be dominant throughout the search process.

The method mentioned above is based on the determination of proper recombination operators. There are other researchers which directly focus on the application of the operators. For instance [26] proposes a method called *Recombinative Guidance for GP*. [26] states that traditional GP blindly combines subtrees and this can disrupt the beneficial building blocks. It is claimed that randomly chosen crossover points ignore the semantics of the parents. The fitness of a GP-tree is denoted as a representative value for the semantics of the tree. The proposed method is based on calculating the performance values for subtrees of a GP tree during evolution and then applying recombination operators so that the subtrees with high performance are not disturbed.

[66] focuses on the automatic extraction of knowledge from the GP programming process, too. The aim is again to increase efficiency by focusing the search. In order to achieve this goal, a knowledge repository which is expected to guide the search towards better solutions, is used. The knowledge repository collects code segments from the genetic population together with some associated information like fitness, number of occurrences, depth and so on. [66] proposes a method to calculate a single score for each segment that would reflect its overall contribution for the current task. The evolution proceeds by adding new code segments with high performance to the knowledge repository and excluding the ones which are subject to performance loss.

It is possible to find other studies where the aim is to control recombination. [63] uses a context free grammar to control crossover and mutation and [12] proposes a method which tries to preserve the context in which subtrees appeared in the parent trees before the crossover operation.

Similar approaches could be found in the area of Genetic Algorithms too. In [4, 41, 48], the focus is on extracting information from the promising solutions in order to generate new solutions. On the other side, [21] and [29] use new representations for the chromosomes so that the building blocks are grouped together and the defining length of the schemata is kept as small as possible.

2.5 Parts versus Whole

An important characteristics of the mentioned methods on controlling GP is to track the good solutions found so far and construct a model on these examples which would guide the rest of the search. This is a new approach where the genetic search gains a self-referential character. The genetic search is started as usual. However, a second meta-level module exists which observes the search process. Once sufficient data is collected to construct the model, the meta-module starts supervising the search. The importance of this approach is explicitly noted by [15]. Natural phenomena is analyzed in relation with the notions of computability and incomputability in [15]. It is stated that recursion, parallelism and adaptation are interesting attributes of complex systems. An adaptive system which also receives feedback from itself is denoted as the final level in the hierarchy of computational systems [15].

On the other side, the attempts presented are mostly based on determining the important building blocks and preventing them to be disturbed by the recombination operations. However, [47] states that for some functions, even if it is possible to decompose the function into some components, the subfunctions could interact. In such a case it becomes impossible to consider each subfunction independently, optimize it and then obtain the optimum by combining the partial solutions. In fact this statement points out the underlying reason why genetic approach fails for a certain class of problems. Genetic approach too, aims to solve the problem by dividing it into smaller components. Note that the chromosomes with higher fitness values are chosen for recombination. Having a high fitness value would mean holding a subpart of the solution. Different subsolutions that appear in different chromosomes of the population are expected to combine by the recombination operations. This can be considered as an intelligent way of implementing the *"Divide and Conquer"* method. This is a very dominant approach in science for hundreds of years. However, as stated by [47] the approach is obliged to fail for a certain class of problems. It is impossible to consider some problems as a combination of subsolutions. It is the case that a subpart of the problem gains meaning only in relation with other parts of the problem. Therefore, it becomes difficult to trace isolated building blocks that might appear in the population. It can be claimed that the degree of the interrelationship among the subparts would determine the success level of the genetic approach on the problem.

For the deceptive problems, it is clear that an attempt based on determining building blocks is not expected to increase the performance a lot. However, an analysis of the global information of well-fit elements might give clues to represent dependencies of subparts in a GP-tree. Hence a new approach which would increase the performance for deceptive problems can be based on such an analysis.

2.5.1 Philosophical Base of Our Work: Prototype Theory

It is clear that a new formalization is needed for the representation of global information. This representation problem is in fact related to a philosophical debate which is unsolved for thousands of years. This debate is based on the relation between the whole and its parts. The *Divide and Conquer* method implicitly implies that the whole equals to the sum of its parts. However a significant amount of scientists and philosophers have found this statement questionable and have tried to develop different approaches for the formalization of the whole. Even Plato was aware of the the notion about two thousands years ago. He clearly pointed out the priority of the whole [13]. Hegel can be accepted as the first philosopher who deeply focussed on the problem and tried to formalize an alternative approach. Hegel considers the relation of the whole and the parts as a unity of contradictions. He notes that whole cannot exist without the parts and the parts cannot exist without the whole. According to his approach this fact forms the essence of the relation between the whole and the parts. He also offers a methodology to overcome the contradiction between the parts and the whole [58]

The problem is also noted explicitly in the following quote of famous physicist David Bohm.

> Indeed, to some extent it has always been necessary and proper for man, in his thinking, to divide things up, if we tried to deal with the whole of reality at once, we would be swamped. However when this mode of thought is applied more broadly to man's notion of himself and the whole world

> in which he lives, (i.e. in his world-view) then man ceases to regard the resultant divisions as merely useful or convenient and begins to see and experience himself and this world as actually constituted of separately existing fragments. What is needed is a relativistic theory, to give up altogether the notion that the world is constituted of basic objects or building blocks. Rather one has to view the world in terms of universal flux of events and processes.

The problem is still a hot topic for today's science. [22] criticizes the classic approach to the problem. [22] gives examples from chemistry and biology in order to denote that the combination of parts is not only a quantitative collection. It is claimed that the combination would result a qualitative leap in terms of the whole. On the other side, Waldrop [62] discusses the problem in his book on order and chaos. A connection between mathematics and the formalization of the whole is proposed in [62]. It is stated that the whole always equals to a good deal more than sum of its parts and this property can be described by mathematics as a non-linear equation.

In the above paragraphs a summary of alternative approaches are presented. However it can be claimed that the proposed ideas are far away from providing a computational approach for the formalization of the whole. On the other side there is an interesting debate in cognitive science about category formation. Although the discussion is not directly related to the problem, it is helpful to give clues about the needed formalization. The classical theory on category formation in cognitive science states that concepts are *atomistic*, that is they can be broken down into smaller building blocks. However the classical view is not shared by all of the cognitive scientists. There are researchers claiming that concept formation is based on more complex processes rather than simple building blocks. The new approach is called *prototype theory.* This theory visualizes concepts as atomic structures and focuses on the overall structures of them. [36]. The comparison of the classical and the new approach in cognitive science provides a strong analogy for the decomposable and deceptive problems. The tree representation used holds different sub-solutions and how they are connected to each other. The representation used can help to solve certain amount of problems. However, when the interaction between different sub-solutions of a problem is high, the genetic approach fails together with the classical theory of concept formation.

This new approach which visualizes concepts as atomic structures has provided us the idea of mapping the chromosomes to single points in a n-dimensional space. It is this mapping which reflects what is considered as global to a tree. Our new approach has been based on this vector representation.

2.6 Decision Tree Learning by $C4.5$

Decision tree learning forms the core of the meta-level module mentioned in Chapter 1. The decision tree generator *C4.5* have been used in this module. Therefore an overview of decision tree learning and some implementation details on C4.5 are presented in this section.

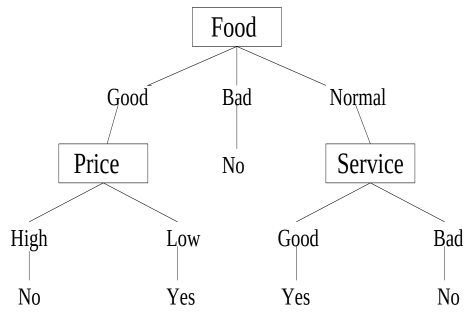

Figure 2.3: En example decision tree.

ID3 and C4.5 are algorithms introduced by Quinlan [52]. ID3 is the original algorithm proposed for the induction of decision trees. C4.5 proposes a number of extensions on this original algorithm. Decision tree learning can be defined as a method for approximating discrete-valued target functions where the function is represented in the form of a decision tree. Each node in the tree performs a test on a certain attribute. Each branch of a node corresponds to a possible value of the attribute. Given a decision tree, a new instance can be classified by starting at the root and traversing the tree depending on the attribute values of the instance until a leaf node is reached. In Figure 2.3 an example decision tree is presented. The decision tree represents a function which decides whether to eat in a restaurant or not. The decision is based on three attributes; food, price and service.

The training data needed to induce a decision tree would consist of records which have the same structure. Each record would consist of a number of attribute-value pairs. One of the attributes denotes the class of the record. In Table 2.1 example records which can be used for the induction of the *"Eat Decision Tree"* are given. Note that the first three columns in the table are reserved for the attribute values and

Table 2.1: Example records for restaurant training data.

Food	**Service**	**Price**	**Eat**
Good	Bad	High	No
Bad	Good	Low	No
Bad	Bad	High	No
Normal	Good	Low	Yes
Normal	Bad	High	No
Normal	Bad	Low	No
Normal	Good	High	Yes
Good	Bad	Low	Yes

the last column denotes the class of each record.

The basic algorithm of ID3 is given below [39].

$ID3(Examples, Target\_Attribute, Attributes)$

- Create a root node for the tree.
- If all $Examples$ are positive, return the single node with label=+
- If all $Examples$ are negative, return the single node with label=-
- If $Attributes$ is empty, return the single node with label= most common value of $Target\_Attribute$ in $Examples$.
- Otherwise begin
  - $A \leftarrow$ the attribute from $Attributes$ that best classifies $Examples$.
  - The decision attribute for $Root \leftarrow A$
  - For possible value, v_i of A,
    * Add a new tree branch below $Root$ which would test if $A = v_i$.
    * Let $Examples_{v_i}$ be the subset of $Examples$ that have value v_i for A.
    * If $Examples_{v_i}$ is empty, then add a new branch below this node with label =most common value of $Target\_Attribute$ in $Examples$.
    * Else below this new branch add the subtree $ID3(Examples, Target\_Attribute, Attributes - A)$
- Return $Root$.

In the presented algorithm $Tartget\_Attribute$ is the attribute which denotes the class of the record. Certainly $Examples$ is the training set and $Attributes$ is the set of attributes. The core part of the above algorithm is the determination of the attribute that best classifies the examples. The process is based on a statistical property named *information gain*. This property denotes how well an attribute separates the training examples according to their target class. The definition of information gain is based on the notion of entropy. Informally entropy can be considered as the unpredictability of a stochastic experiment.

Let S be the set of training examples. If the target attribute can take c different values, then this attribute would divide S into c different subsets. The entropy of S related to the mentioned classification is defined as

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{2.8}$$

In Equation 2.8 p_i is the ratio of the elements in S belonging to class i to $\mid S \mid$. The entropy of S would denote the minimum number of bits of information needed to encode the classification of an arbitrary member of S. Having the definition of entropy, the information gain of an attribute A based on S is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.9}$$

In Equation 2.9 $Values(A)$ is the set of all possible values for attribute A and S_v is the set of elements which have value v for attribute A. The second term in the equation is the expected entropy after S is partitioned using attribute A. Therefore $Gain(S, A)$ is the expected reduction in entropy when the value of A is known. In other words $Gain(S, A)$ is the amount of information obtained about the target function value, when the value of attribute A is known [39].

To illustrate the application of the algorithm, consider the training data presented in Table 2.1. Note that the records have three different attributes; $Service, Price, Food$. The first step would be to determine which of these three attributes would classify the training set best. Hence the $Gain$ value for each attribute has to be calculated. Note that the training set S consists of 8 records three of which belong to the positive class and five to the negative. According to Equation 2.8 the entropy of S would be

$$Entropy(S) = (-\frac{3}{8} \log_2 \frac{3}{8}) + (-\frac{5}{8} \log_2 \frac{5}{8}) = 0.53 + 0.42 = 0.95 \tag{2.10}$$

Now lets consider the information gain of $Food$ attribute. Note that this attribute has the value $Good$ twice, Bad twice and $Normal$ four times. Hence $\mid S_{Good} \mid = 2$, $\mid S_{Bad} \mid = 2$ and $\mid S_{Normal} \mid = 4$. Note that when $Food$ is $Good$ or $Normal$ half of the examples belong to the positive class and the other half to the negative and when $Food$ is Bad all of the examples are from the negative class. Hence using Equation 2.9 the information gain of $Food$ attribute would be

$$
\begin{aligned}
&Gain(S, Food) = 0.95- \\
&\left[\underbrace{\frac{2}{8}\left((-\frac{1}{2}\log_2\frac{1}{2}) + (-\frac{1}{2}\log_2\frac{1}{2})\right)}_{Good} + \underbrace{\frac{2}{8}(-1\log_2 1)}_{Bad} + \underbrace{\frac{4}{8}\left((-\frac{1}{2}\log_2\frac{1}{2}) + (-\frac{1}{2}\log_2\frac{1}{2})\right)}_{Normal} \right] \\
&= 0.95 - 0.25 - 0 - 0.5 = 0.2
\end{aligned} \tag{2.11}
$$

It is possible to determine the information gain for the attributes $Service$ and $Price$ using the same procedure and the results turn out to be $Gain(S, Service) = 0.16$ and $Gain(S, Price) = 0.05$. Certainly the attribute $Food$ is the one to be chosen as the root of the decision tree presented in Figure 2.3. Note that all of the records with attribute $Food = Bad$ belong to the negative class. Hence it is not necessary to carry out further testing on this branch connected to the root. When $Food$ is $Good$ or $Normal$, still it is not possible to decide on the class of the record. Therefore new decision nodes has to be added based on the information gain of the attributes $Service$ and $Price$ for the set of records having $Food = Good$ and $Food = Normal$ this time. After the necessary calculations it turns out to be the case that $Price$ is more informative when $Food = Good$ and $Service$ is more informative when $Food = Normal$. Hence two new decision nodes are added under $Good$ and $Normal$ branches of the root node in Figure 2.3.

In the above paragraphs the basic definitions of the ID3 algorithm are presented. C4.5 introduces the following extensions on the original ID3 algorithm.

- Training sets that have elements with unknown attribute values can be used while building a decision tree.
- Records that have unknown attribute values can be classified. Certainly the result is presented as a set of possibilities with corresponding probabilities.
- Attributes with continues ranges can be used.

CHAPTER 3

CONTROLLING GENETIC PROGRAMMING BASED ON THE GLOBAL INFORMATION OF CHROMOSOMES

The focus of this thesis is to propose a mechanism to control the search process during a GP run. It is possible to activate such a control mechanism at different stages of the search. Various random processes take place throughout the GP run. For instance, an initial random population is created at the beginning of the search. This can be considered as the first stage where a control mechanism can come into play. The randomness of this process aims to achieve a good distribution for the initial chromosomes. On the other hand, starting the search at some certain location of the space could be advantageous in terms of reaching to a solution. However, it is not always possible to propose assumptions about the space to be explored before the search starts. Using a random method can be considered to be the most eligible approach for the creation of the initial population unless domain specific knowledge is used for the process.

Genetic operations like crossover and mutation are other processes where random choices are carried out. The crossover and mutation points are randomly chosen in traditional GP. Also, mutation introduces random changes in some randomly selected part of the chromosome. The important difference between the characteristics of the two operations is based on their affect on the chromosome. Note that mutation is the operator that transfers the chromosomes to different parts of the search space in a random manner. This is a critical operation which enables the search process to escape from local minima. Hence, it becomes possible to proceed with the exploration of other parts of the space. It can be claimed that randomness in mutation is a necessary aspect for accomplishing the considered mission. Qualitatively speaking, GP search is crossover driven. Mutation is only serving to escape local minima traps. On the other hand, crossover is the operation that is used for the exploitation of the search space. It is aimed to obtain fitter elements by exchanging the genetic material in different chromosomes. Therefore it is meaningful to attempt controlling this operation. Aiming to perform more intelligent crossover operations, which would keep the search away from low-fit elements, can enhance the performance of the process. Therefore, crossover is selected as the operation that the aimed control mechanism would focus on.

3.1 Extracting the Global Information

3.1.1 General Framework

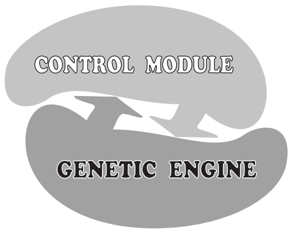

Figure 3.1: The dual structure proposed.

The method that will be used for the control of the crossover operation is based on the global information of the chromosomes. In order to process this information, we have designed a new module called *Control Module*. Figure 3.1 displays the dual structure of our system. The genetic engine which can be considered as the base structure, performs the standard genetic search. The control module as a super structure, keeps an eye on the search carried out by the genetic engine. It focuses on the global information of the chromosomes and performs a meta-level learning at certain periods to determine what it is to be good globally. Once the first learning process has taken place, the control module starts sending feedback to the genetic engine about the consequences of possible crossover operations. Then, the genetic engine chooses the most appropriate crossover points by using the feedback it receives.

As mentioned in the previous chapter, vectors that would be obtained throughout a mapping process have been chosen as the new representation for this scheme. The mapping process would determine what is considered as *global* to a tree. It is considered that the frequency of the elements used and the knowledge of how they are distributed in the chromosome might contribute to the global picture of the structure at hand. It can be claimed that these two forms of information are quite critical in terms of forming the global solution. What is more, the traditional GP is not capable of analyzing such information. The *frequency* information is important since using an element more or less than a certain number of times might be critical in terms of building the global solution. Using this information during the mapping process provides a mechanism to trace such situations. The *position of the elements* on the tree is an another critical factor in terms of the solution. The contribution of an element in the tree might depend on the distribution of the other elements. So, it can be claimed that by using these two forms of information it becomes possible to analyze the dependencies that might exist in different parts of the solution.

In order to comprehend why such a mapping process is used, it is important to focus on the underlying characteristics of the genetic search. Note that genetic search is based on the idea of stepping from one structure to another that is similar to the previous one by modifying a subpart of the chromosome throughout a genetic operation. Crossover is the critical operation that carries out this exploitation. Fitter (as well as less fit) chromosomes are expected to appear as an output of this operation. Among these new borns, the GP process will favor fitter elements. However, in order to achieve an efficient exploitation, the search space should be continuous. That is to say similar chromosomes in terms of shape and size should have fitness values mostly close to each other. In such a case, it becomes easier to proceed towards better solutions throughout the crossover operation. However, for the deceptive problems the picture is quite different. Note that the interdependency among the subparts of a chromosome is the most important aspect of these problems. This property makes the fitness of recombined chromosomes fragile during crossover. A small change on the chromosome might be hazardous or beneficial in terms of the global dependencies that should exist in the chromosome. Hence, the fitness value can change dramatically during recombination. If we could have used a function to denote the similarity of chromosomes, we could observe the following situation for the majority of the chromosomes in the search space.

$$Structually\_Similar(C_1, C_2) \not\Longrightarrow Fitness\_Alike(C_1, C_2), \tag{3.1}$$

where C_1 and C_2 are two chromosomes. *Structually_Similar* is a boolean function that would determine if two given chromosomes are similar in terms of shape and size. *Fitness_Alike* is a boolean function that would check if the fitnesses of two chromosomes are alike or not. With the increasing number of chromosomes, that does not preserve the mentioned regularity, it becomes more difficult to obtain an efficient genetic search. On the other side, even for the deceptive problems, still some similar chromosomes would have a regularity in terms of their fitness values, too. However, the similarity in terms of shape and size is not the sufficient condition for such a regularity in the deceptive domain. Hence, there should be other aspects to be considered in order to achieve the alikeness for the fitness values. The situation can be described as follows

$$Structually\_Similar(C_1, C_2) \wedge \psi(C_1, C_2) \Longrightarrow Fitness\_Alike(C_1, C_2). \tag{3.2}$$

In this implication $\psi(C_1, C_2)$ is the function that denotes the missing aspects, which were not taken into consideration by the standard procedure. The main proposal of this thesis is that an approximation for these extra aspects can be formulated by focusing on the global structure of the chromosomes. The needed information is extracted from the chromosomes by a mapping process from the set of chromosomes to a set of vectors.

In general, this mapping can be defined as

$$f : \mathcal{C} \mapsto \mathcal{F}, \tag{3.3}$$

where $\mathcal{C}$ is the set of chromosomes and $\mathcal{F} = \mathcal{X}_1 \otimes \mathcal{X}_2 \otimes ... \otimes \mathcal{X}_n$. Here, $\mathcal{X}_i \in \{\mathcal{Z}, \mathcal{Q}, \mathcal{R}, \mathcal{E}_i\}$, where $\mathcal{Z}$ is the set of integers, $\mathcal{Q}$ is the set of rational numbers, $\mathcal{R}$ is the set of real numbers and $\mathcal{E}_i$ is an ordered set of features.

Then, by using a learning algorithm, the aim is to induce a boolean function φ which would perform the following mapping.

$$\varphi : \mathcal{F} \mapsto \{True, False\}. \tag{3.4}$$

The learning algorithm would use a training set consisting of elements in $\mathcal{F}$ and the induced function φ would divide the space into two parts. The points having the value $True$ would correspond to chromosomes with high fitness values and the ones with value $False$ to low fitness elements. Hence we can write

$$\psi(C_1, C_2) \stackrel{\triangle}{=} \left[\varphi(f(\mathcal{C}_1)) \stackrel{?}{=} \varphi(f(\mathcal{C}_2))\right] \tag{3.5}$$

where ψ is the boolean function in Formula 3.2. The above formula denotes that if the points corresponding to C_1 and C_2 are in the same region of $\mathcal{F}$ according to the induced function φ, then we can conclude that the other necessary condition needed to achieve the fitness alikeness is satisfied. This is the general framework which can be used to increase the performance of the genetic search in a deceptive domain.

The critical decision about the formalization is the choice of the mapping function f. Different alternatives would exist depending on the structure of the chromosomes used or the feature set aimed to be extracted throughout the process.

Note that the chromosomes of GP are structured-trees. The mapping function to be proposed in this domain should focus on this structure. The mapping process described at the beginning of this section, focuses on two different characteristics that implicitly exists in the GP chromosomes. These are frequency of the elements used and their position in the chromosome. Hence, the transformation enables us to consider a different kind of similarity which we call the (statistical-spatial) *SS_Similarity*. In this domain, the *SS_Similarity* of chromosomes is considered to be the function ψ mentioned in implication 3.2. For the GP domain of interest, the claim of the thesis turns out to be

$$Structually\_Similar(C_1, C_2) \wedge SS\_Similar(C_1, C_2) \Longrightarrow Fitness\_Alike(C_1, C_2). \tag{3.6}$$

By this implication, it is claimed that, for the deceptive problems, if the chromosomes are similar in terms of their statistical and spatial characteristics in addition to their shapes and sizes, than it is possible to expect an alikeness between their fitness values too. Note that crossover operation itself takes care of $Structually\_Similar(C_1, C_2)$ (based on the size and shape of a chromosome). So, an extra testing mechanism is needed to clarify if the $SS\_Similarity$ is destroyed or not during the genetic process. The mapping process used to extract the statistical and spatial characteristics of the chromosomes is defined in the following paragraphs.

The chromosomes of GP are structured trees which can be defined as a tuple $\mathcal{T}(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ is a set of edges with some special constraints. These constraints are

i) Any two vertices in $\mathcal{T}$ are connected by a unique simple path.

ii) $\mathcal{T}$ is acyclic.

We propose the following choice for f and will call this choice as $\hat{f}$.

$$\hat{f} : \mathcal{T}(\mathcal{V}, \mathcal{E}) \mapsto \mathcal{Q}^n. \tag{3.7}$$

Note that the vector space $\mathcal{F}$ defined in Formula 3.3 has been chosen as Q^n for this study. The dimension of this space denoted by n is set as

$$n = \mid Terminal\_Set \mid + \mid Function\_Set \mid, \tag{3.8}$$

where $Terminal\_Set$ and $Function\_Set$ are the two sets consisting of the terminal and non-terminal elements used for the GP search. Given a tree $T(V, E)$, where V is the set of vertices and E is the set of edges, if

$$\left[\hat{f}(T(V, E))\right]_i = x_i \tag{3.9}$$

where $i = 1...n$ and $x_i \in \mathcal{Q}$, then there is a terminal or a function element in tree T which would have the frequency and position information denoted by x_i. The integer part of x_i specifies how many times this element is used in the tree, namely the frequency of that element. This can be formally expressed as

$$\exists S\left[S \subseteq V \wedge [\forall k, \forall z\left[(k \in S \wedge z \in S) \Rightarrow (Label(k) = Label(z))\right]\right] \wedge [|S| = \lfloor x_i \rfloor]] \,, \tag{3.10}$$

The function *Label* is assumed to return the label (or data) of a vertex. On the other hand, the fractional part of x_i holds the position information of the element. This is defined to be the sum of the depth values of all occurrences of the element in the tree. This depth summation is transformed into a fractional value by using a multiplicative constant. This can be formally expressed as

$$\exists U\left[U \subseteq E \wedge \forall z \exists k\left[z \in U \wedge k \in S \wedge z \in Path(k, Root(T))\right] \wedge \left[\mid U \mid = \frac{(x_i - \lfloor x_i \rfloor)}{c_0}\right]\right], \tag{3.11}$$

where S is the set of vertices specified in Formula 3.10. Also, c_0 is the constant used to transform the depth summation into a fractional value. The function *Path* returns the set of edges which form the path between the given two vertices.

The presented $\hat{f}$ has a simple formalization about the global organization of the tree and does not have a heavy computational load. The following can be mentioned about $\hat{f}$.

- Each terminal and function element is mapped to a base vector.
- By using a bottom up construction, it is possible to obtain a single vector for the whole GP-tree.

A leaf node is only mapped to its base vector while the vector for an internal node is obtained by adding the vectors of its children plus the base vector corresponding to it. By this procedure, the frequency information of each element is hold in a different component of the formed vector, as of the form of an integer. The depth information of the same element is stored as the fractional value of that rational.

Mathematically speaking, let $P(C_1, C_2, ..., C_k)$ be a subpart of a chromosome, where P is an internal node and C_i is a child node connected to this parent. The vector that would correspond to P can be obtained using the following formula.

$$V_P = V_{P_{base}}[1 + c_0 \cdot depth(P)] + \sum_{i=1}^{k} V_{C_i}, \tag{3.12}$$

where V_{C_i} is the vector corresponding to child C_i and c_0 is the constant used to transform the depth summation into a fractional value. Here, $c_0 << 1$ and the constant has to be chosen such that

$$c_0 \cdot \left[\underset{l \in TF}{\text{MAX}} \left(\sum_{\forall node \ni Label(node)=l} depth(node) \right) \right] \leq 1, \tag{3.13}$$

where $TF = Terminal\_Set \cup Function\_Set$. The inequality states that the multiplication of c_o with the maximum possible depth summation should stay below 1 so that the position information would not interfere with the frequency information.

For instance, consider the function and terminal sets; $F = \{+, -, *, /\}$ and $T = \{x\}$. The base vectors would be:

- $V_+ = [0, 0, 0, 0, 1]$
- $V_- = [0, 0, 0, 1, 0]$
- $V_/ = [0, 0, 1, 0, 0]$
- $V_* = [0, 1, 0, 0, 0]$.
- $V_x = [1, 0, 0, 0, 0]$.

Note that the dimension of the vectors is determined as the total number of function and terminal elements.

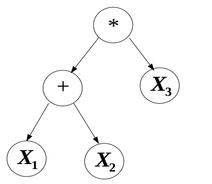

Figure 3.2: A sample chromosome.

For the tree in Figure 3.2, the vector construction mechanism will be as follows. The base vectors are as specified above. The three different occurrences of the terminal element X are labeled as X_1, X_2 and X_3. Since X_1 and X_2 have the same depth value, their vectors will be the same. This vector would be $[1.02, 0, 0, 0, 0]$. On the other side, the vector corresponding to X_3 will be $[1.01, 0, 0, 0, 0]$ due to the depth value of 1. According to Equation 3.12, the vector of $'+'$ will be $[2.04, 0, 0, 0, 1.01]$. Finally, the vector corresponding to $'*'$ which would be the vector of the whole tree, can be obtained by using the vectors of $'+'$ and X_3 this time. Hence, $V_* = [3.05, 1, 0, 0, 1.01]$. Note that, each dimension of this vector provides information about the usage of a terminal or a function element. For instance the first dimension is reserved for the terminal element X. The value in this dimension denotes that the terminal element occurs three times and the sum of the depths of these three different occurrences is five. On the other side, the second dimension denotes that $'*'$ operation is only used for once as the root of the tree.

Note that the constant value that is used to transform the depth value into a fractional one is 0.01. However, if the sum of the depths exceeds 100, depth and frequency information will interfere with each other. If this is possible, a smaller constant has to be used. At least, it should be guaranteed that the number of such ill formed vectors are kept small enough that the learning process does not get affected.

Also note that different chromosomes can be mapped to the same vector. However, this is not contradictory with our assumption since different elements in the base structure could be similar in terms of the super structure.

3.1.2 Implementation of the idea

The interaction between the genetic engine and the control module is as follows. For each chromosome in the population, the corresponding vector is formed and sent to the control module together with its fitness value. The control module collects the vectors and fitness values for a certain period of generations, which we call the *learning period.* Then the average and the standard deviation of the fitness values are calculated.

The control module forms the training set using the elements with fitness values deviating from the average more than the standard deviation. The ones with positive deviation are marked as positive examples and the others as negative. The *"C4.5, Decision Tree Generator"* is used to generate the abstraction over the training set. Then for each crossover operation to be performed, the genetic engine sends to the control module three different alternative crossover points. The control module predicts if the alternative offsprings will be in the positive or the negative class by using the decision tree generated by *C4.5.* The best alternative is chosen by the genetic engine and the learning process is repeated periodically. Note that the using specifically three alternatives is an empirical choice.

Using a certain percentage of the best and the worst elements could be another method to form the training set. However, it is observed that using standard deviation

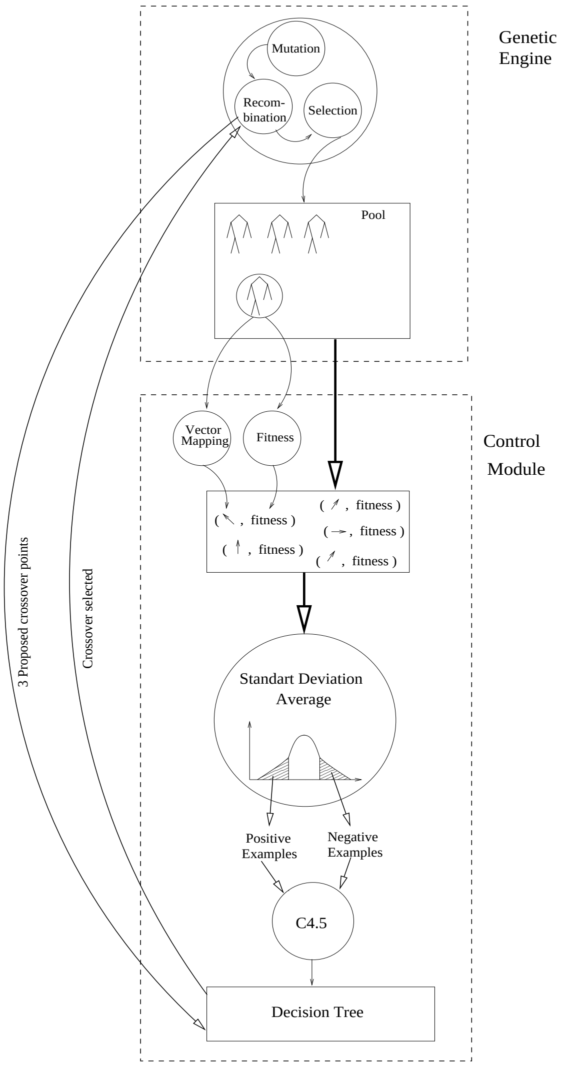

Figure 3.3: Interaction between the Control Module and the Genetic search.

provides a flexibility to the control module. Sometimes it is possible for the control module to guide the genetic search to a local minimum. In such a case, the standard deviation decreases a lot and no positive examples could be found for the training set. Since nothing could be learned, crossover becomes random again. This makes it more probable to escape from the local minima since recombination is not controlled. However, determining a percentage of the examples as positive always, looks like insisting on the mistake that the control module has made.

CHAPTER 4

$TESTBED_1$: CONTEXT FREE GRAMMAR INDUCTION

Grammars are necessary and useful tools for many applications. Therefore there has been a big interest in inducing classes of grammars in the area of machine learning.

Various attempts have been carried out for automatically inferring different grammar classes. A significant amount of research has been devoted to the induction of context free grammars. CFGs can be used for processing natural languages, too. Automatic induction of a natural language grammar by only using example sentences draws attention.

Although statistical approaches form the main stream among the researchers [7, 6, 8], several attempts have been carried out to attack the problem with evolutionary techniques, too.

For instance, [55] proposes a method based on genetic algorithms. The authors claim that although statistical methods offer a possible solution to the problem, drawbacks exist. It is quite difficult to escape from problems like the "zero-frequency" by using a statistical approach. They propose their evolutionary approach where each chromosome in the population represents a CFG. The fitness is measured by the ability of the grammar to parse a set of sample strings. The crossover operation used for the process only combines subsets of two different grammars. Hence, the operation does not create new rules, but just makes new combinations of the existing rules distributed to different chromosomes. On the other side mutation is allowed to modify any symbol in the grammar. Hence, it is the operation used to introduce new rules that may have not appeared in earlier generations. The proposed approach is able to infer simple natural language grammars over a small set of training examples. The example grammar given as an output of the experiments covers simple sentences consisting of a noun and a verb phrase. The noun phrase contains a single determiner and a set of nouns. The verb phrase contains a single verb and the noun phrase recursively.

The work proposed by [38] is very similar to [55]. However, it is stressed that straight forward application of genetic algorithms is not very effective at grammar induction. Thus modifications are proposed to increase the success of the method. An example to the proposed modifications is to enlarge the definition of mutation. The newly proposed operation both mixes the sub-components of a grammar and modifies certain symbols.

4.1 Genetic Programming for CFG induction

GP is appropriate for solving symbolic tasks. Induction of context-free grammars can be visualized as a symbolic task too. The left-hand side of a rewrite rule in a grammar can be treated as a function which is composed of the right-hand side elements of the same rule. Thus, it is possible to represent context-free grammars as structured trees. The problem can be formulated as a search problem among the possible tree structures that can be formed using a set of terminal and nonterminal symbols.

Different normal forms have been proposed for context-free grammars. A normal form can be described as a set of conditions that the rules of the grammar must satisfy [56]. Chomsky Normal Form provides a simple representation for the grammars and it is possible to express any CFG in this form [56]. Therefore it has been decided to use this form for the evolution process.

A CFG is accepted to be in Chomsky Normal Form if the rules of the grammar are in one of the following forms.

(i) $A \rightarrow BC$

(ii) $A \rightarrow a$

(iii) $S \rightarrow \lambda$

The S symbol in item (iii) is the start symbol. Note that the evolved grammar rules can have at most two symbols on the right-hand side due to the length constraint in Chomsky Normal form. The first form denoted in item *(i)* points out that if there are two elements on the right-hand side of a rule, then both of them should be non-terminals. However, this form could easily be disrupted by recombination. Terminal elements can appear on the right-hand side after crossover or mutation. Two alternatives exist to tackle with the problem. Either special constraints should be defined on the recombination process to keep the offsprings in this form or the definition can be loosened for the genetic search. The second alternative is chosen and it has been decided not to use the form directly. Terminal elements are allowed to appear in the first form too. Hence, no constraint has been used for recombination, but the search is carried out in a larger space. Note that the search space still includes grammars in Chomsky normal form.

Also any evolved grammar can be converted to Chomsky Normal Form with a slight modification. For instance assume that a grammar has a rule $X \to a, Y$ which disturbs the first form. The grammar can be converted to Chomsky Normal Form by replacing the rule with two new rules; $X \to Z, Y$ and $Z \to a$ where Z is a new nonterminal that does not exist in the grammar.

The problem can be considered as a highly deceptive one. It is possible to divide a grammar into subparts like noun phrase (NP), verb phrase (VP) or prepositional phrase (PP). However, these subparts do not have clear borders. Overlapping exists due to the fact that (NP) is a part of (VP) and (PP). On the other side the success of a candidate grammar is fragile during the recombination process. Even a single modification on a well-fit element may be hazardous. For instance every sentence has a single verb. This makes the usage of the symbol V very critical. A modification on the configuration of this element might heavily effect the fitness of the offspring. Such factors and the interdependency among the subparts of a grammar makes the search space discontinuous. Very similar grammars might have totally different fitness values. Hence, the genetic search is expected to have difficulties about a satisfactory convergence.

Natural language sentences have been used in order to form the training set for the CFG-induction problem. The training set consists of 21 positive examples and 17 negative examples. The sentences formalize a subset of English including sentences consisting of structures like NP, VP and PP. The Noun phrase (NP) is quite simple and consists of a determiner (D) followed by a noun (N) or compound noun. On the other hand, the verb phrase (VP) can be intransitive, transitive or ditransitive and the prepositional phrase (PP) could be attached to VP or NP. The prepositional phrase consists of a preposition (P) followed by a NP. "$DNNVPDN.$", "$DNV.$" and "$DNNV$" are examples from the positive set and "$VN.$" , "$DVN.$" from the negative set. The aim is to induce a CFG that can parse the positive examples and reject the negative ones. Each chromosome in the population is a candidate grammar. Note that crossover would exchange subtrees of selected parents and mutation would replace a subpart of a chromosome with a randomly created tree. Hence, both of the operations can create new rules. What is more crossover can also exchange rules of existing grammars.

An example grammar that would cover the positive and the negative set is given below. Note that this is a simple grammar which covers only a small subset of English.

- $S \rightarrow NP, VP$
- $NP \rightarrow NP, PP$
- $NP \rightarrow D, \bar{N}$
- $\bar{N} \rightarrow N, \bar{N}$
- $\bar{N} \rightarrow N$
- $VP \rightarrow VP, PP$
- $VP \rightarrow V$
- $VP \rightarrow V, NP$
- $VP \rightarrow V, NP, NP$
- $PP \rightarrow P, NP$

4.1.1 Shift-Reduce Parsing

In order to determine if the sentences in the training set are derivable from the rules of an evolved grammar, a shift-reduce parser has been used. This is a bottom-up parser which works in a depth-first manner. The parses uses a stack for holding the grammar symbols and an input buffer for the string to be parsed. Initially the stack is empty and the parser operates by shifting zero or more symbols onto the stack until a string of grammar symbols that represents the expansion of a non-terminal appears on the stack. Then the string on the stack is reduced to the left side of the appropriate rule of the grammar. The cycle is repeated by the parser until an error is detected or until the stack contains the start symbol and the input is empty.

The algorithm adopted from [56] is given below.

- *Input:* A context-free grammar $G = (V, \sum, P, S)$, a string $p \in \sum^*$ and a stack $\mathbf{S}$
- The stack elements are triples $[u, i, v]$ where $w = uv$ is the sentential form that was reduced and i is the index of the rule that is used for the reduction process. Hence, u is the substring whose suffixes are compared with the right-hand side of the rules.

1 $PUSH([\lambda, 0, p], \mathbf{S})$

2 Repeat

  2.1 $[u, i, v] = POP(\mathbf{S})$

  2.2 dead-end=false

  2.3 repeat

  Find the rule with number j, $(j > i)$ that satisfies

  i) $A \to w$ with $u = qw$ and $A \neq S$ or

  ii) $S \to w$ with $u = w$ and $v = \lambda$

  2.3.1 If there is such a j

  $PUSH([u, j, v], \mathbf{S})$, $u = qA$, i=0

  2.3.2 If there is no such j and $v \neq \lambda$

  $shift(u, v)$, $i = 0$

  2.3.3 If there is no such j and $v = \lambda$ then $dead - end = 0$

  until $(u = S)$ or $dead - end$

  until $(u = S)$ or $EMPTY(\mathbf{S})$

3 If $EMPTY(\mathbf{S})$ then reject else accept

In the above algorithm, note that the condition that detects $dead{-}ends$ is specified in step 2.3.3. Here the condition that string v is empty denotes that all reductions have been examined and the parser is forced to backtrack.

4.2 Experimental Results

Different alternatives exist for the formulation of fitness function. One possible way to overcome the difficulty of deception is focusing on the formulation of a qualified fitness function. However, such attempts usually result in providing domain knowledge for the problem at hand. Note that our focus has been proposing a domain independent approach by using a control module. Therefore, it has been decided to use a simple, standard fitness function for the problem. The fitness function used can be defined as follows. For grammar G, if S is the set of sentences consisting of the positive examples that G cannot parse and the negative examples that G parses, then the fitness of G is:

$$F(G) = \sum_{S_i \in S} SENTENCELENGTH(S_i) \tag{4.1}$$

So the aim is to minimize the fitness function. For the test data the worst fitness for a grammar could be 243 which is the sum of the length of all sentences both in the positive and the negative set. And the best fitness is certainly zero which can be achieved when a grammar parses all of the examples in the positive set and rejects all of the negative set.

The terminal and function sets are $T = \{D, N, V, P\}$ and $F = \{X_1, X_2, ..., X_{10}\}$. Note that ten non-terminal elements are used for the search process. This number is more than the minimum non-terminal elements needed to construct the grammar. Considering the restriction specified in the previous section, each element of the function set could have one or two arguments.

The mapping process described in Chapter 3 is used to form the vectors for the control module. Since the total number of elements in the function and the terminal set is 14, the vectors will be formed in $\mathcal{Q}^{14}$.

The genetic parameters are fixed for all of the experiments carried out. These parameters are listed below.

- Population size = 100
- Crossover at function point fraction = 0.1
- Crossover at any point fraction = 0.7
- Reproduction fraction = 0.1

- Mutation fraction = 0.1
- Number of Generations = 5000
- Selection Method: Fitness Proportional.

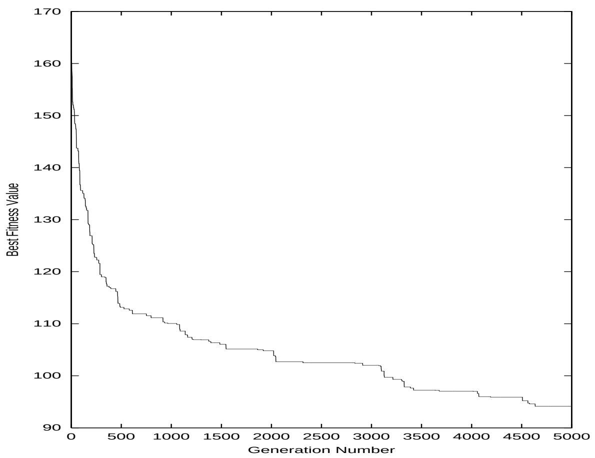

Figure 4.1: Performance of traditional GP on CFG-induction problem. Average of 20 runs are presented.

In Figure 4.1 the performance of traditional GP on CFG-induction problem is presented. Note that a randomly created grammar can have a fitness value around 160. This is the average best fitness value observed in the initial random population. It is not possible to claim that the performance of GP is satisfactory in the proceeding generations. The decrease in the best fitness value is lost very quickly. The best fitness value that can be obtained at the end of the search is around 90. Note that the worse fitness value is 243. Therefore the best grammar found at the end of the search would still fail on more than one third of the training set. This is quite a unsatisfactory result. It is difficult to claim that the solution might be found if the limit on the maximum number of generations is increased. The fitness curve already flattens in the first 5000 generations. It is not probable to encounter a big leap in the fitness values when such a situation occurs.

When the chromosomes that appear throughout the search are analyzed, it is observed that they bloat so quickly. This is a situation observed frequently for deceptive problems. When the search quickly converges to a local minima, it becomes difficult to obtain better chromosomes throughout the recombination process. On the other side, if the success of the well-fit elements in the population is fragile against recombination, this strange but rational process starts. Chromosomes start picking up unfunctional code and get larger and larger [43]. This unfunctional parts do not have an affect on the fitness of the chromosome. However, if a chromosome can get large enough with such code, it can protect its functional parts to be attacked and damaged by the recombination process. Hence, the aim of survival is achieved. Obviously once such a situation occurs it becomes almost impossible to reach to a solution afterwards.

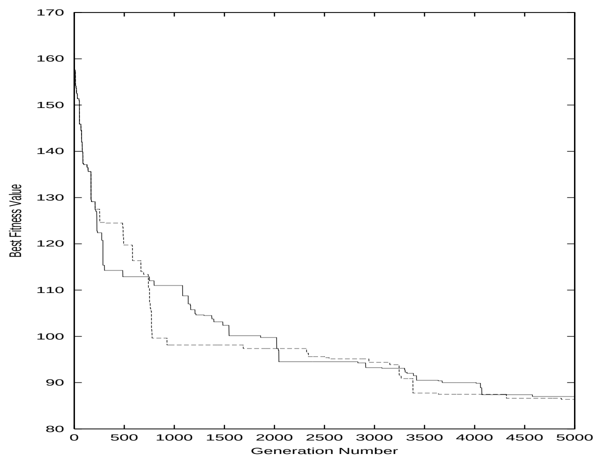

Figure 4.2: Comparison of controlled search and basic GP for the CFG-induction problem. The dashed lines denote the performance of controlled search. Learning period is 200.

Certainly experiments are carried out using the control module presented in the previous chapter. Note that the learning period is an important parameter for our approach. For the first trial the learning period has been set as 30. Both the controlled search and the straightforward application of GP have been run using eight different random seeds. Surprisingly it has been observed that the controlled search performed worse than the straightforward application. It seems that the information sent by the control module to the genetic engine was misleading and directed the search to a local minima resulting a performance worse than random crossover. An increase in the performance had been obtained with simpler data and with smaller number of function elements. The main difference with this initial attempt is the total number of function and terminal elements used. This total number is 14 for this new setup. Therefore it is thought that the data collected with the learning period of 30 might be quite low for making a reasonable abstraction over vectors with this dimension. On the other side, we have observed that the decision trees induced for this case are simple and contain less information. Therefore it has been decided to increase the learning period.

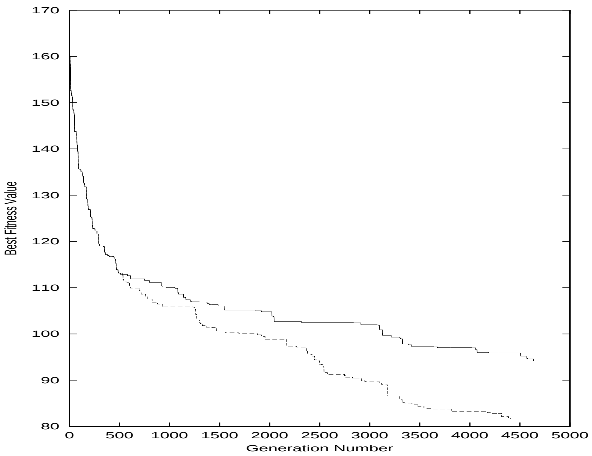

Figure 4.3: Comparison of controlled search and basic GP for the CFG-induction problem. The dashed lines denote the performance of controlled search. Learning period is 500.

Figure 4.2 presents the comparison with basic GP when the learning period is increased to 200. Again the results denote the average of eight runs with different random seeds. The performance of the controlled search clearly increased, compared to the trial with a learning period of 30. However, still it is not the case that controlled search can outperform the straightforward application.

However, the increase in the performance parallel to the increase in the learning period is encouraging. Therefore another trial has been been carried out with a learning period of 500 generations this time. Figure 4.3 presents this new trial. This time the average of twenty different runs are used in order to increase the liability of the performance increase obtained. As it can be seen in the figure, the desired performance increase has been obtained. When the control module is used, at the end of the search it is possible to obtain fitness values close to 80 on the average. Still this is a result far away from the global optimum. However, the increase in the performance denotes that the control module is capable of providing beneficial information by focusing on the overall configuration of the well-fit elements that appear throughout the search.

CHAPTER 5

$TESTBED_2$: N-PARITY PROBLEM

The N-parity problem has been selected too in order to analyze our approach. The aim is to induce a function which takes a binary sequence of length n and returns true if the number of ones in the sequence is even and false otherwise. The function would consist of internal operators $AND, OR, NAND$ and NOR. Figure 5.1 shows the solution for the 2-parity problem. Note that each leaf on the tree is an element of the 2-bit sequence.

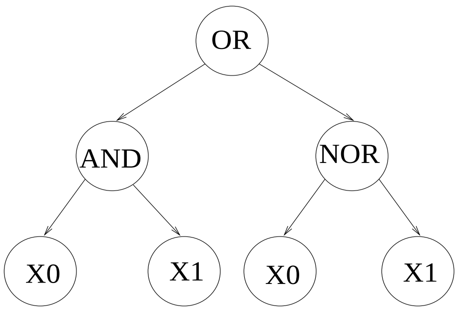

Figure 5.1: Solution for the 2-parity problem.

The problem is to our interest as it is highly deceptive. [11] states that the problem quickly becomes more difficult with increasing order. He also denotes that flipping any bit in the sequence inverts the outcome of the parity function and notes this as a fact to denote the hardness of the problem. In Table 5.1 the length of the smallest solutions for different instances of N are presented [11]. Note that the size of the smallest solution increases proportional to N^2.

The 5-parity problem has been chosen for the test cases since [11] denotes that no

Table 5.1: Length of the shortest solutions for the N-parity problem

n	1	2	3	4	5	6	7
Length	3	7	19	31	55	79	103

solutions is found by basic GP for the 5-parity. [20] states that $N = 5$ represents an upper limit for traditional GP. It is denoted that N-parity is a very hard classification problem for GP and even a large population size of 8000 is not enough to solve the problem for N=5.

The function and the terminal sets are $F = \{AND, OR, NAND, NOR\}$ and $T = \{X_1, X_2, X_3, X_4, X_5\}$. T represents the binary input sequence of length five. The number of possible input binary sequences is 32 for the 5-parity problem. The fitness function simply adds a penalty of one if the induced function returns the wrong answer for an input sequence. Hence, the fitness value may range between 0 and 32.

Again the performance of traditional GP and the controlled search are compared on the problem. The experiments are carried out using different learning periods. The performance of traditional GP is unsatisfactory as expected. A randomly created chromosome can have a fitness value around 15. However, the search carried out can not decrease this initial value below 9 on the average. GP is again far away from reaching to a solution for the problem.

Certainly the same set of genetic parameters are used for all the trials. These parameters are listed below. Note that the number of generations has been increased to 20000. The initial trials on the problem have been carried out using 5000 generations. However, it is observed that the search process still has tendency to decrease the fitness value after 5000 generations. Therefore the generation number is increased until the fitness curve flattens. This denotes that the convergence has stabilized.

The list of genetic parameters used is as follows

- Population size = 100
- Crossover at function point fraction = 0.1
- Crossover at any point fraction = 0.7
- Reproduction fraction = 0.1
- Mutation fraction = 0.1

- Number of Generations = 20000
- Selection Method: Fitness Proportional.

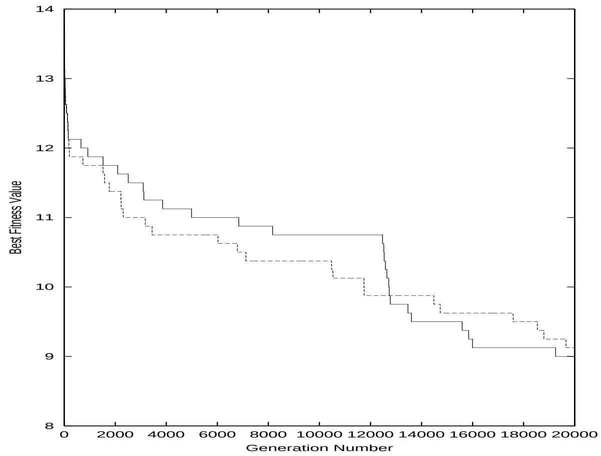

Figure 5.2: Comparison of controlled search and basic GP for the N-Parity problem. The dashed lines denote the performance of controlled search. Learning period is 200.

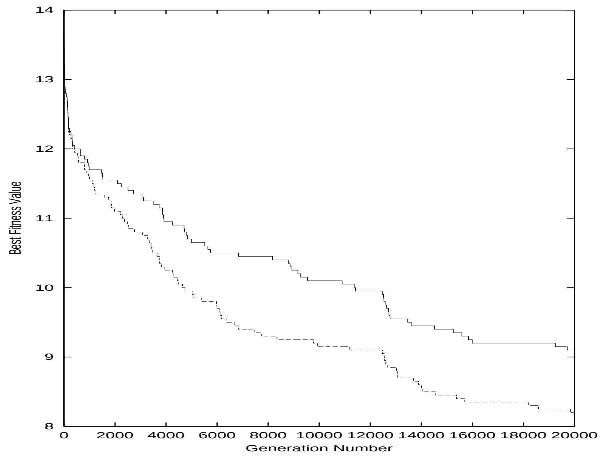

Figure 5.3: Comparison of controlled search and basic GP for the N-Parity problem. The dashed lines denote the performance of controlled search. Learning period is 500.

The first test case has been carried out again using a learning period of 30. Similarly eight different runs have been carried out with various random seeds both for basic and controlled GP. The results obtained were consistent with the CFG-induction problem. Again the controlled search exhibited a worse performance. Considering the total number of terminal and function set elements which is 9,obtaining a similar performance is not surprising. Therefore the second test case has been tried with a period of 200 generations. The results of this test case are presented in Figure 5.2.

Again the results are consistent with the results obtained for CFG-induction. The controlled search can compete with the straightforward application but still cannot outperform it. A test with a learning period of 500 generations has been carried out and the results are presented in Figure 5.3. Again the average of twenty different runs is used for this learning period. This period is sufficient for 5-parity problem too and the performance increase is outstanding.

Note that the output obtained is very similar to the results in CFG-induction domain. Unfortunately the control search is not capable of finding the solution. However, the performance increase in this second domain clearly points out that the control module is capable of providing self-referential knowledge to the search process.

CHAPTER 6

OBSERVATIONS BASED ON $TESTBED_1$ & $TESTBED_2$: DETERMINING THE LIMITS OF THE APPROACH

6.1 The Learning Period

The experiments carried out denote that there is a correlation between the total number of terminal-functional elements used and the learning period. When this total number is large, the learning period has to be increased. This is an expected result since the total number of elements determines the dimension of the vectors formed. Hence, *C4.5* needs more training data to make a reasonable abstraction over larger vectors.

However, there should be an upper limit for performance increase obtained. Note that crossover is random until the end of the first learning period. Therefore, if the learning period is too large, the genetic search may stuck in a local minimum before the control module comes into play. In such a case it is not possible to expect a performance increase.

It would be difficult to formally deduce the exact correlation between the total number of terminal-functional elements used and the learning period. However, a rough interval might be determined experimentally for the optimum period. Certainly, an analysis of the change in performance based on the learning period is needed. Up to now, the performance of the controlled search is tested with three different learning periods. The performance has consistently improved as the learning period is

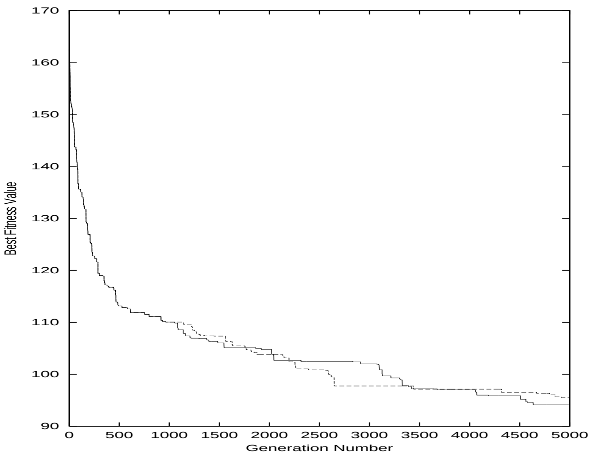

Figure 6.1: Comparison of controlled search and basic GP for the CFG-induction problem. The dashed lines denote the performance of controlled search. Learning period is 1000.

increased. The period is to be increased more in order to determine the upper limit. Hence, new experiments have been carried out in both of the domains with a learning period of 1000.

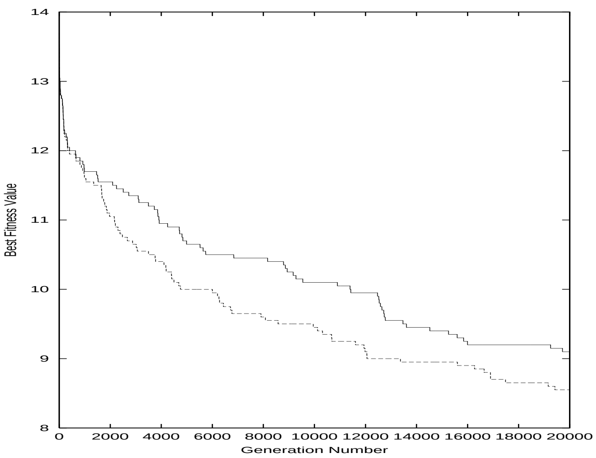

Figure 6.2: Comparison of controlled search and basic GP for the N-Parity problem. The dashed lines denote the performance of controlled search. Learning period is 1000.

In Figures 6.1 and 6.2 the results obtained with this new learning period are presented. The same genetic parameters are used and again the average of twenty different runs are presented in the figures. As seen in Figure 6.1, the performance increase has been lost with the new learning period for the grammar induction problem. In Figure 6.2 the outcome for the N-Parity problem is presented. As seen in the figure, the controlled search can still outperform basic GP, however the performance increase is smaller compared to the increase obtained with the learning period of 500.

There is a difference between the results obtained with this new learning period. In one of the domains the performance increase has been lost totally. In the other domain, the new learning period results in only a small decrease in the performance of the controlled search. In fact this difference can be considered to be quite legitimate. Note that the search carried out in CFG-induction domain continues for 5000 generations. The new learning period 1000 forms %20 of this search process. It is very probable for the convergence to be already close to stabilization after such a large period. Hence, it becomes difficult for the control module to provide beneficial information to improve the process further. The controlled search totally becomes incompetent. On the other side, the new learning period forms only %5 of the search process for the N-parity problem. It can be claimed that the search would still be away from stabilization after the first 1000 generations. Therefore the controlled search is capable of providing a performance increase. However, there is a decrease in performance compared to the period of 500. This denotes that 1000 generations is larger than the optimum learning period for the N-parity problem, too. It can be claimed that 500 generations is roughly close to the optimum learning period for both of the problems.

6.2 Real Time Comparison

The extra processing time required for the control module is a critical issue for the proposed method. The additional cost of the module can be divided into two parts.

The first part is the cost of the learning process which is repeated periodically. Note that the learning period is 500 generations for the successful tests. Hence, the decision tree generator C4.5 is called once in every 500 generation. Researchers have investigated the time complexity of C4.5. [50] denotes that C4.5 produces good classifiers quickly. It is stated that the asymptotic time complexity of C4.5 is $O(ea^2)$, where e is the number of training set elements and a is the number of attributes. The given complexity is for non-numeric data sets. It is denoted that numeric data would require repetitive sorting and hence would add a $loge$ factor at each node of the induced decision tree. It is also stated that empirical determinations show that C4.5's practical time complexity is substantially better than quadratic.

The generation of decision trees requires an additional processing cost at every 500 generations. On the other side, there is an other extra processing cost for every crossover operation that takes place throughout the search. Note that during crossover operation, the control module predicts if the offsprings will be in the positive or the negative class. This procedure includes the formation of the vectors corresponding to the offsprings and checking if the vectors are in the positive or negative class by using the generated decision tree. If both of the offsprings are predicted to be in the positive class at the first attempt, crossover operation is carried out directly. However, if the prediction is negative, then new trials are carried out with new crossover points. Note that the number of trials is limited to be three at most. Empirical analysis of the crossover operation denotes that the average number of trials is about 2.1 for the CFG-induction and 1.8 for the N-Parity problem.

The process of vector formation has been defined in Chapter 3. This process is linear in terms of the total number of nodes on a GP-tree. On the other side, the process of predicting the class of the formed vector is linear in terms of the depth of the decision tree used. The affect of this overhead on the total processing time is related to the fitness function used. For problems with non-linear fitness functions, this overhead could be negligible and the performance increase would probably be more significant. CFG-induction problem is such an example as the fitness function includes the procedure of parsing the training examples.

Note that both of the methods run for the same number of generations in the previous experiments. Certainly the controlled search would in fact take more time than traditional GP because of the the control module. Therefore new experiments are carried out in order to compare the two methods in terms of real CPU time statistics. The performance of traditional GP is observed when the search is allowed to also use the extra time consumed by the control module. First, the real time statistics of the controlled search is determined. The real time consumed by a search varies based on the random seed used. The reason for this variation is due to the change in average size of the chromosomes in different runs. If a chromosome is large then certainly fitness calculation requires more time. If the sizes of the chromosomes in the population tend to increase during a run, then the total running time of the search increases, too. Therefore, the total running time of the controlled search for each

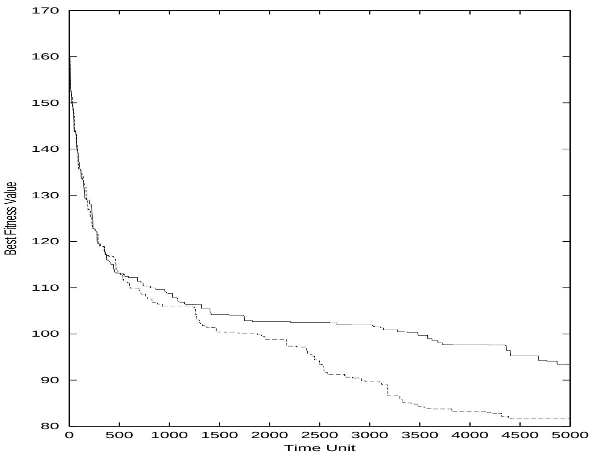

Figure 6.3: Real time comparison of controlled search and basic GP for the grammar induction problem. The dashed lines denote the performance of controlled search. Learning period is 500. The controlled search completes 1 generation in each time unit, whereas basic GP completes 1.04 generations on the average.

seed is determined. Then traditional GP is let to continue its search until the real running time of the corresponding controlled search is exhausted. In these new trials, traditional GP runs have lasted for about 5200 generations on the average, for the CFG-induction problem. This means that the time consumed by the control module corresponds to the time needed for the evaluation of 200 generations traditionally. This can not be considered as an heavy cost. The situation denotes that the fitness evaluation which is a non-linear process is dominant in the total running time for the CFG-induction problem. The same procedure is repeated for N-parity problem, too. Note that fitness evaluation is linear for the problem. Hence, the extra processing time consumed by the control module should be more significant in this domain. It has been observed that traditional GP lasts for about 28000 generations on the average when the search is allowed to use the extra time.

In figures 6.3 and 6.4 the comparison of the obtained results on both of the domains are presented. In this comparison, the x-axis denotes the time consumed by the search. The time unit used in the figures is set as the total time that the controlled search needs to complete a single generation. Obviously, traditional GP would complete more than one generation in one unit. This is 1.04 generations in the CFG-induction domain and 1.4 generations in the N-parity domain. As seen in figure 6.3 the controlled search can still outperform traditional GP. It is not possible for the traditional GP to converge to the same level with the controlled search even it is allowed to use the extra processing

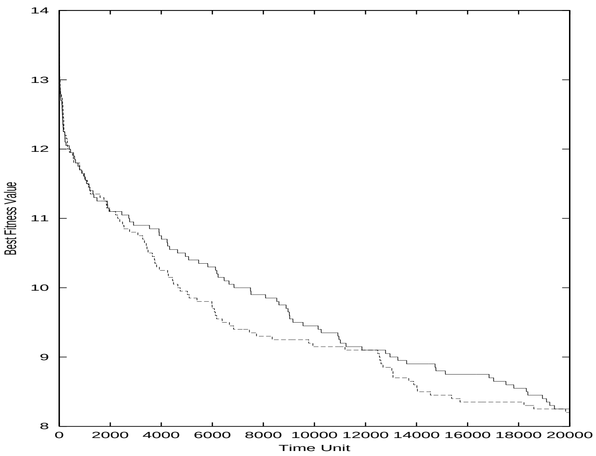

Figure 6.4: Real time comparison of controlled search and basic GP for the N-Parity problem. The dashed lines denote the performance of controlled search. Learning period is 500. The controlled search completes 1 generation in each time unit, whereas basic GP completes 1.4 generations on the average.

time used by the control module. On the other side, the situation is a bit different for the N-parity problem. As seen in figure 6.4, traditional GP can converge to the same level when the extra time is used. However, it can be claimed that, convergence of the controlled search is still more efficient in general.

6.3 Maximum Generation Number

It is possible to consider the evolution in GP as an infinite process. However due to practical needs, the search process has to be terminated at some point. The *Termination Criterion* specifies when a GP run will be stopped.

For some problems, this criterion can be defined as the satisfaction of a success predicate. This predicate would obviously be either finding the optimum solution or approaching to the solution more than a predetermined interval. However, it might be impossible to define the optimal solution in some domains. Optimization problems and model prediction using noisy data are examples for such a case. In these domains an upper limit on the number of generations is set as the termination criterion and the results are analyzed after GP run reaches this limit.

For some deceptive problems, it might be possible to define the global optimum. However, the genetic search has quite a low performance in such domains and it is impractical to let the GP run to continue its search until the solution is found. An upper limit has to be determined for the number of generations in this case too. However no theoretical explanation exists which may be helpful to determine this upper limit. Researchers empirically visualize the change of the best fitness value and determine the upper limit based on convergence obtained throughout the search. The search is stopped when no further significant improvement is obtained in a reasonable period.

Table 6.1: Performance of basic GP for CFG-induction problem in five different periods. Each period is 1000 generations.

Period #	**Decrease in fitness**	**Percentage of the Decrease**
Period 1	59.0	%36.2
Period 2	6.8	%4.2
Period 3	2.3	%1.4
Period 4	3.0	%1.8
Period 5	2.7	%1.7

The two testbed problems we have used are cases where it is possible to define the optimum solution. In both of the domains, the GP search is far away from reaching to the optimal solution in a practical amount of time. However, in the CFG-grammar induction domain the maximum number of generations is set as 5000, whereas in the N-parity domain this number is 20000. This difference depends on the empirical observations on the best fitness change throughout different runs. In Table 6.1 the change of the best fitness mean in five different periods of the GP search is presented for the CFG-induction domain. Each period is 1000 generations. Hence, period 1 denotes the improvement obtained after the first 1000 generations, period 2 presents the improvement between $1000th$ and $2000th$ generations and so on. The numbers in the second column are the absolute decrease in the best fitness mean and the third column denotes the percentage of this decrease compared to the best fitness mean obtained in the initial random population. Note that in Table 6.1, there is a dramatic decrease in the best fitness mean in the first period. However, after the second period the convergence seems to have stabilized and the decrease in best fitness mean stays below %2 in the last three periods. Similar low performance in three consecutive periods denotes that the probability of obtaining a dramatic decrease in the following generations would be quite low and it has been decided to use 5000 generations as the as the upper limit in this domain.

Table 6.2: Performance of basic GP for N-parity problem in four different periods. Each period is 5000 generations.

Period #	**Decrease in fitness**	**Percentage of the Decrease**
Period 1	3.2	%23.4
Period 2	0.5	%3.6
Period 3	0.6	%4.4
Period 4	0.4	%2.9

In the N-parity domain, the maximum number of generations has been set as 5000 for the initial trials. However it has been observed that the convergence obtained is quite low at the end of the search, compared to the CFG-induction domain. In Table 6.2 the change in best fitness mean in four different periods are presented. Note that each period is 5000 generations. The progress obtained in the first period is even less than the performance in the first 1000 generations of the CFG-induction domain. Therefore, it is decided to increase the maximum number of generations for the N-parity problem. Three new periods of 5000 generations are added in this domain reaching to the total number of 20000 generations. As seen in Table 6.2 the convergence obtained in the consecutive three periods is quite low compared to the progress in the first period. Similar to the reasoning used in the CFG-induction domain, low performance in three periods has led us to the idea of fixing the maximum generation number as 20000 in the N-parity domain.

CHAPTER 7

$TESTBED_3$: DESIGN AND USAGE OF A NEW BENCHMARK PROBLEM

The proposed approach aims to provide a mechanism to increase the performance of GP. However, it is claimed that the method is expected to be significant for deceptive problems. The reason underlying this claim depends on the characteristics of the approach proposed. The control module gains experience about the regularities of the search space and enables the GP search to keep away from the low-fit areas. Such an experience would be meaningful when it is difficult to obtain a convergence. If the search can quickly converge to a local or a global optima, the guidance of the control module loses importance. The learning process that takes place in the control module qualifies when the search process fluctuates in an discontinuous search space. Such an uncontrolled wandering in the search space appears for deceptive problems.

In the previous chapters the method is tested on two-real world problems. Obviously obtaining performance increase in only two domains is not adequate to verify the above claim. In order to get more insight about the contribution of the control module, it is decided to use a tunable benchmark problem. By applying the method on different instances of the same problem, it is aimed to verify that the significance of the method increases as the problem is made more deceptive.

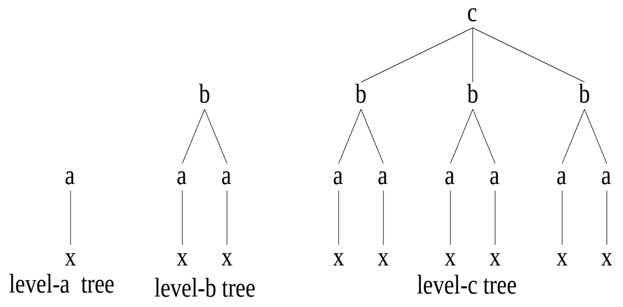

Figure 7.1: Perfect trees of different levels.

7.1 Choosing a Benchmark Problem

Not so many benchmark problems have been proposed in the area of GP. The benchmark problem proposed in [51] is notable since their formalization shares some characteristics of the royal road problem. Royal road problem is a commonly used benchmark problem for tuning the genetic parameters in GA field [27]. Their formalization is based on the definition of *"perfect tree"* of some depth. For instance a level-a tree is perfect when its root is the function a with a single child. Similarly a perfect level-b tree's root should be function b having two perfect level-a trees as children. Certainly a perfect level-c three will have three perfect level-b trees connected to root c and so on. The terminal set consists of a single element x. Note that the series of functions $a, b, c, d...$ are defined with increasing arity. The perfect trees of levels a, b and c are presented in Figure 7.1.

The raw fitness of a tree is defined as the score of its root. On the other side the score of each function is calculated by adding the weighted scores of its children. The weight is a constant larger than one ($FullBonus$) if a child is a perfect tree of the appropriate level. When the child is not a perfect tree, the weight turns out to be a constant smaller or equal to one ($Penalty$ or $Partialbonus$) depending on the child's configuration. If the child has the correct root $Partialbonus$ is used. However, if the root of the child is incorrect too, $Penalty$ is used as the weight. On the other side, if the root itself is the correct root of a perfect tree, then the obtained score is multiplied by $CompleteBonus$. It is stated that typical values that can be used are: $FullBonus = 2, PartialBonus = 1, Penalty = \frac{1}{3}$, and $CompleteBonus = 2$. It is suggested that perfect trees of different depths can be used as benchmark problems for GP.

The proposed definition is quite problematic in terms of our testing purposes. Our main focus is to be able to control the degree of deception in the problem. Deception usually appears as an outcome of epistasis. That is the interdependency among the subparts of the chromosome should have an influence on the global fitness of the chromosome. However, the formalization provided by [51] enables each child to contribute to the global fitness independent of other children. What is more, since the functions are defined with increasing arity, the search space increases rapidly for high levels. The problem is too difficult after level d and e. On the other side levels a, b and c are too simple. It is possible for the solution to appear in the initial random population in these levels. Lastly using more than one constant makes the definition unnecessarily complicated. Therefore it has been decided to simplify and reorganize the definition of perfect tree. The aim is to obtain a tunable benchmark problem where epistasis can easily be controlled.

7.2 A New Benchmark Problem

The new perfect tree is defined to be a full binary tree of some depth. The function set consists of n functions with arity two ($F = \{X_1, X_2, ..X_n\}$). The terminal set consists of a single terminal element t, ($T = \{t\}$). The raw fitness of a tree is again defined as the score of its root. The fitness of a terminal node is simply defined as 1. The fitness of an internal node is defined as the sum of the fitnesses of its two children. The two constraints used to create epistasis for the problem are the following.

When the children of an internal node are not terminal elements:

(i) The index of the parent function should be smaller than the index of its children.

(ii) The index of the right-hand child should be larger than the index of the left-hand child.

The fitness function for an internal node is defined as:

$$f(P) = \begin{cases} f(C_1) + f(C_2) & [1] \\ C_{epis} \cdot f(C_1) + f(C_2) & [2] \\ f(C_1) + C_{epis} \cdot f(C_2) & [3] \\ C_{epis} \cdot f(C_1) + C_{epis} \cdot f(C_2) & [4] \end{cases} \tag{7.1}$$

In Equation 7.1, part [1] is chosen if both of the children are terminal elements or none of the constraints are violated. If the first constraint is violated by any child, the fitness of that child is multiplied with a constant smaller than one. This constant is called the *epistasis constant* (C_{epis}). Parts [2] and [3] in Equation 7.1 reflect this situation. Lastly part [4] is used when both of the children violate the first constraint or when the second constraint cannot be achieved.

Note that these two constraints create interdependency among the subparts of a chromosome. When the epistasis constant is decreased the interdependency increases. It becomes impossible for a subpart of the chromosome to contribute to the global fitness independent of other parts. A well-fit chromosome can easily be ruined when a node that violates a constraint appears after a recombination operation. Hence, fitnesses of similar chromosomes might differ remarkably and the search space becomes discontinuous. What is more when the function set is kept small, it becomes more probable to be stuck in a local minima. Note that the index of the functions should increase as you go down in a chromosome. Therefore a wrong choice close to the root might make it impossible to form a perfect tree.

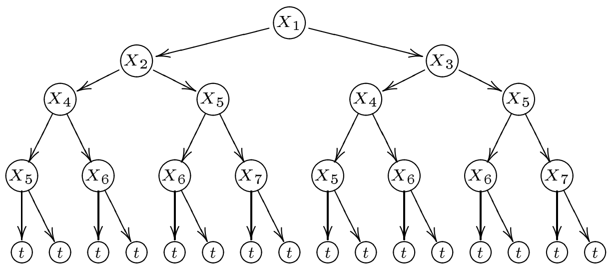

Figure 7.2: A possible solution of depth four.

For the testing phase the function set is fixed as $F = \{X_1, X_2, ..., X_8\}$ and the terminal set is $T = \{t\}$. Also the depth of the perfect tree to be searched is determined as 4. In Figure 7.2 a possible solution is given for depth 4. Here the function X_8 has not been used, however other solutions exist that would include the eighth function too.

Note that two parameters exist for controlling the difficulty of the problem. When you enlarge the function set, the number of possible solutions increases. On the other side when the epistasis constant is decreased the interdependency increases and the problem turns out to be more deceptive. Our focus has been on the interdependency

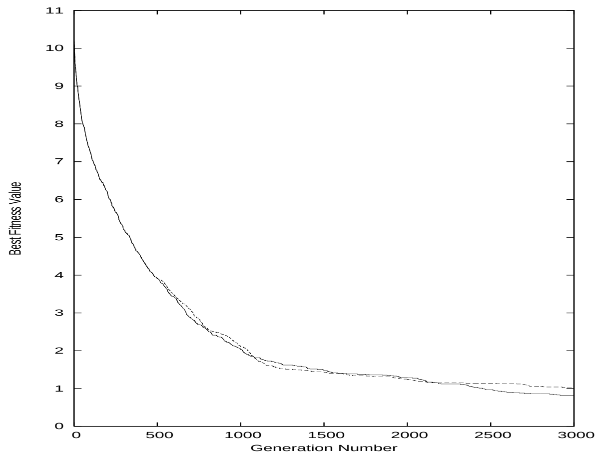

Figure 7.3: Comparison of controlled search and basic GP. The dashed lines denote the performance of controlled search. Learning period is 500. Epistasis constant is 0.5

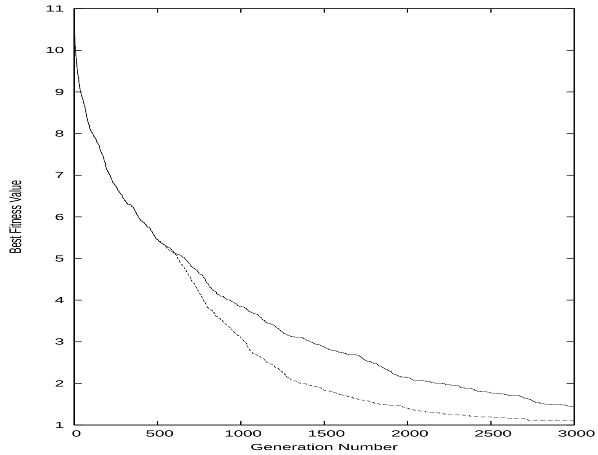

Figure 7.4: Comparison of controlled search and basic GP. The dashed lines denote the performance of controlled search. Learning period is 500. Epistasis constant is 0.35

among the subparts of a chromosome. Therefore the function set is kept the same and our new approach is tested with different epistasis constants.

The different epistasis constants used for the testing process are 0.5,0.35, 0.2, 0.05 and 0.002. The genetic parameters are set as follows

- Population size = 100
- Crossover at function point fraction = 0.1
- Crossover at any point fraction = 0.7
- Reproduction fraction = 0.1
- Mutation fraction = 0.1

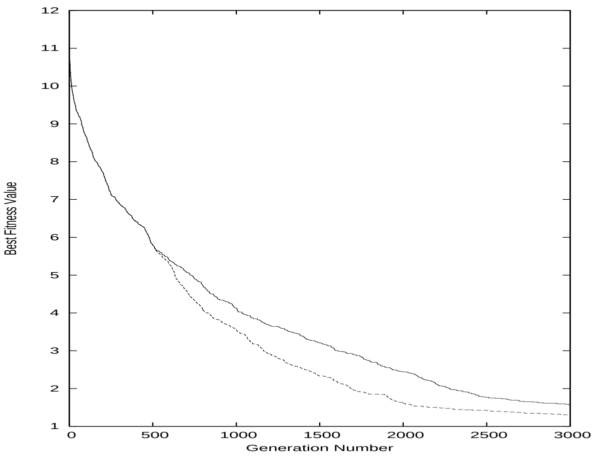

Figure 7.5: Comparison of controlled search and basic GP. The dashed lines denote the performance of controlled search. Learning period is 500. Epistasis constant is 0.2

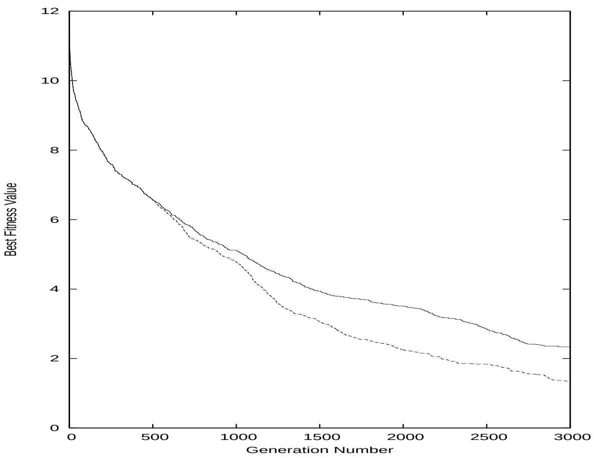

Figure 7.6: Comparison of controlled search and basic GP. The dashed lines denote the performance of controlled search. Learning period is 500. Epistasis constant is 0.05

- Number of Generations = 3000
- Selection Method: Fitness Proportional.

In Figures 7.3, 7.4, 7.5, 7.6, 7.7 the comparison of controlled search and basic GP for each epistasis constant are presented. The results are obtained by using the average of 100 different runs. Note that the largest value that can be obtained by the fitness function proposed in Equation 7.1 is 16. The best fitness value is again set as zero. Therefore the real fitness function used for the experiments would be

$$\hat{f}(T) = 16 - f(Root(T)), \tag{7.2}$$

where f is the function in Equation 7.1, T is a tree and $Root$ returns the root node of

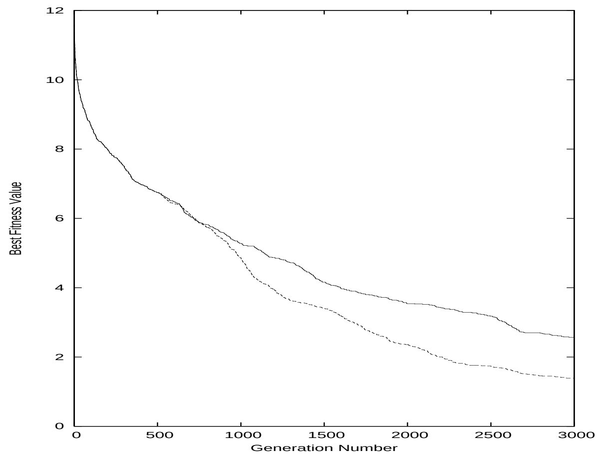

Figure 7.7: Comparison of controlled search and basic GP. The dashed lines denote the performance of controlled search. Learning period is 500. Epistasis constant is 0.002

a given tree. As seen in Figure 7.3, it is not possible to obtain an improvement when the epistasis constant is 0.5. However, as the problem is made more deceptive by decreasing the epistasis constant, the performance increase becomes more significant as seen in Figures 7.4, 7.5, 7.6 and 7.7. The behavior of basic GP and the controlled search can be best seen in Figure 7.8. In this figure the comparison of the best fitness means obtained at the end of 3000 generations is presented for each epistasis constant.

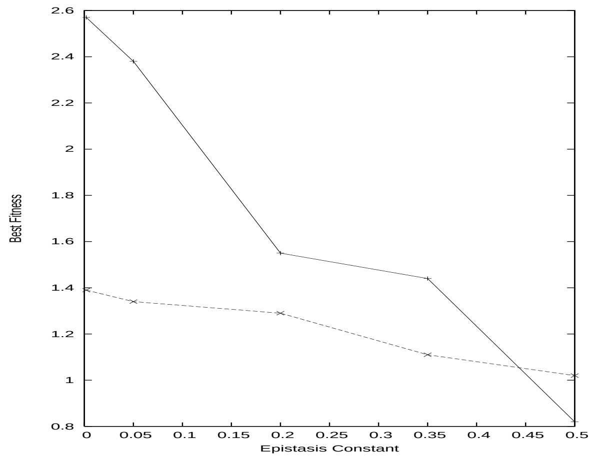

Figure 7.8: Best fitness means obtained at the end of 3000 generations.The dashed lines denote the performance of the controlled search.

As seen in the figure, the controlled search is misleading for the epistasis constant 0.5. Basic GP can achieve a better average at the end of the search. However, as the epistasis constant is decreased the performance of basic GP rapidly goes down. On the

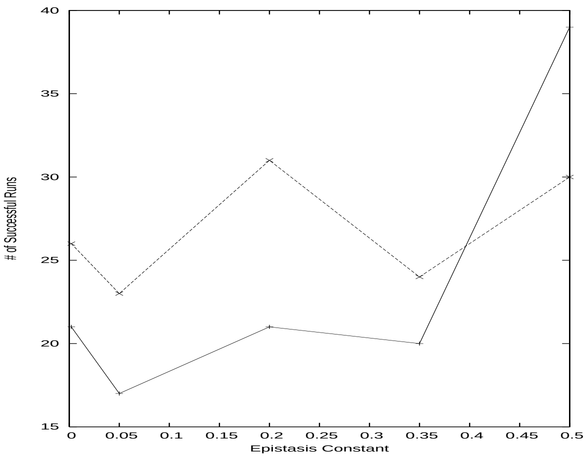

Figure 7.9: Number of successful runs for each epistasis constant. The dashed lines denote the performance of the controlled search.

other side the controlled search can compensate the deception and the performance decrease is not dramatic as deception increases. It can be claimed that the results obtained are consistent with our proposal and the significance of controlled search becomes more apparent with high deception. Also in Figure 7.9 the number of successful runs achieved are presented for each epistasis constant. Again the results are as expected. For the epistasis constant 0.5 basic GP can find more solutions than the controlled search and again as the epistasis constant decreases the controlled search outperforms basic GP.

CHAPTER 8

TESTING THE LIABILITY OF THE RESULTS OBTAINED

The results presented in the previous chapters are the best fitness means obtained by a certain number of sample runs. This is a common comparison method used in GP community. Usually $20 - 30$ runs are excepted to be enough for a satisfactory comparison between two methods. However, this is quite questionable in terms of statistics. Statistical significance between two means depends on two parameters. These are population size (number of sample runs for our case) and the variance in the population. Therefore it is not possible to determine a general lower limit for the number of sample runs that would be satisfactory for all problems. This lower limit would change depending on the standard deviation that would appear throughout the runs. Therefore statistical tests are crucially needed to determine if the number of runs used are enough for a satisfactory comparison between the methods.

Although this approach is not common in evolutionary computation community yet, some researchers have started to highlight the importance of the subject. [46] denotes that performance comparison is an important subject in GP research, since many published research includes the comparison of one technique with another. [46] also states that of the 22 papers examined, 16 used only visual comparison of the graphs.

It is important to comprehend the notion of *Statistical Significance*. Consider the two situations presented in Figure 8.1. The presented graphs denote the distribution

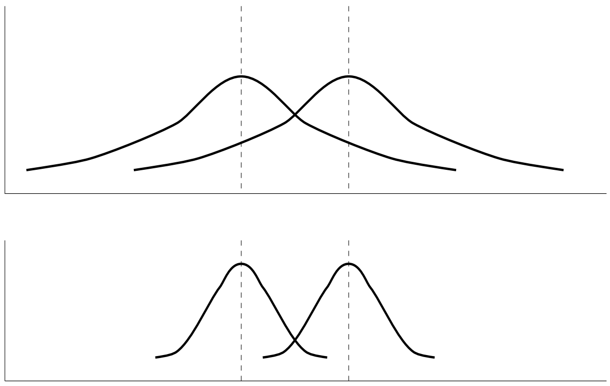

Figure 8.1: Two different scenarios for the differences between the means.

of elements belonging to two different populations. The first thing to be noted about the graphs is that the difference between the two means is the same in both of them. However, the distribution of the elements are clearly different from each other. In the first graph the variability of the elements is quite high and the two populations overlap so much. In the second case variability is smaller. Clearly, it can be claimed that the populations in the second graph seem to be more different or distinct. It is reasonable to state that if population sizes are the same, then obtaining such a difference between the two populations by chance would be more probable for the situation presented in the first graph, rather than the situation in the second one. This leads us to the idea that while judging the difference between two means, the variability of the elements has to be taken into consideration, too. On the other side, the number of elements used to determine the distribution is obviously another factor that would affect the judgment on the difference between the two means. Hence, the statistical significance between two means should be a measure based on the variability and the sizes of the populations relative to the difference between the means.

T-test is offered as a statistical method that can be used for comparing small samples, [46]. The formulation for calculating the t value is given in [18] as follows.

$$t = \frac{M_1 - M_2}{\sqrt{\frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{n_1+n_2-1} \cdot (\frac{1}{n_1} + \frac{1}{n_2})}} \tag{8.1}$$

In Equation 8.1, M_1 and M_2 are the two means to be compared, n_1 and n_2 denote the population sizes used to obtain the corresponding means and lastly s_1, s_2 are the standard deviations of the two populations. The t-value turns out to be a ratio of the difference between the means to some value calculated based on the population sizes and the deviation in the populations. The value obtained by the equation corresponds to a risk level depending on the degrees of freedom and the initial hypothesis used for the testing. Our hypothesis is to determine if there is really a difference between the means obtained by the two approaches. Therefore we have used the two-tailed version of the test, [46]. The degrees of freedom is defined as the sum of the samples in both populations minus two. The risk level denotes the probability of obtaining the difference between the two means by chance. In statistics usually a risk level smaller than 0.05 is considered to be statistically significant.

In order to apply the test on our method, the t-value for the difference between the two means are calculated for each generation. Then using an automated tool corresponding risk levels are obtained. Note that there is no difference between the two means until the end of the first learning period. Therefore the risk level would be 1 before the controlled search starts. After the first learning period, the risk level is expected to decrease below 0.05 in a certain amount of generations.

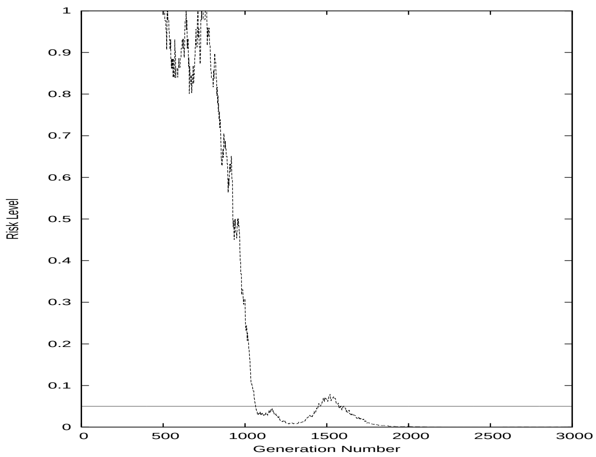

Figure 8.2: The t-test for the benchmark problem. Epistasis constant is 0.002.

The test is first applied on the instances of the benchmark problem except the instance with epistasis constant 0.5. Note that no improvement has been obtained for that case. The number of sample runs that have been used for this problem is 100. This can be considered as quite a big population for the test and as expected all of the four instances of the problem was able to pass the test. It has been possible to obtain risk levels smaller than 0.01 towards the end of the search. This denotes that the performance increase obtained is statistically significant with probability larger than 0.99.

In Figure 8.2 the change of the risk level for the instance with epistasis constant 0.002 is presented. As the two means start to differ after generation 500, the risk level dramatically decreases. The statistical significance is obtained after about 1000 generations.

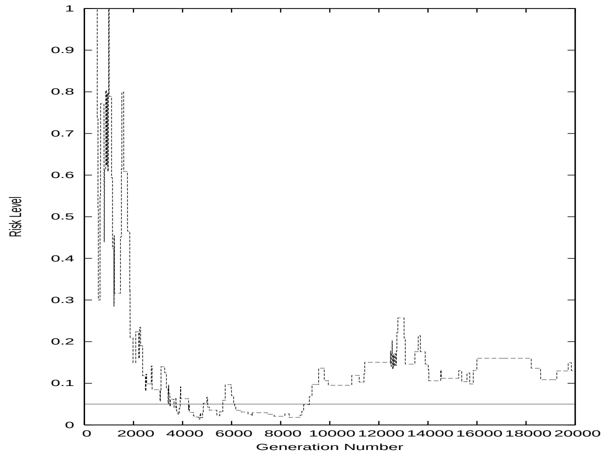

Figure 8.3: The t-test for the N-Parity problem. Learning Period is 500 and number of sample runs is 20.

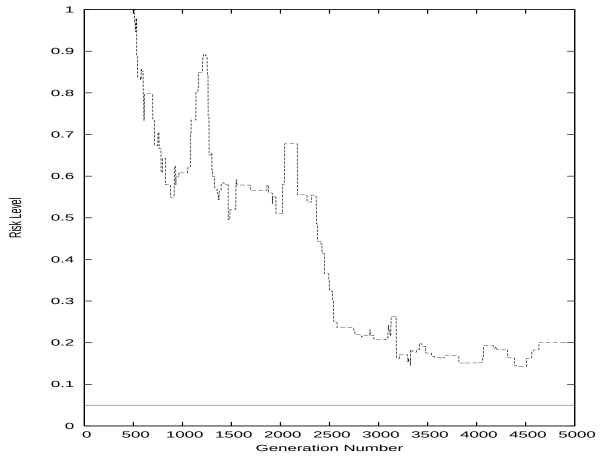

Figure 8.4: The t-test for the CFG induction problem. Learning Period is 500 and number of sample runs is 20.

The test has also been applied to the two real world domains. Note that the number of sample runs used in these domains is 20. This can be considered as a small population in terms of statistics. So it is more probable to encounter problems with such a population size. In Figures 8.3 and 8.4 the results of the test are presented. For the N-Parity problem the risk level goes below the critical value 0.05 around generation 4000. However, the statistical significance is lost around generation 9000. This is due to the increase in standard deviation in the second part of the search. The situation for the CFG-induction problem is even worse. The risk level stays above 0.2 all through the search. This is again due to the high standard deviation all through the search.

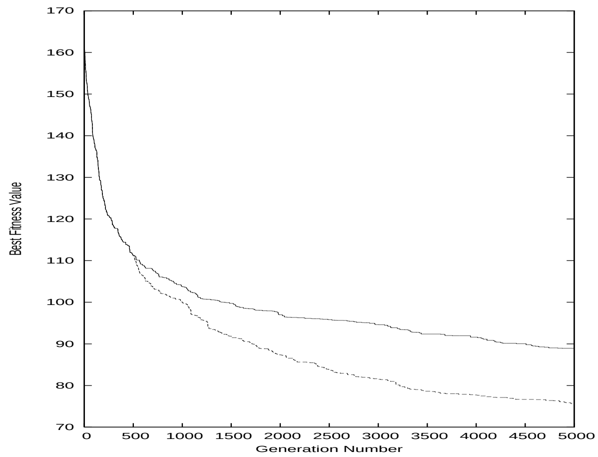

Figure 8.5: Comparison of the means when number of sample runs are increased to 80 for the CFG induction problem. Learning Period is 500.

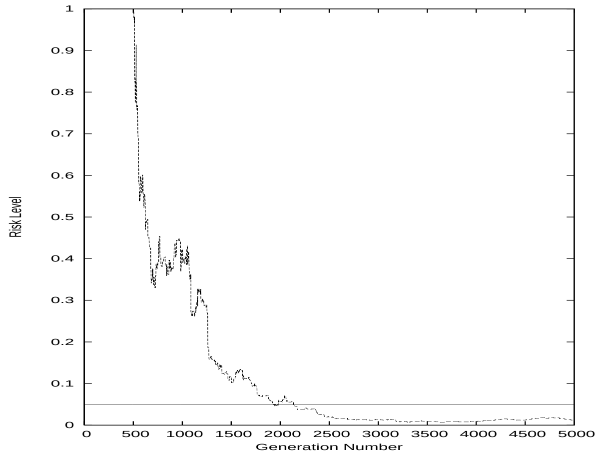

Figure 8.6: The t-test for the CFG induction problem. Learning Period is 500 and number of sample runs is 80.

The results obtained for the two real world domains denote that the number of sample runs used is not enough to statistically prove that a performance increase has really been achieved. It should be noted that the results obtained do not point out that the controlled search cannot lead to a performance increase. It is the case that the number of sample trials used are not enough for a definite decision and there is the risk of losing the performance increase when larger number of trials are carried out. More sample trials have to be included for a definite judgment about the method. Certainly, the number of extra sample runs needed depends on the standard deviation that appears during the runs.

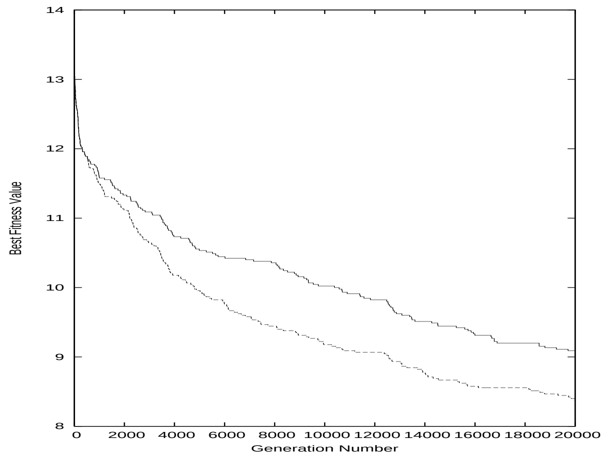

Figure 8.7: Comparison of the means when number of sample runs are increased to 45 for the N-parity problem. Learning Period is 500.

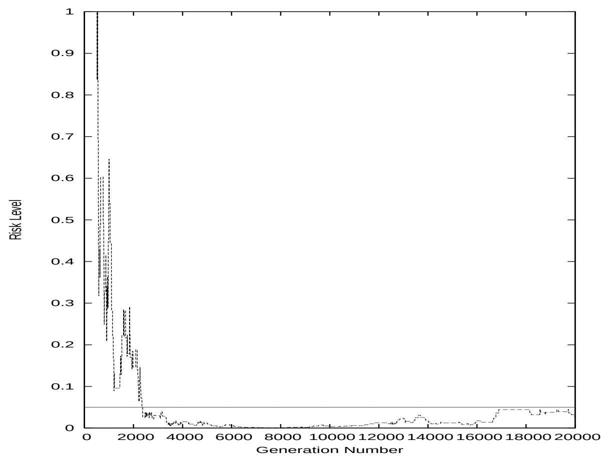

Figure 8.8: The t-test for the N-Parity problem. Learning Period is 500 and number of sample runs is 45.

For instance for the grammar induction problem the number of sample runs has been increased to 40 as a first step. However, still the desired significance has not been obtained. The results of the test were not satisfactory until the number of sample runs has reached to 80. On the other side 45 sample runs have been enough to obtain statistical significance for the N-parity problem. In Figures 8.5 and 8.7 the comparison of the means obtained with the new population sizes are presented for the two problems. It should be noted that the performance increase obtained by the controlled search is similar to the ones presented in the previous sections. The difference between the two means has not altered significantly compared to the mean differences obtained by less number of runs. However, this time statistical significance has been obtained as seen in Figures 8.6 and 8.8. Hence, it can be concluded the controlled search can provide a performance increase in the two real world domains, too.

CHAPTER 9

CONCLUSION AND FUTURE WORK

Genetic programming is a convenient search algorithm for symbolic problems. However the method easily becomes impractical for a certain class of problems. The focus of this thesis was proposing a method to increase the performance of GP on such problems. An important aspect of the study is the attempt to design a self referential system, which can deduce beneficial knowledge from its own experience. It is aimed to use this information to control recombination afterwards.

The second important aspect of the study is based on the fact that GP is only an effective way of implementing the *Divide and Conquer* method. This approach is questionable for deceptive problems. The global meaning of finding a possible solution goes beyond determining the sub solutions and bringing them together. Therefore, the self referential information collected during the search is based on the global structure of the chromosomes. A simple formalization is proposed for the representation of this global information. The proposed method does not have a heavy computational load. This is one of the reasons for choosing a simple formalization. What is more, the representation presented has the capability of holding two important global characteristics of a chromosome; the frequency information of the elements and their position in the chromosome. Certainly other attempts can be carried out with different variations of the proposed representation. On the other side, note that decision tree generator *C4.5* is the only machine learning tool used for inducing the models for the well-fit chromosomes. It is possible to replace decision tree learning with other machine learning methods like Neural Networks. It is open to discussion if decision tree learning is the best choice for the current task. However, the thesis can be seen as an initial attempt aiming to improve the performance of GP by using a self referential system, which focuses on the global information of chromosomes. Hence, our main concern has been providing detailed empirical evidence for the success of the proposed approach. Therefore, the subjects mentioned above still remain as open research points.

The proposed approach has been applied to two real world-domains. Both of the selected domains are highly deceptive. The search spaces are discontinuous and the partial success of a chromosome is fragile during recombination in both domains. The selected problems are quite suitable for the research purposes of the study. It is a fact that the proposed approach cannot provide a total solution for the problems at hand. However, the performance increase obtained in two different domains provides strong evidence about the success of extracting beneficial self-referential knowledge from the search process. It can be claimed that the proposed approach is on the right track and open to further improvement.

Certainly, the performance increase obtained in only two domains is not sufficient to claim a general improvement for GP. However, the experiments carried out on the benchmark problem provides an insight about the contribution of the approach. The different instances of the problem has made it possible to carry out a controlled experiment to trace the behavior of the new approach. The results obtained clearly denote that the performance decrease that occurs due to epistasis can be blocked by the control module. The contribution of the module becomes significant when the interdependency among the subparts of chromosomes is increased. It can be claimed that the results are consistent with the initial ideas deduced from the results obtained in the two real-world domains.

An important notion about the proposed method is determining the optimum learning period for the problem at-hand. Providing a theoretical proof for determining this optimum point would be outside the limits of this thesis. However, a rough interval has been proposed empirically for both of the problems. Another critical question about the method proposed is about the extra processing time required for the control module. The performance of traditional GP is also analyzed, when the search is allowed to also use the extra time consumed by the control module. It has been observed that the overhead of the control module depends on the fitness function used. For the CFG-induction problem, this overhead is almost negligible as the fitness function is non-linear. For the N-parity problem the overhead is significant. However the control module is still capable of providing a more efficient search.

The statistical tests form an important part of the thesis. Verification of the results statistically is not a common approach in evolutionary computation community yet. However, it is obvious that visual comparison of means is not sufficient for a sound conclusion about the results obtained.

Our initial question was, if it could be possible to extract information during the genetic evolution and use this information to control the search process afterwards. Although the proposed approach is open to further research and development, the content of the research carried out clearly points out that the aimed approach is possible.

REFERENCES

[1] Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Genetic programming and deductive-inductive learning: A multistrategy approach. In Jude Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning, ICML'98*, pages 10–18, Madison, Wisconsin, USA, July 1998. Morgan Kaufmann.

[2] David Andre. Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 23, pages 477–494. MIT Press, 1994.

[3] Peter John Angeline. Genetic programming and emergent intelligence. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 4, pages 75–98. MIT Press, 1994.

[4] Shumeet Baluja and Scott Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proc. 14th International Conference on Machine Learning*, pages 30–38. Morgan Kaufmann, 1997.

[5] Wolfgang Banzhaf. Genetic programming for pedestrians. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, page 628, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.

[6] Eugene Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 18(4):33–44, 1997.

[7] Eugene Charniak. A maximum-entropy-inspired parser. Technical Report CS-99-12, Department of Computer Science, Brown University, August 1999. Wed, 4 Aug 1999 17:39:56 GMT.

[8] Michael John Collins. A new statistical parser based on bigram lexical dependencies. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 184–191, San Francisco, 1996. Morgan Kaufmann Publishers.

[9] Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In John J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA, 24-26 July 1985.

[10] Sumit Das, Terry Franguidakis, Michael Papka, Thomas A. DeFanti, and Daniel J. Sandin. A genetic programming application in virtual reality. In *Proceedings of the first IEEE Conference on Evolutionary Computation*, volume 1, pages 480–484, Orlando, Florida, USA, 27-29 June 1994. IEEE Press. Part of 1994 IEEE World Congress on Computational Intelligence, Orlando, Florida.

[11] Edwin D. de Jong, Richard A. Watson, and Jordan B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 11–18, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[12] Patrik D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[13] Huntington Cairns (Editor) Edith Hamilton (Editor). *The Collected Dialogues of Plato*. Princeton University Press, 1961.

[14] J. Eggermont and J. I. van Hemert. Stepwise adaptation of weights for symbolic regression with genetic programming. In *Proceedings of the Twelveth Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'00)*, 2000.

[15] Gary William Flake. *The Computational Beauty of Nature*. The MIT Press, Cambridge, Massachusetts, 1998.

[16] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. Artificial intelligence through a simulation of evolution. In M. Maxfield, A. Callahan, and L. J. Fogel, editors, *Biophysics and Cybernetic Systems: Proc. of the 2nd Cybernetic Sciences Symposium*, pages 131–155, Washington, D.C., 1965. Spartan Books.

[17] Richard Forsyth. BEAGLE A Darwinian approach to pattern recognition. *Kybernetes*, 10:159–166, 1981.

[18] Gary A. Freund, John E. Simon. *Modern Elementary Statistics*. Prentice Hall, 1992.

[19] Cory Fujiki and John Dickinson. Using the genetic algorithm to generate lisp source code to solve the prisoner's dilemma. In John J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the second international conference on Genetic Algorithms*, pages 236–240, MIT, Cambridge, MA, USA, 28-31 July 1987. Lawrence Erlbaum Associates.

[20] Chris Gathercole and Peter Ross. Tackling the boolean even N parity problem with genetic programming and limited-error fitness. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 119–127, Stanford University, CA, USA, 13-16 1997. Morgan Kaufmann.

[21] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learing.* Addison Weslay, 1989.

[22] Alan Woods & Ted Grant. *Reason in Revolt - Marxist Philosophy and Modern Science.* Wellred Publications, 1995.

[23] Simon Handley. Automatic learning of a detector for alpha-helices in protein sequences via genetic programming. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 271–278, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.

[24] John Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[25] Tzung-Pei Hong. Evolution of appropriate crossover and mutation operators in a genetic process. *Applied Intelligence*, 16(1):7–17, 2002.

[26] Hitoshi Iba and Hugo de Garis. Extending genetic programming with recombinative guidance. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 4, pages 69–88. MIT Press, Cambridge, MA, USA, 1996.

[27] Terry Jones. A description of holland's royal road function. *Evolutionary Computation*, 2(4):409–415, 1995.

[28] Hugues Juille and Jordan B. Pollack. A sampling-based heuristic for tree search applied to grammar induction. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) Tenth Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, Madison, Wisconsin, USA, 26-30 1998. AAAI Press Books.

[29] H. Kargupta. Revisiting the gemga: Scalable evolutionary optimization through linkage learning. In *Proceedings of 1998 IEEE International Conference on Evolutionary Computation*, pages 603–608. IEEE Press, 1998.

[30] Mike J. Keith and Martin C. Martin. Genetic programming in C++: Implementation issues. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 13, pages 285–310. MIT Press, 1994.

[31] B. Keller and R. Lutz. Evolving stochastic context-free grammars from examples using a minimum description length principle. In *Proceedings of the Workshop on Automata, Inductive Grammatical Inference and Language Acquisition. ICML-97*, 1997.

[32] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[33] John R. Koza. The genetic programming paradigm: Genetically breeding populations of computer programs to solve problems. In Branko Soucek and the IRIS Group, editors, *Dynamic, Genetic, and Chaotic Programming*, pages 203–321. John Wiley, New York, 1992.

[34] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge Massachusetts, May 1994.

[35] Jeremy Kubica and Eleanor Rieffel. Collaborating with A genetic programming system to generate modular robotic code. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 804–811, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[36] George Lakoff. *Women, Fire, and Dangerous Things.* Chicago University Press, Chicago, Mi, 1987.

[37] William B. Langdon and Adil Qureshi. Genetic programming – computers using "natural selection" to generate programs. Research Note RN/95/76, University College London, Gower Street, London WC1E 6BT, UK, October 1995.

[38] Lucas. Structuring chromosomes for context-free grammar evolution. In *IEEE-CEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 1994.

[39] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[40] David J. Montana. Strongly typed genetic programming. Technical Report #7866, 10 Moulton Street, Cambridge, MA 02138, USA, 7 1993.

[41] Heinz Muhlenbein and Gerhard PaaB. From recombination of genes to the estimation of distributions: I. binary parameters. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving From Nature-PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 178–187. Springer-Verlag, Berlin, 1996.

[42] Peter Nordin. A compiling genetic programming system that directly manipulates the machine code. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 14, pages 311–331. MIT Press, 1994.

[43] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[44] Una-May O'Reilly. *An Analysis of Genetic Programming.* PhD thesis, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada, 22 September 1995.

[45] Una-May O'Reilly and Franz Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 73–88, Estes Park, Colorado, USA, 31 July–2 August 1994 1995. Morgan Kaufmann.

[46] Norman Paterson and Michael Livesey. Performance comparison in genetic programming. In Darrell Whitley, editor, *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 253–260, Las Vegas, Nevada, USA, 8 July 2000.

[47] Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz. Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation*, 9(4):311–340, 2000.

[48] Martin Pelikan and Heinz Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. K. Chawdhry, editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, London, 1999. Springer-Verlag.

[49] Riccardo Poli and William B. Langdon. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252, 1998.

[50] F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, 3, 1999.

[51] William F. Punch, Douglas Zongker, and Erik D. Goodman. The royal tree problem, a benchmark for single and multiple population genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 15, pages 299–316. MIT Press, Cambridge, MA, USA, 1996.

[52] J. R. Quinlan. *C4. 5: Programs for Machine Learning*. MK, San Mateo, CA, 1993.

[53] J. P. Rosca and D. H. Ballard. Learning by adapting representations in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence, Orlando, Florida, USA*, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[54] Hans-Paul Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 1975. German.

[55] T.C. Smith and I.H. Witten. A genetic algorithm for the induction of natural language grammars. In *Proceedings of IJCAI-95 Workshop on New Approaches to Learning for Natural Language Processing*, pages 17–24, Montreal, Canada, 1995.

[56] Thomas A. Sudkamp. *Languages and Machines*. Addison-Wesley, Reading, MA, 1988.

[57] Walter Alden Tackett. Genetic programming for feature discovery and image discrimination. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 303–309, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.

[58] George Allen & Unwin. Translated by A. V. Miller. *Hegel's Science of Logic*. 1969.

[59] A. M. Turing. *Computing Machinery and Intelligence.* McGraw-Hill, 1963.

[60] J. Von Neumann. *The Computer and the Brain, Silliman Lectures.* Yale University Press, New Haven, CT, 1958.

[61] J. von Neumann. A system of 29 states with a general transition rule. In A. Burks, editor, *Theory of Self-Reproducing Automata*, pages 305–317. University of Illinois Press, 1966.

[62] M. Mitchell Waldorp. *Complexity: The Emerging Science at the Edge of Order and Chaos.* Simon and Shuster/Viking., 1992.

[63] P. A. Whigham. Grammatically-based genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA, 9 July 1995.

[64] P. A. Whigham. A schema theorem for context-free grammars. In *1995 IEEE Conference on Evolutionary Computation*, volume 1, pages 178–181, Perth, Australia, 29 November - 1 December 1995. IEEE Press.

[65] N. Wiener. *Cybernetics, or Control and Communication in the Animal and the Machine.* John Wiley, New York, 1948.
Wiener's classic book on cybernetics. Second edition with additions published in 1961.

[66] Elena Zannoni and Robert G. Reynolds. Learning to control the program evolution process with cultural algorithms. *Evolutionary Computation*, 5(2):181–211, summer 1997.

VITA

Emin Erkan Korkmaz was born in Merzifon on November 23, 1972. He received his B.S. degree in Computer Engineering from Bilkent University in July 1994. He received his M.S. degree from Middle East Technical University in September 1997. He worked as a teaching assistant in the Computer Engineering Department of the Hacettepe University between 1994 and 1996 and in the Computer Engineering Department of the Middle East Technical University between 1996 and 2002.