Program distribution estimation with grammar models

Program Distribution Estimation with Grammar Models

Yin Shan

B.Sci., M.Sci. (Wuhan Cehui Technical University, China)

A thesis submitted in fulfillment of the requirements
for the degree of
Doctor of Philosophy

University of New South Wales

Supervisors:

Robert I. McKay, University of New South Wales, Australia
Daryl L. Essam, University of New South Wales, Australia
Hussein A. Abbass, University of New South Wales, Australia

Examiners:

Alan Blair, University of New South Wales, Australia
Hitoshi Iba, University of Tokyo, Japan
Una-May O'Reilly, Massachusetts Institute of Technology, USA

Abstract

Program Distribution Estimation with Grammar Models

Yin Shan

This thesis studies grammar-based approaches in the application of Estimation of Distribution Algorithms (EDA) to the tree representation widely used in Genetic Programming (GP).

Although EDA is becoming one of the most active fields in Evolutionary computation (EC), the solution representation in most EDA is a Genetic Algorithms (GA) style linear representation. The more complex tree representations, resembling GP, have received only limited exploration. This is unfortunate, because tree representations provide a natural and expressive way of representing solutions for many problems. This thesis aims to help fill this gap, exploring grammar-based approaches to extending EDA to GP-style tree representations.

This thesis firstly provides a comprehensive survey of current research on EDA with emphasis on EDA with GP-style tree representation. The thesis attempts to clarify the relationship between EDA with conventional linear representations and those with a GP-style tree representation, and to reveal the unique difficulties which face this research.

Secondly, the thesis identifies desirable properties of probabilistic models for EDA with GP-style tree representation, and derives the PRODIGY framework as a consequence.

Thirdly, following the PRODIGY framework, three methods are proposed. The first method is Program Evolution with Explicit Learning (PEEL). Its incremental general-to-specific grammar learning method balances the effectiveness and efficiency of the grammar learning. The second method is Grammar Model-based Program Evolution (GMPE). GMPE realises the PRODIGY framework by introducing elegant inference methods from the formal grammar field. GMPE provides good performance on some problems, but also provides a means to better understand some aspects of conventional GP, especially the building block hypothesis. The third method is Swift GMPE (sGMPE), which is an extension of GMPE, aiming at reducing the computational cost.

Fourthly, a more accurate Minimum Message Length metric for grammar learning in PRODIGY is derived in this thesis. This metric leads to improved performance in the GMPE system, but may also be useful in grammar learning in general. It is also relevant to the learning of other probabilistic graphical models.

Extended Abstract

Program Distribution Estimation with Grammar Models

Yin Shan

This thesis studies grammar-based approaches in the application of Estimation of Distribution Algorithms (EDA) to the tree representation widely used in Genetic Programming (GP).

Evolutionary Computation (EC), motivated by evolution in the real world, has become one of the most widely used Machine Learning techniques, because of its effectiveness and versatility. Partially due to the disruptiveness of the genetic operators, there has been a growing interest in Estimation of Distribution Algorithms (EDA). The basic idea of EDA is to capture the interactions among genes, which represent the internal structure of problem solutions, and in this way to estimate the distribution of good solutions directly, rather than employing genetic operators. More specifically, in each generation, a model, typically a probabilistic model, is inductively learnt from good individuals (training examples), and is sampled to create new individuals of the next generation. A population is not usually maintained between generations, and genetic operators are omitted from EDAs.

Although EDA is becoming one of the most active fields in EC, the solution representation in most EDA is a Genetic Algorithms (GA) style linear representation (one dimensional array, known as the chromosome in GA literature). The more complex tree representations, resembling Genetic Programming (GP), have received only limited exploration, perhaps as a result of their intrinsic complexity. This is unfortunate, because tree representations provide a natural and expressive way of representing solutions for many problems. This is a significant gap in EDA and GP research, because EDA not only has the potential to improve GP performance, but also provides a new perspective for understanding GP. This thesis aims to help fill this gap, exploring grammar-based approaches to extending Estimation of Distribution Algorithms to GP-style tree representations.

Extending conventional EDA to tree representations is not trivial, because of the complexity of tree structures. Tree representations provide an alternative way of representing solutions, which is important to a large class of problems, but at the same time introduce important complications, including larger search spaces and more

complex interactions among tree nodes. This thesis proposes a framework, called Program Distribution Estimation with Grammar Models (PRODIGY), which significantly extends previous applications of EDA to GP-style tree representations. The core of this framework is a Stochastic Context-free Grammar model (SCFG). In this research, we confirm both theoretically and empirically that PRODIGY is able to learn a wide range of the dependencies among nodes in tree form individuals that are believed to be important in current GP literature. Two different approaches under the PRODIGY framework are proposed and implemented. The first one iteratively refines the grammar model (but requires additional constraints on the search), while the second one builds a full grammar model, including both its probability and structure components, directly from the training samples. These approaches performed well on the problems studied, exhibiting search characteristics complementary to those of GP. Specifically, the contributions of the thesis are as follows.

1. Providing a comprehensive survey of current research on EDA with emphasis on EDA with GP-style tree representation. We attempt to clarify the relationship between EDA with conventional linear representations and those with a GP-style tree representation, and to reveal the unique difficulties which face this research.

2. Identifying desirable properties of probabilistic models for EDA with GP-style tree representation, and deriving the PRODIGY framework as a consequence.

3. Proposing Program Evolution with Explicit Learning (PEEL) as one implementation of PRODIGY. PEEL's incremental general-to-specific grammar learning method balances the effectiveness and efficiency of the grammar learning.

4. Proposing Grammar Model-based Program Evolution (GMPE) as another implementation of PRODIGY. GMPE realises the PRODIGY framework by introducing elegant inference methods from the formal grammar field. It provides good performance on some problems, but also provides a means to better understand some aspects of conventional GP, especially the building block hypothesis.

5. Proposing Swift GMPE (sGMPE), which is an extension of GMPE, aiming at reducing the computational cost.

6. Deriving a more accurate Minimum Message Length metric for grammar learning in PRODIGY. This metric leads to improved performance in the GMPE system, but may also be useful in grammar learning in general. It is also relevant to the learning of other probabilistic graphical models.

Acknowledgement

My journey towards this Ph.D. was a long and arduous one, and would not have been possible without the support of so many wonderful people.

The people to whom I am most indebted are my parents, Zuo Fengxian and Shan Boquan, and my brother, Shan Di. I can never overstate their importance to me, for this achievement would never have been possible without their love and support.

I thank my current supervisors Dr. Robert I. McKay, Dr. Daryl Essam and Dr. Hussein Abbass for their patience, encouragement and guidance. I also would like to thank a number of people with whom I am fortunate enough to work, in particular David Paull, Rohan Baxter and Chris Lokan.

The road to my Ph.D. can be, to some extent, traced back to my stay in Hong Kong. I would like to thank my supervisor, Dr. Lin Hui, and my friends who made my stay in HK a most enjoyable experience: Zhao Yibin, He Jie, Freeman Wong, Lu Jia, Tan Qian and Han Haiyang.

I want to thank all of my fellow postgraduates for their assistance and their company, particularly Yang Ang, Jennifer Badham, Mehrdad Khodai-Joopari, Xie Shuiwei and Nguyen Xuan Hoai.

Arriving and staying overseas has always been a challenge to me. I thank my friends in Canberra who made my stay here more comfortable and pleasant. Above all I thank Anthony Mabanta, Cesar Gonzales, Terry Stocker and my two fantastic flatmates Do-Seong Byun and Wayne Bland. Special thanks go to Rodolfo Gomez-Balderas for his constant support and encouragement.

Finally, I would like to thank all the members of the school who provided such an amiable environment in which to work, especially Wen Ung, Pam Giannakakis, Alison McMaster, Tony Watson, Eri Uchida and Philip McGarva.

Certificate of Originality

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by colleagues, with whom I have worked at UNSW or elsewhere, during my candidature, is fully acknowledged.

I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Yin Shan

List of Publications

Journal Publications

1. Y. Shan, R. I. McKay, H. A Abbass, D. Essam. Program Distribution Estimation with Grammar Models. IEEE Transaction on Evolutionary Computation. (submitted)

2. Y. Shan, D. Paull, R. I. McKay. Machine Learning of Poorly Predictable Ecological Data. Ecological Modelling. Elsevier (accepted, subject to revision)

Refereed Conference Publications

1. Y Shan, R.I. McKay, D. Essam, J. Liu (2004, Dec). Modularity and position independence in EDA-GP. In Proceedings of The Second Asia-Pacific Workshop on Genetic Programming, Cairns, Australia.

2. Y. Shan, R. I. McKay, H. A Abbass, D. Essam (2004, Dec). Program Distribution Estimation with Grammar Models: A Preliminary Study. In Proceedings of The 8th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Cairns, Australia.

3. Y. Shan, R.I. McKay, R. Baxter, H. Abbass, D. Essam, H.X. Nguyen (2004, June). Grammar Model-based Program Evolution. In Proceedings of The Congress on Evolutionary Computation 2004, Portland, US. pp. 478–485.

4. Y. Shan, R. I. McKay, H. A. Abbass, D. Essam (2003, Dec). Program Evolution with Explicit Learning: a New Framework for Program Automatic Synthesis. In Proceedings of The Congress on Evolutionary Computation, Canberra, Australia, pp. 1639-1646. Also available as Technical Report CS04/03. School of Computer Science, Univ. College, Univ. of New South Wales. 2003

5. Y. Shan, H. A. Abbass, R.I. McKay, D. Essam (2002, Nov). AntTAG: a Further Study. In Proceedings of The Sixth Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems, Canberra, Australia.

6. N.X. Hoai, Y. Shan and R.I. McKay (2002, Nov). Is Ambiguity Useful or Problematic for Grammar Guided Genetic Programming? A Case Study. In Proceedings of 4th Asia-Pacific Conference on Simulated Evolution And Learning, Singapore.

7. Y. Shan, R.I. McKay and D. Paull (2002, Nov) Building Ecological Models Using Genetic Programming. In Proceedings of 4th Asia-Pacific Conference on Simulated Evolution And Learning, Singapore.

8. Y. Shan, R.I. McKay, C.J. Lokan and D.L. Essam (2002). Software Project Effort Estimation Using Genetic Programming. In Proceedings of International Conference on Communications Circuits and Systems, Chengdu, China, pp. 1108-1112, UESTC Press.

Edited Proceedings

1. S.-B. Cho, H. X. Nguyen, Y. Shan (2003, Dec). Proceedings of The First Asian-Pacific Workshop on Genetic Programming. Canberra, Australia. ISBN: 0975172409

Contents

List of Figures

List of Tables

List of Acronyms

EC	Evolutionary Computation
EDA	Estimation of Distribution Algorithms
EDA-GP	EDA with tree representation, resembling GP individual.
GA	Genetic Algorithms
GGGP	Grammar Guided Genetic Programming
GMPE	Grammar Model-based Program Evolution. One of the algorithms proposed in this thesis.
GP	Genetic Programming
IGC merge	Initial-Grammar-Consistent merge
MBS	Model-base Search
MDL	Minimum Description Length
MML	Minimum Message Length, also referring to minimum encoding inference.
NLP	Natural Language Processing
PEEL	Program Evolution with Explicit Learning. One of the algorithms proposed in this thesis.
PIPE	Probabilistic Incremental Program Evolution
PRODIGY	Program Distribution Estimation with Grammar Models
SCFG	Stochastic Context-free Grammar

Chapter 1

Introduction

This thesis studies grammar-based approaches in the application of Estimation of Distribution Algorithms to the tree representation widely used in Genetic Programming.

1.1 Background

Evolutionary Computation (EC) (Spears, Jong, Bäck, Fogel, and de Garis 1993), motivated by evolution in the real world, has become one of the most widely used Machine Learning techniques, because of its effectiveness and versatility. It maintains a population of solutions, that evolve according to rules of selection and genetic operators, such as recombination and mutation. Each individual in the population receives a measure of its fitness. Then, genetic operators perturb those individuals, thus providing general heuristics for exploration. EC algorithms often provide robust and powerful adaptive search mechanisms.

EC, generally speaking, is an inductive learning process. However, the knowledge learnt is cryptically encoded in the form of the population, which is under continuous evolution. Because of the highly dynamic nature of the evolving population, it is

extremely difficult to work out what the encoded knowledge is, let alone make use of it to improve the search performance. Furthermore the genetic operators, which are usually imposed with uniform distribution on the nodes of individuals, may cause severe disruption of partial solutions (also known as building blocks).

These issues in EC [1] led to the proposal of Estimation of Distribution Algorithms (EDA) (Larrañaga and Lozano 2001; Müehlenbein and Paaß 1996), also known as Probabilistic Model-building Genetic Algorithms (PMBGA) (Pelikan 2002) or Iterated Density Estimation Evolutionary Algorithms (IDEA) (Bosman and Thierens 1999). EDAs explicitly encode the knowledge accumulated in the course of the search in well-structured models, typically probabilistic models, and thus explicit exploitation of the knowledge becomes possible. More specifically, these models are inductively learnt from good individuals (training examples), and are sampled to create new individuals of the next generation. A population is not usually maintained between generations, and genetic operators are omitted from EDAs, either partially of completely. Instead, EDA is designed to capture the interactions among genes, which represent the internal structure of problem solutions, and in this way it estimates the distribution of good solutions directly, rather than employing genetic operators. The major differences between EDA methods lie in the model formalisms, depending on the type of knowledge which is intended to be represented.

There are several reasons for the increasing interest in EDA. The first reason is the theoretical attraction. The highly complex and dynamic consequences of genetic operators are extremely hard to understand and predict, hindering efforts to further improve the performance of EC systems. Replacing genetic operators and populations with a simple yet powerful model makes it simpler to understand system behaviour. In some simple cases, EDA is a quite accurate approximation of conventional EC (Harik, Lobo, and Goldberg 1999; Müehlenbein and Paaß 1996). Secondly, in terms of practical usefulness, superior performance of EDAs relative to conventional EC has been reported in a number of publications (Blanco, Inza,

[1] EC covers a wide range of techniques, but we will frame our discussion primarily in terms of Genetic Algorithms (Goldberg 1989) and Genetic Programming (Cramer 1985; Koza 1992).

and Larrañaga 2003; Bengoetxea, Larrañaga, Bloch, Perchant, and Boeres 2002; Sagarna and Lozano 2004; Paul and Iba 2004; Pelikan, Goldberg, Ocenasek, and Trebst 2003).

1.2 Theme of the Thesis

Although EDA is becoming one of the most active fields in EC, the solution representation in most EDA is a Genetic Algorithms (GA) style linear representation (one dimensional array, known as the chromosome in GA literature). The more complex tree representations, resembling Genetic Programming (GP) have received only limited exploration, perhaps as a result of their intrinsic complexity. This is unfortunate, because tree representations provide a natural and expressive way of representing solutions for many problems. This is a significant gap in EDA and GP research, because EDA not only has the potential to improve GP performance, but also provides a new perspective for understanding GP. This thesis aims to help fill this gap, exploring grammar-based approaches to extending Estimation of Distribution Algorithms to GP-style tree representations.

Extending conventional EDA to tree representations is non-trivial, because of the complexity of tree structures. Tree representations provide an alternative way of representing solutions, which is important to a large class of problems, but at the same time introduce important complications, including larger search spaces and more complex interactions among tree nodes. This thesis proposes a framework, called Program Distribution Estimation with Grammar Models (PRODIGY), which significantly extends previous applications of EDA to GP-style tree representations. The core of this framework is a Stochastic Context-free Grammar model (SCFG). In this research, we confirm both theoretically and empirically that PRODIGY is able to learn a wide range of the dependencies among nodes in tree form individuals, which are believed to be important in current GP literature. Two different approaches under the PRODIGY framework are proposed and implemented. One iteratively re-

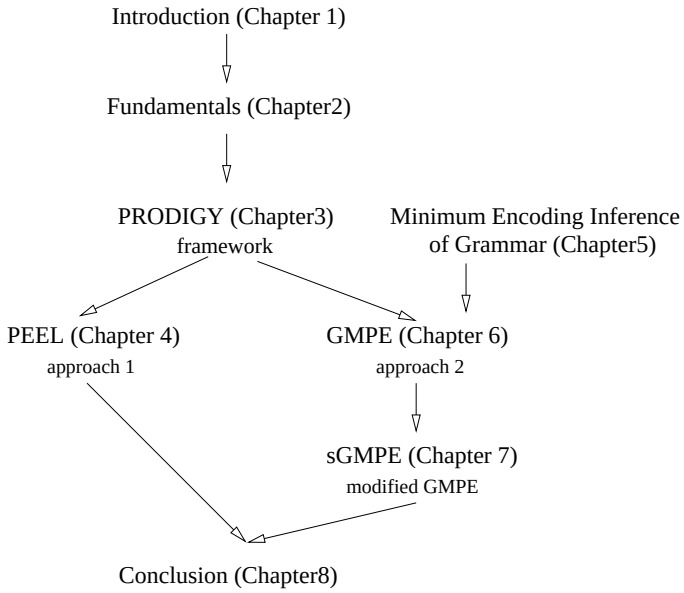


Figure 1.1: Structure of the thesis

fines the grammar model (but requires additional constraints on the search), while the other builds a full grammar model, including both its probability and structure components, directly from the training samples. On the problems studied, these approaches performed well, exhibiting search characteristics complementary to those of GP.

1.3 Thesis Outline

There are eight chapters in this thesis. The relationships among these chapters are illustrated in Figure 1.1. The first two chapters provide the basics of this thesis, including the themes of the thesis and the necessary fundamentals. Chapter 3 presents the PRODIGY framework for extending EDA to tree representations. Chapter 4 and Chapter 6 present two approaches (PEEL and GMPE respectively) under the PRODIGY framework. Chapter 5 and Chapter 7 are related to Chapter 6. Chapter 5 describes minimum encoding inference of grammars, which provide the basis for grammar learning in Chapter 6, while Chapter 7 presents a modified version of the

method proposed in Chapter 6, with improved computational efficiency. The final chapter gives the conclusions and discusses future research directions.

1.4 Major Contributions

The object of the thesis is the important and challenging effort to extend conventional EDA to GP-style tree representation. This research attempts to address some of the critical issues of this extension. Specifically, the major contributions of the thesis are as follows.

1. Providing a comprehensive survey of current research on EDA with emphasis on EDA with GP-style tree representation. We attempt to clarify the relationship between EDA with conventional fixed-complexity representations and those with a GP-style tree representation, and to reveal the unique difficulties which face this research.

2. Identifying the desirable properties of probabilistic models for EDA with GP-style tree representation, and deriving the PRODIGY framework as a consequence.

3. Proposing Program Evolution with Explicit Learning (PEEL) as one implementation of PRODIGY. PEEL's incremental general-to-specific grammar learning method balances the effectiveness and efficiency of the grammar learning.

4. Proposing Grammar Model-based Program Evolution (GMPE) as another implementation of PRODIGY. GMPE realises the PRODIGY framework by introducing elegant inference methods from the formal grammar field. It provides good performance on some problems, but also provides a means to better understand some aspects of conventional GP, especially the building block hypothesis.

5. Deriving a more accurate Minimum Message Length metric for grammar learning in PRODIGY. This metric leads to improved performance in the GMPE system, but may also be useful in grammar learning in general. It is also relevant to the learning of other probabilistic graphical models.

Chapter 2

Fundamentals

This chapter presents the fundamental knowledge upon which this thesis is based. This thesis aims at filling a gap between the Estimation of Distribution Algorithms (EDA) (Bosman and Thierens 1999; Larrañaga and Lozano 2001; Müehlenbein and Paaß 1996; Pelikan 2002) and Genetic Programming (GP) (Cramer 1985; Schmidhuber 1987; Koza 1992). One of the essential issues in EDA is choosing both an appropriate probabilistic model and also its corresponding learning method. The probabilistic model employed in this research is a grammar model. Hence, there are three bodies of knowledge involved in this research

- Stochastic Context-free Grammars (SCFG) and their inference methods,
- Genetic Programming (GP), in particular, Grammar Guided Genetic Programming (GGGP),
- EDA, especially EDA with tree representations.

The start of this chapter discusses grammars and their corresponding learning methods, then moves on to GP and GGGP (Wong and Leung 1995; Whigham 1995b; Gruau 1996). The following sections present the basics of EDA, with emphasis on EDA with GP-style tree representations. Related works using Ant Colony Opti-

misation (Bonabeau, Dorigo, and Theraulaz 1999) to synthesise programs are also briefly discussed.

2.1 Grammar and Grammar Inference

Stochastic Context-free Grammars (SCFG) (Manning and Schütze 1999) are discussed in this section. We are particularly interested in the inference of SCFGs (Stolcke 1994; Chen 1996), since model inference is one of the essential parts of EDA. In this research, the scoring metric is based on minimum encoding inference (Wallace and Boulton 1968; Wallace and Dowe 1999; Rissanen 1989). Grammars are also closely related to Grammar Guided Genetic Programming (GGGP).

2.1.1 Stochastic Context-free Grammar

The most common grammar formalism used in GGGP is the Context-free Grammar (CFG). This research mainly uses a Stochastic Context-free Grammar (SCFG), which can be viewed as a straightforward extension of the Context-free Grammar (CFG), i.e. a CFG with an extra probability component.

Formally, a Stochastic Context-free Grammar (SCFG) M consists of

- a set of nonterminal symbols N,
- a set of terminal symbols (or alphabet) Σ,
- a start nonterminal $S \in N$,
- a set of productions or rules R. The productions are of the form

  $$X \to \lambda$$

  where $X \in N$ and λ is in the union of all powers of the elements of $N \cup \Sigma$, i.e. $\lambda \in (N \cup \Sigma)^*$. X is called the left-hand side (LHS) of the production, while λ is the right-hand side (RHS),

- production probabilities $p(r)$ for all $r \in R$. For any given LHS symbol X, the sum of the probabilities of rules which have LHS X must be 1, i.e.

$$\sum_{r \ has \ LHS \ X} p(r) = 1.$$

The naming convention is as follows. The nonterminal symbols are distinguished by starting with a capital letter. Terminal symbols are defined by lower-case letters. A string that may be composed of both terminal and nonterminal symbols is represented by lower-case Greek letters.

Figure 2.1 is an example of a Stochastic Context-free Grammar (SCFG) for a symbolic regression problem. The symbol "|" represents disjunction which means the particular LHS may be rewritten with any of the RHSs connected by |. For example, LHS symbol *Exp* may be rewritten with either rule 1, 2 or 3. Similarly, *Op* may be written as $+, -, \times, /$ under rules, 4, 5, 6, 7, respectively. We assume that the probabilities follow a uniform distribution, and therefore the probabilities attached to the rules are omitted.

2.1.1.1 Derivation

For any strings γ and δ in $(N \cup \Sigma)^*$, if string $\gamma S \delta$ can be transformed to string $\gamma\alpha\delta$ by rule $S \rightarrow \alpha$, we say that $\gamma S \delta$ *directly derives* $\gamma\alpha\delta$ in grammar M, or $\gamma\alpha\delta$ is directly derived from $\gamma S \delta$. This is denoted as follows.

$$\gamma S \delta \overset{S \rightarrow \alpha}{\Rightarrow} \gamma\alpha\delta$$

If there exists a sequence of direct derivations $\alpha_0 \Rightarrow \alpha_1, \alpha_1 \Rightarrow \alpha_2, \ldots, \alpha_{n-1} \Rightarrow \alpha_n$ where $\alpha_0 = \alpha,\ \alpha_n = \beta,\ \alpha_i \in (N \cup \Sigma)^*$, and $n \geq 0$, which transforms string α to string β, we say α *derives* β, or β is derived from α, denoted as follows.

$$\alpha \overset{*}{\Rightarrow} \beta$$

Such a sequence is called a derivation. Thus a derivation corresponds to a sequence of applying productions to generate a string. A derivation can be represented in a

S	→	Exp				(0)
Exp	→	Exp	Op	Exp	|	(1)
	→	Pre	Exp	|		(2)
	→	x				(3)
Op	→	+	|			(4)
	→	−	|		(5)
	→	×	|		(6)
	→	/				(7)
Pre	→	sin	|			(8)
	→	cos	|			(9)
	→	e^	|			(10)
	→	ln				(11)

Figure 2.1: A commonly used Stochastic Context-free grammar for symbolic regression problems.

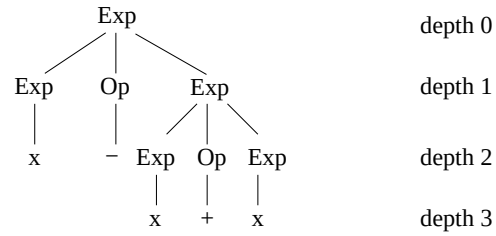


Figure 2.2: Derivation tree of string x-(x+x)

parse tree format, called a *derivation tree* or *parse tree*. For example, Figure 2.2 is a derivation tree of string $x - (x + x)$. Unfortunately CFG theory and GP theory use the term "parse tree" inconsistently with each other; to avoid confusion, this thesis will use the term "derivation tree" (which is not otherwise used in GP) rather than "parse tree".

2.1.1.2 Probability of Derivation

The probability attached to each production rule indicates the probability of being chosen. If the probability component in the above definition is removed (i.e. a uniform distribution is imposed on the productions), an SCFG becomes a Context-free Grammar (CFG). CFGs are commonly used in GGGP to impose grammar constraints. A simple example of a CFG can be found in Figure 2.1 (it is in fact an SCFG, but it is equivalent to a CFG because a uniform distribution is assumed.) As can be seen, it is possible that one nonterminal can be rewritten in different ways. For example, nonterminal "Exp" can be rewritten using either rule 1, 2 or 3. In a CFG, all of these rules have equal probabilities to be chosen and therefore there is no bias. In an SCFG, the added probability component makes it possible to bias toward some rules rather than others.

In SCFGs, the probability of a derivation (or a derivation tree) is the product of the probabilities of all the production rules involved. Formally, the probability of a derivation $\alpha_0 \Rightarrow \alpha_1, \alpha_1 \Rightarrow \alpha_2, \ldots, \alpha_{n-1} \Rightarrow \alpha_n$, where each step of derivation is a direct derivation, is

$$p(\alpha_0 \overset{*}{\Rightarrow} \alpha_n) = \prod_{i=0}^{i=n-1} p(X_i = \lambda_i) \tag{2.1}$$

where production rule $X_i = \lambda_i$ is used to derive α_{i+1} from α_i and $p(X_i = \lambda_i)$ is the probability of rule $X_i = \lambda_i$.

Note: although a derivation corresponds to only one string, a string without bracketing to represent its internal structure, may (in general) be derived in a number of different ways from a given grammar. In Natural Language Processing (NLP), this phenomenon, known as ambiguity, causes severe difficulty when inferring grammars from sentence examples. However in Grammar Guided Genetic Programming (GGGP), and in the research presented in this thesis, this is not an issue, because in both fields of research, the derivations of the individuals/examples from the given grammar are preserved, and thus correct, unique derivations are known.

In the thesis, sampling an SCFG grammar means deriving a set of strings from the

given SCFG grammar. When deriving a string, a LHS may have more than one RHS. If we come to this kind of LHS, we need to choose one of its RHSs. This chapter has previously defined that an SCFG has a probability component attached to each rule (more accurately to each RHS), the RHS is chosen based on this probability.

For example, a common SCFG grammar for the symbolic regression problem is illustrated in Figure 2.1. An individual, which may be derived or sampled from this grammar, is illustrated in Figure 2.2. The individual in Figure 2.2 is derived by applying the following sequence of rules. Starting from the starting symbol S, rule 0 is chosen to rewrite S and obtain

Exp Op Exp.

After probabilistically choosing rules 3, 5 and 1 to respectively rewrite these three symbols, the following string is obtained:

x - (Exp Op Exp).

Symbols “x” and “-” are terminals because there is no rule which can rewrite them. Therefore, only the last three symbols are written with rules 3, 4 and 3 respectively. The eventual result is:

x - (x + x).

As defined in Equation 2.1, the probability of the above derivation can be calculated as the product of the probabilities of all the rules involved. The rules involved in this derivation were used in the following order

0 3 5 1 3 4 3.

The probability of rule 0 is 1 because there is no other alternative. Given the assumption of uniform distribution, because “Exp” can be written with rule 1, 2 or

3, the probability of rule 3 is 1/3. Similarly we may work out the probability of other rules. Thus, the probability of this derivation is

$$1 \cdot \frac{1}{3} \cdot \frac{1}{4} \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{4} \cdot \frac{1}{3} \approx 0.00077.$$

As mentioned before that some symbols can be rewritten with more than one rule. For example, the symbol "Exp" can be rewritten using either rule 1, 2 or 3. In conventional GGGP, uniform distribution of the probabilities of rules is assumed. Therefore, the choice of which rule to apply is made uniformly randomly. However, it does not have to be so. In some work, such as (Whigham 1995b), weights, which roughly correspond to the probabilities attached to rules, are assigned to rules, so that this choice can be made to stochastically favour some rules over others.

2.1.2 Learning Grammar with Minimum Encoding Inference

Since grammar is originated from the field of Natural Language Processing (NLP) and it is an important means to model language, its inference (learning) has been extensively studied in NLP (Sakakibara 1997).

In this thesis, we are particularly interested in the inference of SCFGs. An SCFG can be understood as a CFG with probabilities attached. Therefore, it has two parts: the structure (the CFG part) and the probability part. To learn an SCFG model, we have to learn these two parts. The inference of the structure (CFG) (Angluin 1990; Sakakibara 1990; Sakakibara 1992) and of the probabilities given CFG (Lari and Young 1990; Lari and Young 1991; Pereira and Schabes 1992) were initially studied separately. However, recent works address the inference of the full SCFG simultaneously, including both the structure and probabilities of SCFG (Stolcke 1994; Chen 1995; Keller and Lutz 1997; Grunwald 1994; Osborne 1999). The methods inferring the full SCFG are usually based on the Bayesian framework.

More specifically, given data set D, the preferred model G is a maximum a posteriori

(MAP) model which maximises the posterior probability:

$$\begin{aligned} G_{MAP} &= \text{argmax } p(G|D) \\ &= \text{argmax } \frac{p(D|G)p(G)}{p(D)} \\ &= \text{argmax } p(D|G)p(G) \end{aligned} \tag{2.2}$$

Equation 2.2 can also be interpreted in the light of basic concepts from information theory. From this perspective, it can be rewritten as:

$$\begin{aligned} G_{MAP} &= \text{argmax } p(D|G)p(G) \\ &= \text{argmax } \log p(D|G) + \log p(G) \\ &= \text{argmin } -\log p(D|G) - \log p(G) \end{aligned} \tag{2.3}$$

As the negative logarithm of probability is the optimal description length, Equation 2.3 can be expressed in the form of description length:

$$\begin{aligned} G_{MAP} &= \text{argmin } -\log p(D|G) - \log p(G) \\ &= L(D|G) + L(G) \end{aligned} \tag{2.4}$$

where $L(\cdot)$ is the description length.

Equation 2.4 indicates that the preferred model is the one which minimises the sum of description length of the model and the description length of accuracy of the model given data. This is the basic idea of minimum encoding inference, more commonly referred to as the Minimum Message Length (MML) principle (Wallace and Boulton 1968; Wallace and Freeman 1987; Wallace and Dowe 1999) or Minimum Description Length (MDL) principle (Rissanen 1989). In this thesis, the term MML is employed to refer to both.

Inferring an SCFG is a difficult problem. Given the MML principle, most of the current methods in SCFG inference are based on greedy search. In greedy search, the SCFG is altered using some variation operators. If after a given number of variations, a better SCFG is found, it is accepted as the basis for the next iteration. Otherwise, the search usually stops. During the search, the MML principle is used to compare competing models.

1. Generate a population $\mathcal{P}$ randomly.
2. Select a set of fitter individuals $\mathcal{G}$ from population $\mathcal{P}$.
3 Apply genetic operators on the set of selected individuals $\mathcal{G}$ to obtain a set of children $\mathcal{G}'$.
4. Incorporate the children $\mathcal{G}'$ into population $\mathcal{P}$.
5. If the termination condition is not satisfied, go to 2.

Figure 2.3: Basic Algorithm of GP

More detailed discussion of grammar inference with minimum encoding inference will be presented in Chapter 5.

2.2 Genetic Programming and Grammar Guided Genetic Programming

This section covers GP and one of its variants, Grammar Guided Genetic Programming (GGGP) which is particularly related to this thesis. Some GP issues, such as semantics, building blocks and introns, are also discussed. There are several terminologies used inconsistently in the field of GP, especially in GGGP. To eliminate confusion caused by the inconsistent usage, these important concepts are clarified at the end of this section.

2.2.1 Genetic Programming

Genetic Programming (GP) (Cramer 1985; Schmidhuber 1987; Koza 1992) is an Evolutionary Computation approach (EC). GP can be most readily understood by comparison with Genetic Algorithms (GA) (Holland 1975; Goldberg 1989). The basic algorithm of GP can be found in Figure 2.3. This algorithm is very similar to that of GA. Essentially, an initial population is randomly generated. Then to each population in turn, genetic operators are imposed on stochastically selected

individuals from one population to obtain its subsequent population.

Rather than evolving a linear string as GA does, Genetic Programming evolves computer programs, which are usually tree structures. The fitness of each individual is a function of the effectiveness of the program. The genetic operators, such as crossover, are refined to act upon sub-trees, so as to ensure that the child is syntactically correct.

Given the basic algorithm in Figure 2.3, some important details of this algorithm will be highlighted in the following subsections. In step 2, several selection methods may be applied. They are discussed in Section 2.2.1.1. There are usually three genetic operators involved in step 3, which are presented in Section 2.2.1.2. In step 4, two possible approaches can be used, generational approach and steady-state approach, which are presented in Section 2.2.1.3.

2.2.1.1 Selection Mechanism

A number of selection methods have been proposed in EC. Most of them are applicable to both GA and GP. One of the important reasons for this variety of selection mechanism is that different selection mechanisms are needed to impose an appropriate level of selection pressure for the problem at hand.

Truncation Selection This selection method has very strong selection pressure. In every generation, only the top $t\%$ of individuals are deterministically selected for breeding. Thus, the individuals with poorer fitnesses rapidly disappear. Hence, truncation selection may lead to too-rapid convergence.

Fitness Proportionate Selection This is one of the most common selection methods in GA, also known as "roulette wheel selection". For this method, the individuals are selected probabilistically, where the probability of selecting individual

I_j is given by

$$p(I_j) = \frac{fitness(I_j)}{\sum_k^{pop} fitness(I_k)} \tag{2.5}$$

where $fitness(I_k)$ is the fitness of individual I_k and the sum is over the entire population, which has size *pop*. The fitness proportionate selection has less selection pressure than truncation selection.

Tournament Selection Tournament selection is not based on population wide competition, but is limited to a subset of the population. Given a tournament size t, t individuals are randomly chosen (regardless of their fitness). The best individual in this subset is then selected. Tournament selection has adjustable selection pressure, controlled by tournament size t, where a bigger tournament size has stronger selection pressure.

2.2.1.2 Genetic Operators

There are three basic genetic operators in GP and GA: reproduction, crossover and mutation.

Reproduction Reproduction is straightforward. It simply copies the individual and places it into the new population.

Crossover Crossover combines the genetic material of two parents by swapping certain parts from both parents. Given two parents which are obtained by some selection method, the crossover performs two steps:

1. Select randomly a sub-tree in each parent.
2. Swap the selected sub-trees between parents.

This is illustrated in Figure 2.4. The sub-trees selected in parents are highlighted in the rectangle box on the left hand side. On the right hand side, two sub-trees have

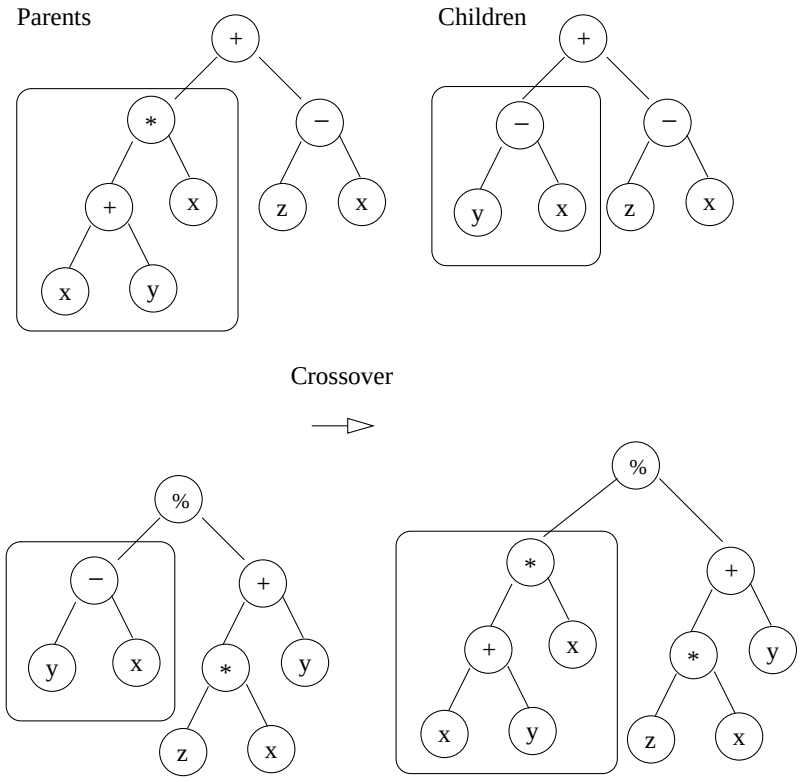


Figure 2.4: Crossover in Genetic Programming

been swapped to generate children.

Mutation Mutation acts on only one individual. It introduces a certain amount of randomness, to encourage exploration. Given one parent obtained by some selection method, the mutation performs three steps:

1. Select randomly a sub-tree in the parent.
2. Remove the selected sub-tree.
3. Generate randomly a new sub-tree to replace the removed sub-tree.

This is illustrated in Figure 2.5. To obtain the child, the selected sub-tree highlighted in the rectangular box on the left hand side is replaced by the newly generated sub-tree on the right hand side.

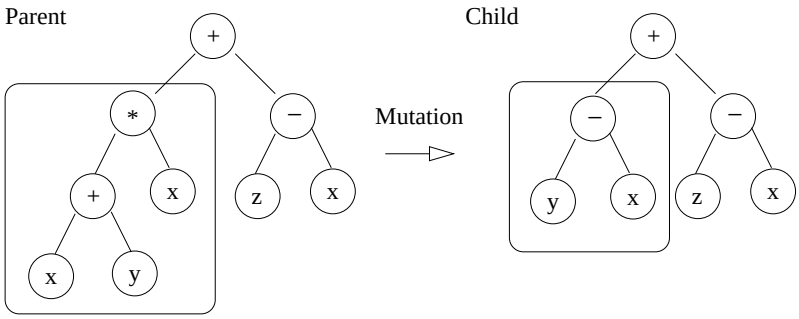


Figure 2.5: Mutation in Genetic Programming

2.2.1.3 Incorporation of New Individuals

There are two alternatives to implement step 5 of the algorithm in Figure 2.3, the generational approach and the steady-state approach. In each iteration, the generational approach discards the entire old population $\mathcal{P}$ and replaces it with a newly created set of individuals $\mathcal{G}'$. Each iteration is a generation. In contrast, for the steady-state approach, once the individuals $\mathcal{G}'$ are created, they are incorporated back into the population $\mathcal{P}$ directly, i.e. the old population $\mathcal{P}$ is maintained and some of its individuals are replaced by the new individuals according to some rules. Therefore, there are no clearly identifiable generations.

2.2.2 Some Issues in GP

Although the basic ideas of GA and GP are similar, they have unique characteristics because of the representation differences. This section discusses some of the characteristics of GP that are relevant to this thesis. In order to highlight these characteristics, we need to sometimes make a comparison with classic GA on which the conventional EDA is base. Classic GA in this discussion is roughly the GA system with fixed length, semantics attached to the locus and no genotype to phenotype mapping.

2.2.2.1 Semantics

When evolving a linear structure, as classic GA does, it is usually assumed that the semantics are attached to the locus (each position of GA linear string is called locus). For example, when using classic GA to solve a travelling salesman problem, one common encoding method has each locus representing one step, in which one specific city is visited. In other words, the first locus is interpreted as a city ID that is to be visited in the first step, the second locus is the second city to be visited, etc.

However, it is very hard, if not impossible, to find such an analogy in GP. GP tries to assemble a set of symbols which have meaning *on their own*. Thus, the meaning does not change, no matter where the symbol is. Consequently, the effect of the node has to be understood in its surrounding context, not by the absolute position of the symbol. For example, the Artificial Ant Problem (Koza 1992) is one of the standard GP benchmark problems, requiring GP to find a sequence of instructions (GP symbols) manipulating the artificial ant to find the food in the given grid. Each symbol has its meaning. In the Artificial Ant Problem, symbol *move* means move forward one step. Hence, the meaning does not need to be interpreted according to its position. No matter whether *move* sits on either the root node or a node at some other depth level, *move* has the same meaning. However, the effect of *move* needs to be understood by considering its surrounding context.

2.2.2.2 Building Blocks

The Schema Theorem and its related Building Block Hypothesis (Holland 1975; Goldberg 1989) provide a mathematical perspective to analyse GA and interpret its behaviour. These ideas have also been extended to GP research.

In GA, the schema is defined as any string composed of 0's, 1's and *'s. Each schema represents the set of bit strings containing the indicated 0's and 1's, with each "*" interpreted as a "don't care". For example schema 0*1 represents the set of bit strings composed of 001 and 011 exclusively. As we can see, in this definition,

a schema implicitly takes position as a reference. This is related to the semantics discussed above. In theoretical GP research, such as (Koza 1992; O'Reilly and Oppacher 1995; Whigham 1995c), due to fact that semantics are attached to the symbols, the schemas are non-rooted and thus the position reference is removed. In these studies, a schema is defined as a sub-tree of some kind and the absolute position of the sub-tree is not considered. The quality of a non-rooted schema is hence determined by its own structure, not just by where it is. There are some recent studies which introduce rooted schema. However, we believe it is more likely due to the mathematical tractability of schema modelling. This issue will be revisited in Section 3.2.3)

This subtle change in the definition of GP schema makes GP schema research more relevant than just taking the original GA definition into GP. For example, in the Artificial Ant Problem, the effect of the symbol *move* would depend on where the ant has been positioned by the sequence of symbols before this *move*, i.e. the context surrounding *move*. Taking another example from symbolic regression problems, the symbol '+' returns completely different values depending on its surrounding context – two operands in this case.

2.2.2.3 Bloat and Introns

In GP, the individuals tend to bloat, i.e. to grow rapidly in size over the course of a run. This has been observed many times, and is well recognised in GP research (Angeline 1994; Angeline 1994; Banzhaf, Nordin, Keller, and Francone 1998). Bloat often involves introns. When evolving solutions in GP, it is common that some parts of an individual do not contribute to its fitness, and thus can be removed when evaluating fitness. These parts of the individual are called introns. More rigorously, introns in Genetic Programming are areas of code that are unnecessary since they can be removed from the program without altering the solution the program represents. There is no consensus on the cause of introns or their impact on GP performance, but their existence is amply confirmed. In contrast, this phenomenon does not exist

in classic GA, which usually employs a fixed length linear representation.

2.2.3 Grammar Guided Genetic Programming

Grammar Guided Genetic Programming (GGGP) (Wong and Leung 1995; Whigham 1995b; Gruau 1996) is a Genetic Programming approach relying on a grammar constraint. All kinds of grammars can be used to describe the constraint but this thesis will mainly focus on Context-free Grammars (CFG). Similar to Strongly-typed GP (Montana 1993), grammars in GGGP provide a systematic way to handle typing.

More importantly, GGGP can constrain the search space so that only grammatically correct individuals can be generated. Thus it can be used to reduce the search space or to introduce some background knowledge. Use of a grammar requires that the genetic operators, such as mutation and crossover, must respect the grammar constraint, i.e. after imposing a genetic operation, the child must still be grammatically correct. In other words, the child must still be consistent, after genetic operations, with the given grammar.

2.2.4 Individuals of GP and GGGP

In the early days of GP, the connections between GP representation and CFGs was recognised, and some terminology from natural language processing was imported into GP. Unfortunately, the interconnection was not carefully explored, and as a consequence, when grammars were fully adopted in GGGP, some inconsistencies in terminology appeared. These concepts and terminologies are closely related to this thesis, and thus deserve this dedicated section. This confusion is largely caused by the different forms of individuals in GP and GGGP.

The GP individual is the tree structure proposed in (Koza 1992), which has become the standard concept in the GP literature. In the GP tree, each node is a GP

terminal or nonterminal, and therefore can be evaluated. To remove confusion, we refer to this structure as an *expression tree*.

The GGGP individual has a very different form. A typical individual is represented in GGGP in Figure 2.2. As can be seen, the nonterminals, such as "Exp", in this GGGP individual, are not conventional GP nonterminals. They are grammar nonterminals; they are used for indicating the structure and may not be able to be evaluated. This difference has been discussed in (Whigham 1995b). To distinguish these two concepts, we use the terms GP terminal/nonterminal and grammar terminal/nonterminal, respectively. On most occasions in this thesis, if no confusion is likely to occur, we will use terminal/nonterminal to refer to the latter.

The expression tree of GP is not irrelevant to GGGP. The individual in GGGP is a derivation tree not an expression tree. A derivation tree records the structure of the tree and therefore can be directly used for evaluation. Although a derivation tree can be uniquely mapped to an expression tree with parentheses (which indicate the structure), it seems there is not any need to do so. In GGGP, as well as most of the research reported in this thesis, since we generally have the derivation tree as an individual, there is no need to parse an individual. Note that this is an important difference from the field of Natural Language Processing (NLP), where grammar theory originated. In NLP, most of the time we do not have the derivation tree and thus we have to parse the sentence to obtain its hierarchical structure (derivation tree), so that the sentence may be understood. However, in GGGP, each individual is a derivation tree and it is not usual to remove its hierarchical structure, convert it into a linear string and then re-parse it. In NLP, it is possible that a sentence can be parsed in many different ways for a given grammar. This phenomenon is known as ambiguity, and often causes severe difficulty in NLP. However, because of the elimination of parsing, this ambiguity problem in general does not arise in either GGGP or most of the research reported in this thesis.

In the remainder of this thesis, we will usually refer to the GGGP individual (derivation tree) as a tree, without further detail; this is unlikely to cause confusion, because

the expression tree of GP is largely irrelevant to this research.

2.3 Estimation of Distribution Algorithms

One common theory to interpret GA, as well as GP, is that through the genetic operations - especially crossover - building blocks, which are high quality partial solutions, are discovered and combined to form better solutions (Holland 1975; Goldberg 1989).

Therefore, if we can learn these building blocks directly, instead of applying the semi-blind genetic operators (which may destroy building blocks) we hope to significantly improve the performance of GA and GP.

A more formal perspective is as an estimation of the distribution of good solutions, known as Estimation of Distribution Algorithms (EDA) (Larrañaga and Lozano 2001; Müehlenbein and Paaß 1996). EDA is also known as Probabilistic Model-building Genetic Algorithms (PMBGA) (Pelikan 2002), or Iterated Density Estimation Evolutionary Algorithms (IDEA) (Bosman and Thierens 1999). This thesis will use the term EDA. EDA uses a probabilistic model to estimate the distribution of promising solutions and to guide further exploration of the search space. By iteratively building and sampling the probabilistic model, the distribution of good solutions is (hopefully) approximated.

Due to the lack of knowledge of the true distribution, a probabilistic model to approximate the distribution has to be introduced. The particular form of introduced probabilistic model is strongly related to specific assumptions about building blocks.

The formal description of EDA can be found in (Bosman and Thierens 1999). Briefly, assume $\boldsymbol{Z}$ is the vector of variables we are interested in. $D^H(\boldsymbol{Z})$ is the probability distribution of individuals whose fitnesses are greater than some threshold H (without loss of generality, we assume we are dealing with a maximisation problem). Now if we know $D^{H_{opt}}(\boldsymbol{Z})$ for the optimal fitness H_{opt}, we can find a solution by simply

drawing a sample from this distribution. However, usually we do not know this distribution.

Hence, because we have no prior information on this distribution, we start from a uniform distribution. In the commonest form of the algorithm, we generate a population $\mathcal{P}$ with n individuals and then select a set of good individuals $\mathcal{G}$ from $\mathcal{P}$. Since $\mathcal{G}$ contains only selected individuals, it represents the search space that is worth further investigation. We now estimate a probabilistic model $\mathcal{M}(\zeta, \theta)$ from $\mathcal{G}$. ζ is the structure of the probabilistic model $\mathcal{M}$ while θ is the associated parameter vector. With this model $\mathcal{M}$, we can obtain an approximation $\bar{D}^H_{\mathcal{M}}(\boldsymbol{Z})$ of the true distribution $D^H(\boldsymbol{Z})$. To further explore the search space, we sample distribution $\bar{D}^H_{\mathcal{M}}(\boldsymbol{Z})$, and the new samples are then re-integrated into population $\mathcal{P}$ by some replacement mechanism. This starts a new iteration.

Two representations of encoding solutions may be used in EDA, i.e. GA style linear string and GP-style tree representation. This section presents the general theory of EDA while the following two sections will respectively discuss EDA based on GA-style linear strings, and EDA based on GP-style tree representation. For simplicity, the former will be referred to as conventional EDA and the later as EDA-GP. This term EDA-GP reflects the connection between EDA and GP, but is not intended to imply that EDA-GP employs genetic search.

2.3.1 Algorithm

All the EDA methods share a similar high level algorithm. This high level algorithm of EDA can be found in Figure 2.6. EDA starts with a randomly generated population $\mathcal{P}$. A probabilistic model $\mathcal{M}$ is learnt from a set of individuals $\mathcal{G}$ selected from this population. A new set of individuals $\mathcal{G}'$ is generated by sampling the learnt model $\mathcal{M}$. The new population is formed by incorporating $\mathcal{G}'$ into the original population $\mathcal{P}$. The next iteration starts again from this new population.

Given this high level algorithm, there are many variants. For example, each iteration

1. Generate a population $\mathcal{P}$ randomly.
2. Select a set of fitter individuals $\mathcal{G}$ from $\mathcal{P}$.
3. Estimate a probabilistic model $\mathcal{M}$ over $\mathcal{G}$.
4. Sample the probabilistic model $\mathcal{M}$ to obtain a set of new individuals $\mathcal{G}'$.
5. Incorporate $\mathcal{G}'$ into population $\mathcal{P}$.
6. If the termination condition is not satisfied, go to 2.

Figure 2.6: High level algorithm of EDA

may create a new population, or may simply replace part of the old population with newly generated individuals. The system may learn a model from scratch or simply update the previous model.

In the field of EDA, due to the lack of prior knowledge of the true distribution, it is assumed that the distribution follows some well studied model. Therefore, a vital part in the EDA algorithm is the accuracy of the model chosen with respect to the true model. This is the reason that current EDA research is largely devoted to finding appropriate models and is hence one of the natural ways to differentiate EDA methods, i.e. with respect to their probabilistic models.

2.3.2 Example

The following is an extremely simplified example of EDA, similar to the example given in (Pelikan, Goldberg, and Cantú-Paz 1999; Larrañaga and Lozano 2001).

The problem we want to solve is to maximise the *OneMax* function. This problem was originally defined in the context of GA. Given a binary string encoding of a solution, corresponding to GA individuals, the OneMax function is defined as the sum of the "1" bits in the string, i.e.

$$f(X) = \sum_{i=0}^{n-1} X_i \tag{2.6}$$

where $X = (X_0, \ldots, X_{n-1})$ is a possible solution (binary string) of the OneMax

problem and n is the length of the solution. In this example, $n = 5$, i.e. $X = (X_0, X_1, X_2, X_3, X_4)$, the population size is 6 and truncation selection is used.

The first two generations of a simple EDA run are illustrated in Figure 2.7. In the 0-th generation, six individuals are randomly generated. With truncation selection, the first half of the population is retained, i.e. the top 3 individuals.

The probabilistic model used in this example is a simple probability vector with dimension n. In this case, $n = 5$, and the probabilistic vector is $P = (p_0, \ldots, p_4)$. Each element of this vector is the probability of 1 occurring in the corresponding location in the individual, $p_i = p(X_i = 1)$. This probability is learnt by counting the frequency of 1s in each position. For example, the left most element in probability vector $p_0 = \frac{3}{3} = 1.00$. This is obtained by counting the frequency of $X_0 = 1$ in the selected individuals, e.g. $X_0 = 1$ in three out of three selected individuals.

Once the probabilistic model is constructed, the next population is obtained by sampling this probability model, i.e. the value of each locus is decided stochastically based on its probability. Not surprisingly, the fitnesses of this population are generally better than the previous one. This cycle is then repeated until some termination conditions are satisfied.

In this example, a probability vector is used as a probabilistic model to estimate the distribution of the selected individuals. However, in most cases, this simple model is not adequate to represent the complex interactions among genes. Therefore, methods with more complex probabilistic models are proposed.

2.4 Estimation of Distribution Algorithms with Linear Representation

The two possible encodings for EDA solutions are GA style linear strings and GP-style tree representations. Therefore, there are two identifiable streams of EDA

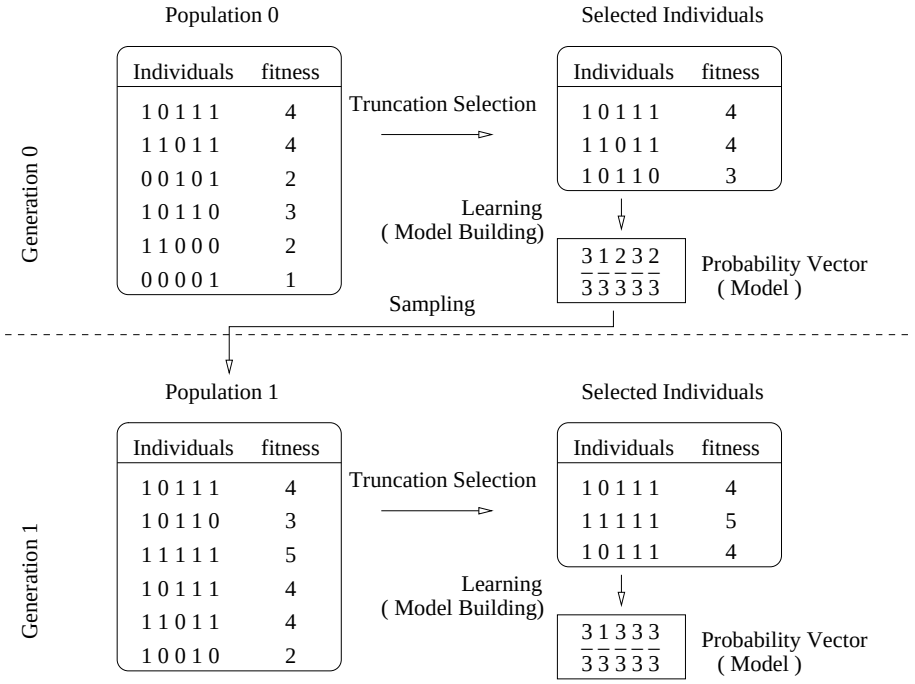


Figure 2.7: A simplified example of EDA.

works, i.e. EDA with GA style linear string representation, referred to as conventional EDA, and EDA with GP-style tree representation, referred to as EDA-GP. In this section, we discuss conventional EDA studies and leave EDA-GP to the following section.

If each locus in the GA chromosome is a random variable, the entire GA chromosome is a vector of random variables. Therefore, EDA attempts to estimate the probability distribution of this vector of random variables. One of the desirable ways to estimate this probability distribution, is to consider the joint probability distribution of all the random variables in this vector.

Assume $X = \{X_i\}$ is the vector of random variables, i.e. each of its X_i elements is a random variable. A sample of X corresponds to a chromosome of GA. The joint probability distribution over X is:

$$p(X) = p(X_1|X_2 \dots X_n)p(X_2|X_3 \dots X_n) \dots p(X_{n-1}|X_n)p(X_n) \tag{2.7}$$

Obviously, due to the combinatorial nature of this problem, it is not practical to

directly compute this joint probability distribution. Therefore, the main problem in EDA is how to estimate this probability. This has led to three types of approximation where the specific probability distribution is assumed to factorise the joint probability. These three types of methods are:

1. assuming the genes in the chromosome are independent (Baluja 1994; Harik, Lobo, and Goldberg 1999; Müehlenbein and Paaß 1996),
2. taking into account pairwise interactions (de Bonet, Isbell, and Viola 1997; Baluja and Davies 1997; Pelikan and Mühlenbein 1999),
3. accurately modelling very complex problem structures with highly overlapping multivariate building blocks (Mühlenbein and Mahnig 1999; Harik 1999; Pelikan, Goldberg, and Cantú-Paz 1999; Etxeberria and Larrañaga 1999).

Since there are comprehensive surveys (Pelikan, Goldberg, and Lobo 1999; Larrañaga and Lozano 2001) in the field of conventional EDA, this section just describes typical methods of these three categories in this section. It is organised as follows. First, some related works prior to conventional EDA are briefly mentioned in Section 2.4.1. Then, conventional EDA approaches which assume gene independence are presented in Section 2.4.2. Finally, conventional EDA approaches assuming pairwise and multivariate dependence are presented in Section 2.4.3 and Section 2.4.4, respectively.

2.4.1 Learning in Genetic Algorithms

Before the EDA framework was proposed, there had been some studies in GA which explicitly identified and facilitated the interactions among genes, known as linkage, such as Messy GA (Goldberg, Korb, and Deb 1989) and Linkage Learning GA (LLGA) (Harik and Goldberg 1997).

In both of these two studies, the chromosome is of variable length. It consists of movable genes, encoded as *(gene number, allele)* pairs. This coding makes it possible

to tighten and preserve the linkages among closely related genes, which might be far apart in conventional GA encoding. Modified genetic operators and new mechanisms for interpreting chromosomes were proposed in these methods, to suit the variable length chromosome and also to encourage the discovery of building blocks.

2.4.2 No Dependence Model

One very simplified way to approximate the joint probability in Equation 2.7 is to assume that all of the random variables in vector X are independent from each other, i.e. there is no dependence.

Population-Based Incremental Learning (PBIL) (Baluja 1994) is a typical work which assumes no dependence. In PBIL, as in other works which make the same assumption, the n-dimensional joint distribution factorises as a product of n univariate, independent probability distributions. Under this independence assumption, $p(X)$ can be approximated as follows.

$$\bar{p}(X) = \prod_{i=1}^{n} p(X_i) \tag{2.8}$$

In the basic PBIL algorithm, each random variable p_i on probability vector P is the probability of generating a 1 at the corresponding position in the binary chromosome, $p_i = p(X_i = 1)$. The probability of generating a 0 is $1 - p_i$. Initially, because we have no prior knowledge, vector P is initialised with uniform distribution, i.e. $p_i = 0.5$. P is updated every iteration with the best individual. If the value at position i of chromosome is 1, the probability p_i is shifted a certain amount toward 1.0. Otherwise toward 0.0. Formally,

$$p_i \leftarrow p_i \cdot (1.0 - lr) + X_i^{max} \cdot lr \tag{2.9}$$

where $lr < 1$ is a predefined learning rate, X_i^{max} is the value at position i of the chromosome of the best individual max. Since it is a binary chromosome, $X_i^{max} \in \{0, 1\}$.

As can be seen, the assumption in PBIL is that each random variable X_i is independent, and thus each of them is treated separately. However, it is obvious that this assumption will not hold in general. As a result, more complex models have been proposed.

2.4.3 Pairwise Dependence Model

One of the typical works which considers pairwise dependence is Information-Maximising Input Clustering (MIMIC) (de Bonet, Isbell, and Viola 1997).

In MIMIC, only pairwise conditional probabilities, $p(X_i|X_j)$ are considered, and thus X_i is conditionally independent of any other X_k, where $k \neq j$. Therefore, $p(X)$ in Equation 2.7 can be approximated as follows:

$$\bar{p}_\pi(X) = p(X_{i_1}|X_{i_2})p(X_{i_2}|X_{i_3}) \dots p(X_{i_{n-1}}|X_{i_n})p(X_{i_n}) \tag{2.10}$$

where $\pi = i_1 i_2 \dots i_n$ is a permutation of the numbers between 1 and n.

It is not possible to capture all possible joint distributions of n variables using only pairwise conditional probabilities. Hence, the problem becomes how to choose a chain of pairwise conditional probabilities to approximate the true joint distribution $p(X)$ of the training samples as closely as possible.

Formally, assuming the distribution $\bar{p}_\pi(X)$ uses π as an ordering for the pairwise conditional probabilities, our goal is to choose the permutation π that maximises the agreement between $\bar{p}_\pi(X)$ and the true distribution $p(X)$. The agreement between two distributions can be measured by the Kullback-Liebler divergence:

$$\begin{aligned} D(p||\bar{p}_\pi) &= \sum_X p(\log p - \log \bar{p}_\pi) \\ &= E_p \log p - E_p \log \bar{p}_\pi \\ &= -h(p) - E_p[\log p(X_{i_1}|X_{i_2})p(X_{i_2}|X_{i_3}) \dots p(X_{i_{n-1}}|X_{i_n})p(X_{i_n})] \\ &= -h(p) + h(X_{i_1}|X_{i_2}) + h(X_{i_2}|X_{i_3}) + \dots + h(X_{i_{n-1}}|X_{i_n}) + h(X_{i_n}) \end{aligned} \tag{2.11}$$

where $h(\cdot)$ is entropy. The optimal π is the one which minimises this divergence. The first term in the divergence does not depend on π. Therefore, we only need to minimise the rest of the terms. However, it is not practical to enumerate all $n!$ permutations to obtain the optimal solution, and thus a simple greedy search is employed in (de Bonet, Isbell, and Viola 1997). The model $\bar{p}_\pi(X)$ is estimated every generation using the selected superior individuals.

2.4.4 Multivariate Dependence Model

It is conceivable that more complex dependence (than pairwise interaction) may exist among genes. Therefore, methods that can handle multivariate interaction have been proposed. We briefly describe a typical work – the Bayesian Optimisation Algorithm (BOA) (Pelikan, Goldberg, and Cantú-Paz 1999). BOA factorises the joint probability into a Bayesian Network (BN).

A BN is a probabilistic graphical model. It has two components: a structure S and a set of probability distributions on S. The structure S for X represents a set of conditional dependence assertions on the variables on X. It is usually described as a Directed Acyclic Graph (DAG). Given structure S, Pa_i^S is the set of parents of variable X_i. In the structure S, X_i and variables other than its parents, $\{X_1, \ldots, X_n\} \backslash Pa_i^S$, are independent given Pa_i^S, $i = 2, \ldots, n$. Thus the factorisation of $p(X)$ of Equation 2.7 is as follows:

$$\begin{aligned} \hat{p}_\pi(X) &= p(X_1) \cdot p(X_2|Pa_2^S) \cdot \cdots \cdot p(X_n|Pa_n^S) \\ &= \prod_{i=1}^{n} p(X_i|Pa_i^S) \end{aligned} \tag{2.12}$$

As can be seen, the BN is a more general model than the models described in previous sections. In each generation, the BN is re-estimated. Since the BN is a more complex model, the estimation is more difficult. It includes both estimation of the structure S, and also of its probability distributions. Usually, a greedy search is employed for this estimation. To measure the goodness of the model, a number of

scoring metrics have been proposed. A more detailed description of BOA and other EDA work considering multivariate dependence are referred to in (Mühlenbein and Mahnig 1999; Harik 1999; Pelikan, Goldberg, and Cantú-Paz 1999; Etxeberria and Larrañaga 1999).

In this section, only some typical works in conventional EDA have been briefly reviewed. For a comprehensive survey of this field, readers are referred to (Pelikan, Goldberg, and Lobo 1999; Larrañaga and Lozano 2001; Bosman and Thierens 1999).

2.4.4.1 Learnable Evolution Model

In addition to the conventional EDA works described above, a further interesting work considering complex interaction among multiple genes is the Learnable Evolution Model (LEM) (Michalski 2000).

The basic procedure of LEM is similar to conventional EDAs. A model, which is a set of inductive hypotheses in LEM, is inferred from high-fitness individuals. The new population is then created according to this model.

The model used in LEM differentiates it from other EDA works of this subsection. Instead of a conventional probabilistic model, a model consisting a set of inductive hypotheses in the form of annotated predicate calculus (Michalski 1983) is employed (Other models from Machine Learning can potentially be used as well, such as decision tree, neural network and propositional logic). It is not hard to see that, although they are in different forms, the function of the model in LEM is similar to that of models in EDA, i.e. to bias the search toward promising areas of the search space.

LEM can be loosely understood as an EDA approach with a multivariate dependence model. In the current version of LEM, the model is a set of attribute rules, which describes the interactions among multiple genes. Hence, the model in LEM can also be roughly viewed as a probabilistic graph model with only binary probabilities, i.e.

either 1 or 0. Therefore, although the learning methods and the forms of models are different, LEM is consistent with the other conventional EDA works in this subsection.

2.4.5 Inference of Probabilistic Graph Model

Besides the traditional perspective of model dependence, there is an alternative perspective for viewing EDA. That is the perspective of probabilistic graph model. Because most of the models used in EDA can be regarded as a kind of probabilistic graph model, the EDA works can be conceptualised into three categories based on the learning methods of probabilistic graph models:

1. inferring only the probability of the probabilistic graph model (assuming a fixed structure),
2. inferring only the structure of the probabilistic graph model (assuming a crisp binary value of probability),
3. inferring both the structure and probability of the probabilistic graph model.

On this basis, most EDA models require inference of both structure and probability of the probabilistic graph model and therefore fall into the third category. Those which assume independence among genes, such as PBIL, and thus do not learn the structure, belong to the first category, while LEM alone - so far - belongs to the second category.

2.5 Estimation of Distribution Algorithms with Tree Representation

2.5.1 Introduction

In GA, a solution or individual is usually a one dimensional array. In other words, an individual is a linear string. Most EDA research is based on this GA-style linear string representation because of the close relation between EDA and GA. There is only a very limited amount of work on more complex tree representations (Salustowicz and Schmidhuber 1997; Yanai and Iba 2003; Sastry and Goldberg 2003; Ratle and Sebag 2001; Tanev 2004; Bosman and de Jong 2004), resembling Genetic Programming (GP) (Cramer 1985; Schmidhuber 1987; Koza 1992). Given small number of publications in this area, it is very unfortunate that some of the works in this area often are not fully aware of the existence of the others. This section aims at providing a comprehensive and critical survey on these works. For simplicity, we refer to the idea of applying EDA approaches to estimate the distribution of GP-style tree form solutions as EDA-GP.

Although EDA-GP is in its infancy, the idea of EDA-GP is not entirely new. At first, this type of learning was applied in forms such as a kind of adaptive genetic operator as a mechanism for learning modularity, i.e. as various forms of modification of conventional GP. Later, this research became more and more independent of conventional GP and methods which systematically learn mathematical model were proposed. EDA-GP, and EDA in general, is not just about improving EC performance, but more importantly, about providing insight into aspects of problem decomposition and solution distribution.

There are two main streams of works in EDA-GP. The first stems from Probabilistic Incremental Program Evolution (PIPE) (Salustowicz and Schmidhuber 1997) and other derived works based on the probabilistic model proposed in PIPE. The second is a stream of grammar based work. These two streams will be discussed separately.

In this section, we describe related works in conventional GP in Section 2.5.2. Two distinctive streams of EDA-GP work are reviewed in Section 2.5.3 and Section 2.5.4, respectively.

2.5.2 Learning in Genetic Programming

In conventional GP research, for a long time, the knowledge encoded in the population, and the value of using it, have been a focus of study. Consequently, numerous methods have been proposed to utilise this knowledge, for example adaptive genetic operators, and learning modularity.

Although a significant proportion of these kinds of studies are ad hoc and lack a solid theoretical support, studies in this direction did provide some insight into how GP works and as a result have had an important influence on EDA-GP research.

2.5.2.1 Modularity and Building Blocks

Modularity and building blocks are related to the processes of hierarchical problem solving and decomposition. Building blocks (BBs) are defined as frequently appearing sub-trees in good individuals. If building blocks can be correctly identified and used, the performance of GP may be significantly improved.

This line of research includes Automatically Defined Functions (ADF) (Koza 1992), Genetic Library Builder (GliB) (Angeline and Pollack 1994) and Adaptive Representation (AR) (Rosca and Ballard 1994). The basic idea is that during the search, the sub-trees are identified, either heuristically or by means of evolution, and are then explicitly encapsulated in some form as one entity, so that they can be reused later on.

2.5.2.2 Permutation and Crossover

Permutation and crossover are closely related to Building Blocks (BBs). Since discovery and utilisation of BBs are important aspects of GP, it is important to adapt genetic operators so that they can help to preserve and promote BBs.

In (Angeline 1995), two self-adaptive crossover operators, selective self-adaptive crossover and self-adaptive multi-crossover, were proposed. These new operators adaptively determine where crossover will occur in an individual. Experimental results demonstrate that both of these self-adaptive operators perform as well or better than standard Genetic Programming crossover.

Recombinative guidance for GP is proposed in (Iba and de Garis 1996). In this approach, all the performance values for all the sub-trees of a GP tree are calculated. These values are then used to decide which sub-tree will be chosen to apply GP operations. Although GP with recombinative guidance performs well on some problems, the definition of sub-tree value is problem dependent.

2.5.3 PIPE Model

Having discussed the related work in conventional GP, we now move on to EDA-GP. The earliest EDA-GP work was the Probabilistic Incremental Program Evolution (PIPE) (Salustowicz and Schmidhuber 1997). PIPE is motivated by the corresponding work in conventional EDA. A number of studies based on the probabilistic model of PIPE have followed, including ECGP (Sastry and Goldberg 2003) and EDP (Yanai and Iba 2003).

Interestingly, PIPE and its related works can fit into the same framework as conventional EDA. This is possible because the prototype tree of PIPE is basically a model assuming no dependence among random variables, while EDP considers pairwise dependences only, and ECGP extends this to multivariate dependence.

2.5.3.1 Probabilistic Incremental Program Evolution

PIPE (Salustowicz and Schmidhuber 1997) uses the Probabilistic Prototype Tree (PPT) to represent the probability distribution of tree form programs. Its basic algorithm is consistent with EDA's illustrated in Figure 2.6 and the learning method to update PPT resembles the probability learning methods of PBIL (Baluja 1994).

PIPE iteratively generates successive populations of tree form programs according to an adaptive probability distribution over all possible programs, represented as a Probabilistic Prototype Tree (PPT). For example, in Figure 2.8 which is adopted from (Salustowicz and Schmidhuber 1997), the left hand side is a PPT where each node is a probability vector (more precisely each node is a random variable), indicating the probability of occurrence of different symbols. The right hand side is one of the possible GP trees, sampled from the PPT.

More specifically, the basic procedure of PIPE is as follows. Firstly, probabilities on the prototype tree are initialised uniformly. Starting from the root, we keep visiting nodes on the prototype tree until a valid tree, acting as an individual, is generated. When visiting each node, we choose a symbol according to the probabilities in the probability table of that node. In this way, a set of individuals (i.e. a population) is generated. Good individuals are then selected from the population and all of the probabilities of entries in the probability table, which were used to generate those selected individuals, are increased. In other words, the probabilities of generating those good individuals are increased. Therefore, in each iteration, the probability distribution is updated using the best programs. Thus, the structures of promising individuals are learnt and encoded in the PPT. This is the learning process for a prototype tree. A new population is then generated from this updated prototype tree and a new iteration starts.

PIPE is the first work of its kind. However, it can be regarded as a tree-based extension of the linear string based PBIL (Baluja 1994). In PIPE, each node of the PPT is treated as an independent random variable in that its probability is learnt

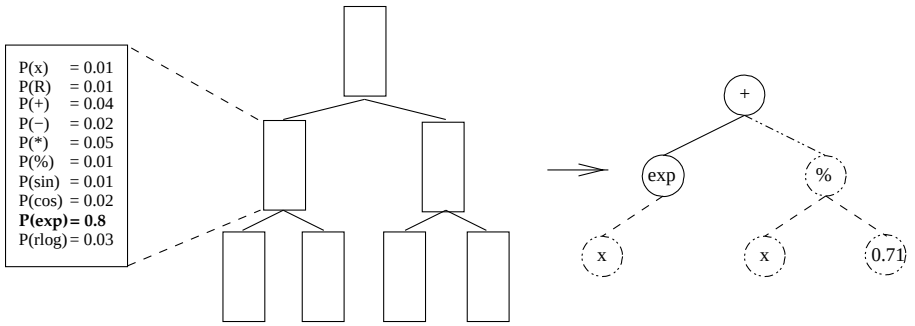


Figure 2.8: Probabilistic Incremental Program Evolution

independently from the other nodes. Therefore, what PIPE tries to learn is the probability of particular functions. PIPE also implicitly assumes that the building blocks are position dependent, i.e. in the PPT, the useful sub-trees/building blocks are attached to specific positions and cannot be easily moved to other positions.

2.5.3.2 Extended Compact Genetic Programming

ECGP (Sastry and Goldberg 2003) is a direct extension of ECGA (Harik 1999) to the tree representation. It is based on PIPE prototype tree.

In ECGA, Marginal Product Models (MPMs) are used to model the interaction among genes, represented as random variables, given a population of Genetic Algorithm individuals. MPMs are formed as a production of marginal distribution on a partition of random variables. For example, in ECGA, a MPM for a four-bit problem is

$$[1,3][2][4]$$

It could represent that the 1st and 3rd genes have intensive interaction and the 2nd and 4th genes are independent. That MPM would consist of the following marginal probabilities. $\{p(x_1 = 0, x_3 = 0)$, $p(x_1 = 0, x_3 = 1)$, $p(x_1 = 1, x3 = 0)$, $p(x_1 = 1, x_3 = 1)$, $p(x_2 = 0)$, $p(x_2 = 1)$, $p(x_4 = 0)$, $p(x_4 = 1)\}$, where x_i is the value of the ith gene.

This idea has been extended to the GP tree representation in ECGP. ECGP is

based on PIPE prototype tree and thus each node in the prototype tree is a random variable. ECGP decomposes or partitions the prototype tree into sub-trees, and the MPM factorises the joint probability of all nodes of the prototype tree, to a product of marginal distributions on a partition of its sub-trees.

A greedy search heuristic is used to find an optimal MPM mode. A metric is needed to score MPM. Minimum Description Length (MDL) (Rissanen 1989) is taken as a quality measure of the MPM model in ECGP. The MDL metric is the sum of two components:

$$minimise \ \ (C_m + C_p) \tag{2.13}$$

where model complexity C_m is the complexity of the partition, and compressed population complexity C_p, also known as the accuracy of the model, represents the cost of representing the population given the model. It is very intuitive that the preferred model should be simple and accurate. Further, as although a too-complex model may have better accuracy, it is very probable that it will not generalise well, i.e. it will not work well on unseen data. Conversely, a too-simple model may have very bad accuracy, and thus should also be penalised. MDL, and the very similar metric, Minimum Message Length (MML), are important information-theoretic inference methods, which are both intuitive and also have solid theoretical support. The concept of MML and MDL will be revisited later.

ECGP can represent the probability distribution for more than one node at a time. Unfortunately there is no explicit definition of partition in (Sastry and Goldberg 2003). In particular, it is unclear whether it is required to be a fully connected sub-tree, or just a set of nodes with no connectivity constraint. How the partition is found, is not fully specified, either. However, it is clear that it extends PIPE in that the interactions among nodes are considered.

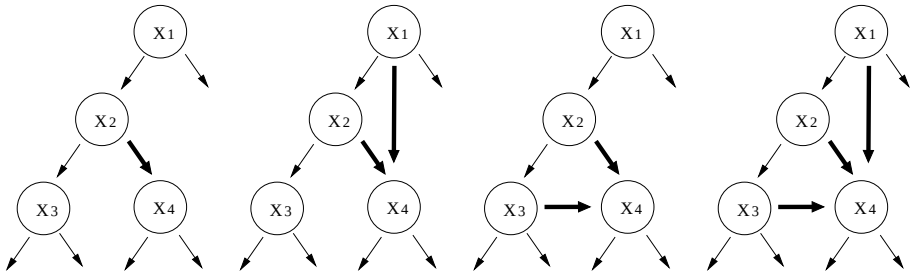


Figure 2.9: Probability distribution model in EDP

2.5.3.3 Estimation of Distribution Programming

Estimation of Distribution Programming (EDP) is another extension of PIPE. Instead of treating each node as an independent random variable, EDP tries to model the conditional dependency among adjacent nodes in the PIPE prototype tree.

It is argued in (Yanai and Iba 2003) that strong dependence should exist between each particular node and its parent, grandparent and sibling nodes. Some possible combinations of these dependences are illustrated in Figure 2.9. The basic structure is again a PIPE prototype tree. For example, if node X_4 is under examination, then the thick lines indicate the important dependences. The right-most model is the most comprehensive model, which captures all the dependences believed to be important, while the left-most one is the most simplified model, in which only dependence between one node and its immediate parent node is considered. Because of the computational overhead, among these possible models, the left-most model is implemented in (Yanai and Iba 2003).

2.5.3.4 Summary

A visualised comparison of PIPE, ECGP and EDP can be found in Figure 2.10. Each grey circle stands for a node of a PIPE prototype tree. The dependences considered in each model are illustrated by the bounded regions.

The left most figure in Figure 2.10 corresponds to PIPE. Clearly, each node is treated

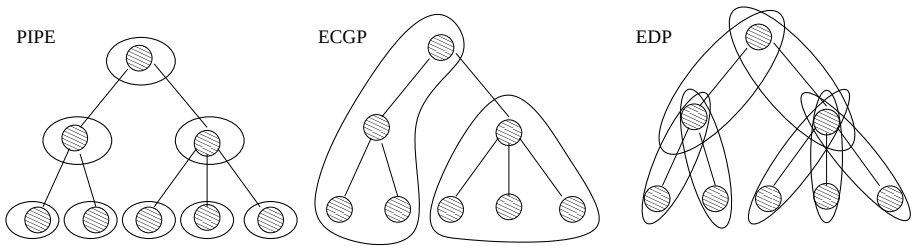


Figure 2.10: Comparison of different probabilistic models in EDA-GP

as an independent variable. The figure in the middle is ECGP. The prototype tree is divided into sub-trees and there is no dependence considered among different sub-trees. The right most figure is EDP, the conditional probability of the node given its parent is modelled.

In summary, all these works are based on the prototype tree of PIPE where probability tables of a Prototype tree are organised in a tree form. Therefore, these PIPE-based works can well handle the GP-style tree structure. In the original PIPE, each node is an independent random variable and thus its probability does not depend on any neighbouring nodes. The extensions, made in EDP and ECGP, make it possible to consider interactions among nodes. Further, the PIPE prototype tree does not have a problem in handling individuals with varying complexity. However, we cannot see any obvious way for PIPE-based methods to handle building blocks with no fixed position. These issues will be revisited in the next chapter.

2.5.4 Grammar Model

Grammar model-based EDA-GP has a close connection with GGGP. In GGGP, the grammar, as a formal model, effectively imposes a constraint on the search space, but the main search mechanism is still conventional genetic search.

Grammar model-based EDA-GP takes grammars as probabilistic models, just like any other probabilistic model used in EDA research. Grammars are well-studied formalism, originally proposed to model the internal hierarchical structure of lan-

guages, either natural languages or formal languages. They are particularly suitable for modelling GP-style tree structures because GP-style tree structures are just another kind of hierarchical structure.

Grammar model-based EDA-GP work can be also fitted into the same framework as conventional EDA. In a grammar model, each rule describes the dependence between the LHS symbols and the RHS symbols. Therefore, it is primarily a model of pairwise dependence. Through the chain of dependency, it is also adequate to describe structures which have more than two closely related symbols. This thesis is particularly interested in grammar model based EDA-GP.

In this section, we first describe some related works in GGGP. EDA-GP with a grammar model is discussed in the two separate subsections that follow. These subsections are divided according to the types of grammar learning methods within them.

2.5.4.1 Conventional GGGP with Grammar Learning

In conventional GGGP, it had been noticed early on that a grammar can not only be used to impose the initial bias, but can also be revised during search to further bias the search according to updated available experience. There are two studies in this field.

Whigham's work In (Whigham 1995a), grammar refinement is introduced into conventional GGGP. More specifically, it is a conventional GGGP system but the grammar is refined during the search, and new individuals generated from the refined grammar are fed back into the population.

The refinement of the grammar has two components. The first is to update the probabilities of rules. Merit, derived from probability, is attached to each rule to reflect the frequency of the rule use. This merit is updated according to the superior individuals. Then, when generating new individuals, the merit has a similar function

as the probability in SCFG. The second component is to add new production rules. New productions are learnt from the superior individuals, and they are chosen in a way to have minimum impact on the original grammar.

The population in the new generation not only has individuals obtained by applying genetic operators, but also individuals generated from the updated grammar.

As can be seen, this work is essentially a conventional GGGP, because its main search mechanism is the conventional genetic search, and the incorporation of individuals generated from the refined grammar is only an aid to further bias the search. The grammar learning is also ad hoc. However, the importance of this work is that it first showed that grammars may be a good model for incrementally guiding the search.

Tanev's work Another similar approach is proposed in (Tanev 2004). Tanev incorporated learning Stochastic Context-sensitive Grammars (SCSG) into conventional GGGP.

This work is in the context of a dynamic environment. At the end of the run, the grammar is learnt from the best-of-run individuals. It is then moved to the new environment. In the new environment, part of the new population is generated from the learnt grammar, and the mutation operator must also respect the learnt grammar.

The grammar used here is an SCSG, which in this circumstance can be loosely understood as an SCFG with a context constraint. With the extra constraint, whether a rule is admissible is not only decided by matching the LHS, but also by matching the context. In (Tanev 2004), the probabilities and context are learnt in order to favour some rules, and to restrict the places where a rule can be applied.

More specifically, SCSG grammar learning happens at two levels. One is the probability learning. The other is the context learning. A fixed amount of context is added to the grammar to make the grammar more specific, so that some areas of the search space can be more intensively investigated.

This research is an enhancement of GGGP with the aid of a grammar model. The grammar learning occurs only once, and the main search mechanism is still genetic search. Through a comparison with conventional GGGP, this work empirically verifies that grammars can be used to efficiently bias search to promising areas and thus obtain superior performance.

2.5.4.2 Learning Parameters of a Grammar Model

Having discussed the conventional GP with the aid of grammar learning, we move on to EDA-GP with a grammar model. Regarding the grammar model, we are largely interested in Stochastic Context-free Grammars (SCFGs). We know that an SCFG model can be understood as a normal Context-free Grammar (CFG) with a probability component. Therefore, the inference of an SCFG model usually consists of inferring these two parts, namely the structure of the SCFG (which is essentially a CFG) and its associated probability component. Accordingly, we identify two streams of EDA-GP with grammar models. One stream learns the probability only. The other learns both structure and probability. The former will be discussed in this section and the latter will be left to the next section.

Note that, theoretically, there is no intrinsic difference between grammar structure learning and probability learning. The grammar structure is actually only a form of probability where absence of a particular rule implies that the probability attached to the rule is zero. Therefore, theoretically we may have a grammar which has all possible rules, and where probability learning will assign probability 0 to the unnecessary or incorrect rules. Distinguishing grammar structure learning and probability learning is largely a matter of implementation convenience, efficiency and comprehensibility.

Stochastic Grammar-based GP Stochastic grammar-based GP (SG-GP) (Ratle and Sebag 2001) is an interesting but overlooked work proposed by Ratle et. al. To the best of our knowledge, it is the earliest attempt to introduce grammar

models into EDA-GP.

Essentially, what SG-GP does, is to learn the probability of an SCFG while keeping the structure of the grammar fixed. The basic algorithm of SG-GP is consistent with the EDA algorithm described in Section 2.3.1, i.e. it is an iteration of model learning and sampling.

SG-GP starts with a CFG and weights attached to each rule (the probability can be obtained by normalising the weights). Initially, because we have no prior knowledge, all of the weights are set to equal values (corresponding to a uniform distribution). Generating individuals from the grammar is similar to generating individuals from an SCFG, as discussed previously. Also, similarly at each generation, the probability is updated. To do this, the weight of those rules which contribute to superior individuals are increased, while the weights of those rules involved in generating inferior individuals are decreased. More precisely, assuming rule r_i is used in a superior individual and r_j in an inferior individual, their weights w_i and w_j are updated as follows:

$$\begin{aligned} w_i &\leftarrow w_i(1+lr) \\ w_j &\leftarrow \frac{w_j}{1+lr} \end{aligned} \tag{2.14}$$

where lr is a predefined learning rate.

There are two variants of SG-GP proposed in (Ratle and Sebag 2001), namely scalar and vectorial SG-GP. What we have discussed is scalar SG-GP while vectorial is a straightforward extension of scalar SG-GP. Note that in scalar SG-GP, tree position does not play a role. Each rule can be used to generate any position of the tree as long as its LHS matches the nonterminal. However, this causes serious problems when the number of rules is small, which is usually the case. For example, the grammar in Figure 2.1 which is a commonly used grammar for symbolic regression problems, has only 11 rules. No matter how we update the probability attached to each rule, it is unlikely that this grammar would be able to hold enough information to progressively bias the search. In other words, the model is too simple to be able to adequately describe the solution space. For example, suppose one particular rule

is only beneficial if it is applied at depth d, but not at depth d', where $d \neq d'$. In scalar SG-GP there is no way to record this information.

To alleviate this problem, vectorial SG-GP was proposed. In vectorial SG-GP, a weight vector is attached to each rule. Each element of the vector then represents the weight of the rule at a particular depth. Therefore, the depth information is used to effectively increase the total number of rules, i.e. the model complexity was increased.

However, in both scalar and vector SG-GP the overall structure of the grammar is fixed and it does not change with the progress of the search. Clearly, in this method, because of the fixed grammar structure, i.e. the number of rules is fixed, the complexity of the grammar model does not change. Therefore, the search will either stop very quickly, especially when the number of rules is very small, which is usually the case, or the search will converge very slowly if too many redundant rules are involved in the probability learning.

2.5.4.3 Learning Structure and Parameter of Grammar Model

Besides the work reported in this thesis, Bosman's work (Bosman and de Jong 2004) is, to our best knowledge, the only work which intentionally extends EDA to EDA-GP, and infers a full grammar (both grammar structure and probabilities).

In (Bosman and de Jong 2004), we understand that the derivation trees are not preserved - an unusual practise in GGGP - so ambiguity occurs when re-parsing the individual. Therefore, a large amount of effort is put into correctly re-parsing the derived sentence (individual), and a highly complicated method is used.

The basic idea of (Bosman and de Jong 2004) is as follows. It starts with a minimum SCFG. In each iteration, the grammar is learnt and sampled to obtain the next population. The grammar structure learning method is rule expansion (with some constraints to ensure correct parsing). The learning method is a greedy search

similar to most EDA work, and the scoring metric is MDL. In MDL, the model complexity term is measured by the number of symbols in the grammar. When estimating the probabilities of rules, depth is introduced as an extra dimension, similar to SG-GP.

Through the expansion of grammar rules, more production rules are added and their probabilities are estimated accordingly. Thus the grammar becomes more specific, enabling the system to progressively bias search.

2.5.5 Inference of Probabilistic Graph Model

In Section 2.4.5, we provided an alternative view of conventional EDA. The same view can be applied to EDA-GP as well, as most of the models used in EDA can be regarded as a kind of probabilistic graph model. Based on the graph model learning method, EDA-GP can be divided into the same three categories as with conventional EDA:

1. inferring only the probability of the probabilistic graph model (assuming a fixed structure) (Salustowicz and Schmidhuber 1997; Yanai and Iba 2003; Ratle and Sebag 2001; Abbass, Hoai, and McKay 2002),
2. inferring only the structure of the probabilistic graph model (assuming the crisp binary value of probability - not, to our knowledge, studied yet),
3. inferring both the structure and probability of the probabilistic graph model (Sastry and Goldberg 2003; Bosman and de Jong 2004; Shan, McKay, Abbass, and Essam 2003; Shan, McKay, Baxter, Abbass, Essam, and Nguyen. 2004).

This perspective provides a unified view to understand EDA-GP regardless of the specific form of model. Interestingly, we are not aware of any work of category 2 in EDA-GP, even though such approaches do exist in conventional EDA (LEM). However, we cannot see any intrinsic difficulty in applying this kind of method to EDA-GP.

2.5.6 Summary

In this section, we reviewed the major works in EDA-GP. It is not hard to see that this field is still in its infancy, given the limited number of studies. However, as tree representation, which has been widely used in GP, is suitable for a number of problems, the limited research in EDA-GP is a significant gap.

We have identified two streams of works in EDA-GP. One is based on the PIPE prototype tree, while the other is based on a grammar model. PIPE-based work is relatively less computationally expensive than grammar-based work, while the latter is more flexible in terms of capturing different kinds of interactions among nodes. In this thesis, we emphasise the latter approach, i.e. EDA-GP with grammar model. The relationship among these existing EDA-GP works, in particular with respect to the works proposed in this thesis, will be further discussed in Section 3.3.5 and the Figure 3.5 in that section will provide an visualised description of this relationship.

The lack of research in EDA-GP may have two reasons. The first is the strong connection between EDA and GA. Since EDA research started from addressing the problems in GA, it is natural that most studies focus on the GA side of EDA research. The second reason is the complication due to the complex tree structure. Suitable probabilistic models are needed to model the GP-style tree structure but common probabilistic models are not directly applicable to EDA-GP. Because of this, we have witnessed slow progress in this field. The earliest EDA-GP work (Salustowicz and Schmidhuber 1997) dates back to 1997, and there appears to have been no subsequent work until 2001 (Ratle and Sebag 2001). Most EDA-GP publications have appeared in the last three or four years, which may suggest that appropriate probabilistic models are emerging; we are convinced, and also hope to demonstrate, that a stochastic grammar model is one of them.

2.6 Ant Colony Optimisation and Automatic Program Synthesis

Sometimes, EDA may be viewed as a Model-based Search (MBS) (Zlochin, Birattari, Meuleau, and Dorigo 2004). In MBS algorithms, candidate solutions are generated using a parameterised probabilistic model that is updated using the previously seen solutions in such a way that the search will concentrate in the regions containing high quality solutions.

Ant Colony Optimisation (ACO) (Bonabeau, Dorigo, and Theraulaz 1999) is another well-studied field of MBS. The similarity between ACO and EDA methods which only consider univariate probability distribution, such as PBIL (Baluja 1994), can be easily seen because of the similarity in the probabilistic model and the probability updating methods. However, it is harder to make a direct analogy between EDA considering more complicated dependences and ACO at the level of probabilistic model and mechanism of probability updating although they are almost identical at the abstract level. For more detailed description of MBS and the discussion of ACO and EDA from the perspective of the MBS, please refer to (Zlochin, Birattari, Meuleau, and Dorigo 2004).

In this section, given the major emphasis of the thesis is EDA-GP, a number of closely related works using ACO to synthesise programs will be reviewed. With a few exceptions, most ACO based automatic synthesis methods are *ad hoc* and, with very limited theoretical justification, simply try to translate the conventional program synthesis problem by converting the program space to a representation, such as a graph, amenable to search by ACO. In the following subsections, as with the review of EDA-GP in the previous section, these works are grouped into grammar based and non grammar based categories.

2.6.1 Grammar Based Works

ant-TAG (Abbass, Hoai, and McKay 2002) use grammar and ACO to synthesise programs. In this work, Tree-adjunct Grammars (TAGs) (Joshi, Levy, and Takahashi 1975), instead of CFGs, are used. The individuals are assembled by combining elementary trees which can be loosely understood as repetitively applying rules of an SCFG to nonterminal symbols.

ant-TAG starts with a given grammar with uniform distribution because it does not have any prior knowledge. At each generation, the probabilities are updated by increasing the probabilities of those rules which contributed to superior individuals.

Note that, in each iteration, only the probabilities are updated while the structure of the grammar is fixed. From the perspective of EDA-GP, ant-TAG is very similar to SG-GP (Ratle and Sebag 2001), except that different grammar formalisms are used. We anticipate that ant-TAG may suffer from the same difficulties as SG-GP due to the fixed grammar structure, namely due to fixed complexity, it might not be able to capture all the necessary information for constructing a good solution. The name ant-TAG reflects the facts that the probability update mechanism in ant-TAG is motivated by Ant Colony Optimisation (Bonabeau, Dorigo, and Theraulaz 1999) and TAG grammars are involved.

Generalised Ant Programming (GAP) (Keber and Schuster 2002) is a CFG grammar based work. It is different from ant-TAG in that it deals with a model of non-fixed structure. Instead of only updating the probabilities of grammar and keeping the structure unchanged, GAP records the whole path which an ant has visited. Therefore, it is very much like PIPE based EDA-GP work, especially EDP, because the probabilities in GAP are attached to the rules, and thus are conditional probabilities, representing the pairwise interactions.

However, because the probabilities in GAP are attached to the rules, they are conditional probabilities representing the pairwise interactions, while probabilities in PIPE are independent of other nodes.

2.6.2 Non-grammar Based Works

Among the non-grammar based work, Ant Programming (AP)(Roux and Fonlupt 2000) appears to be the earliest attempt at using ACO to synthesise programs. It is elegant and consistent with PIPE based EDA-GP works. ACO search is used to explore good paths along the prototype tree of PIPE, representing a program. The probability update mechanism (pheromone update policy) is the major characteristic that discriminates it from PIPE.

Other non-grammar based works, such as Ant Colony Optimisation (Green, Whalley, and Johnson 2004) and Ant Colony Programming (ACP) (Boryczka and Czech 2002), employ arbitrarily predefined graphs whose nodes could be either a GP symbol (terminal or nonterminal) or an arbitrarily chosen program statement. ACO search is used to find a path representing a program. Slightly different, Grid Ant Colony Programming (GACP) (Rojas and Bentley 2004) introduces a temporal index, which closely corresponds to the depth constraint of some EDA-GP methods, so that the probability (pheromone) can have a depth reference.

2.7 Conclusion

This section introduces the necessary background on which this thesis is based. The focus of this thesis is EDA with tree representation, with emphasis on grammar as a probabilistic model. Therefore, at the start of section, grammar and Grammar Guided Genetic Programming are discussed. The following sections presented the basics of EDA, especially EDA with GP-style tree representations, referred to as EDA-GP. A critical and comprehensive survey of EDA-GP is presented. Because of the close relationship between EDA and Ant Colony Optimisation (ACO), related works in this area are also discussed at the end of the section.

Chapter 3

Program Distribution Estimation with Grammar Models

3.1 Introduction

There has been growing interest in Estimation of Distribution Algorithms (EDA) (Bosman and Thierens 1999; Larrañaga and Lozano 2001; Müehlenbein and Paaß 1996; Pelikan 2002). EDA mainly uses a linear string representation, resembling an individual of Genetic Algorithms (GA), because conventional EDA research is closely related to GA.

There has only been a very limited amount of work on more complex tree representations (Salustowicz and Schmidhuber 1997; Yanai and Iba 2003; Sastry and Goldberg 2003; Ratle and Sebag 2001; Tanev 2004; Bosman and de Jong 2004), resembling Genetic Programming (GP). However the success of GP demonstrates that there is an important class of problems for which a tree representation is suitable, so that the limited consideration of tree representation in previous EDA literature is a significant gap in the research. EDA not only has the potential to improve GP performance on some problems, but also provides a new perspective for understanding GP. Our work aims to help to fill this gap.

In this chapter, we propose a framework, called **Pro**gram **Di**stribution Estimation with **G**rammar Models (PRODIGY), which extends conventional EDA to a more complex tree representation. The basic algorithm of PRODIGY is consistent with the EDA algorithm. The core of this framework is a grammar model. In this research, we show, both theoretically and experimentally, that a grammar model has many of the properties we need for the estimation of distribution of tree form solutions.

The purpose of this chapter is to fulfil the following goals:

1. to clarify why grammars, in particular SCFG, are a suitable probabilistic model for EDA-GP,
2. to propose a high level algorithm for introducing grammars into EDA-GP,
3. to address some considerations regarding the implementation of PRODIGY,
4. to establish a connection between grammar based EDA-GP and grammar learning in Natural Language Processing (NLP).

This Chapter is organised as follows. Properties that a probabilistic model should have for EDA-GP are discussed in Section 3.2. The PRODIGY framework is proposed in Section 3.3. Some important issues related to the implementation of PRODIGY are presented in Section 3.4. In Section 3.5, we briefly introduce two specific methods, developed under the PRODIGY framework, which will be presented in detail in later chapters. Conclusions are given in the last section.

3.2 Lessons from GP Research in Searching for Models

Because the basic idea of EDA is to approximate the true distribution using a model $\mathcal{M}$, it is vital to choose an appropriate model $\mathcal{M}$. Consequently, conventional EDA research, which often employs a GA-style linear string to encode individuals, heavily focuses on finding a suitable probabilistic model $\mathcal{M}$. The GA literature supports the belief that there are dependencies between genes (also known as linkages). In the EDA literature, many different kinds of probabilistic models have been proposed to represent linkage.

Estimation of the distribution of tree form solutions is far more complex than simply applying conventional EDA to a tree representation. One of the major difficulties lies in finding an appropriate probabilistic model. We may take lessons from GP research in this respect. Based on current GP understanding, we are convinced that a good model for EDA-GP should have the properties listed below. In addition to the theoretical discussion of these properties in this chapter, the relevance of some of these properties to problem solving will be empirically verified in the next chapter.

3.2.1 Internal Hierarchical Structure

The tree representation has an internal structure – an intrinsic hierarchical structure. The model for EDA-GP needs to be able to represent this structure. Thus when sampling the model, i.e. when generating individuals from the model, a valid tree structure should be guaranteed. In this context, a valid tree is a tree that can be evaluated and complies with the given constraints, such as legitimate terminal and non-terminal symbol sets, depth limit and typing.

3.2.2 Locality of Dependence

In conventional EDA, the model employed does not usually assume specific dependencies between adjacent loci - a reasonable assumption for GA representations. However, in a tree structure, dependence exhibits strong locality, and this is the

primary dependence which we should consider in tree representation. For example, dependencies (relationships) between parent and child nodes are expected to be stronger than among other nodes.

Another perspective for viewing this locality of dependence comes from semantics. When evolving a linear structure, as conventional EDA does, usually we assume the semantics are attached to the locus. For example, when using GA to solve a travelling salesman problem, one common encoding method uses each locus to represent a single step in which one specific city is visited. By contrast, in tree representation, the meaning of the node is clearly defined by the symbol attached and the effect of the node has to be interpreted in its surrounding context. For example, in an Artificial Ant Problem (Koza 1992), one of the standard GP benchmark problems, the node *move* would have the same meaning anywhere. Its effect would depend on the current location of the ant, i.e. the surrounding symbols which previously positioned the ant. Taking another example from symbolic regression problems, the symbol '+' returns completely different values depending on the two operands. The model chosen for evolving tree structure has to be able to represent this strong local dependence, as well as dependence on a larger scale.

3.2.3 Position Independence

Position independence is actually very closely related to locality of dependence. It emphasises that, in GP tree representations, the absolute position of a symbol does not play a large role. Because of locality of dependence, a sub-structure may occur in various locations in different GP trees, but still have an identical or similar contribution to the overall fitness.

This belief is well reflected in various GP schema theories (Koza 1992; O'Reilly and Oppacher 1995; Whigham 1995c). In these studies, a schema is defined as a sub-tree of some kind and absolute position of the sub-tree is often not considered. Consequently, the quality of a schema is determined by its own structure, not just

by where it is. Although there are some recent studies introducing positional information into schema (Rosca 1997a; Langdon and Poli 2002), this appears to be more for the sake of mathematically tractability rather than because it is a desirable characteristic of building blocks.[1]

The prevalence of introns (Angeline 1994) in GP representations provides a second reason for emphasising the importance of positional independence in EDA-GP models. When evolving solutions in GP, it is common that some parts of an individual do not contribute to the fitness, and thus can be removed when evaluating fitness. There is no consensus on the cause of introns or their impact on GP performance, but their existence and importance is amply confirmed. If EDA-GP models are position-dependent, introns which move the location of a particular building block complicate the learning process, since each location has to be learnt separately; in a position-independent model, occurrences of a building block in different locations can reinforce each other. However, a position-independent EDA-GP model is very computationally expensive to obtain. Thus, one of the two methods proposed in this thesis – PEEL – does not completely insist on position independence, but rather allows for an adaptive sliding scale between dependence and independence, which can change appropriately during the searching process.

[1]The differences between position-dependent (rooted) and position-independent (non-rooted) schemata are clearly presented in (Langdon and Poli 2002) but it is hard to find direct evidence in this publication to support the view that position-dependent is superior to position independent in terms of GP performance. One of the major concerns is that the position-independent schema may be counted multiple times in one individual. However, we cannot see any problem with this in problem solving using GP. Real world programming experience has shown that a subprogram should be allowed to be called multiple times (logically equivalent to one piece of code occurs multiple times in one program). This issue of modularity would be further discussed in the Section 3.2.4. Position-dependent schema certainly makes mathematical modelling possible and it is helpful in conceptualising the dynamics of GP. However, we are not convinced it is a desirable property for GP problem solving.

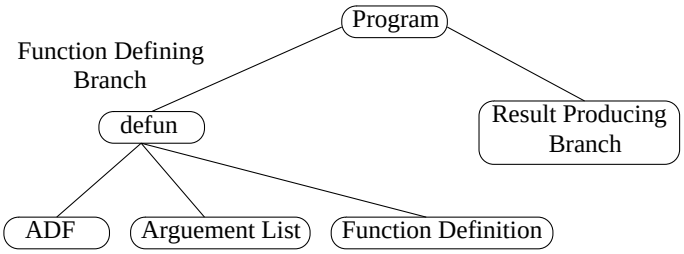


Figure 3.1: Automatically Defined Functions of GP

3.2.4 Modularity

In a tree representation, it is common that building blocks, which are relatively independent sub-solutions, may need to be shared among tree branches and individuals. Therefore, one building block may occur multiple times, either in one individual or across the population.

This has been validated by numerous studies, such as Automatically Defined Functions (ADF) (Koza 1992), Genetic Library Builder (GLiB) (Angeline and Pollack 1994) and Adaptive Representation (AR) (Rosca and Ballard 1994). In these studies, the "useful" sub-trees are identified, either by means of evolution or by some other heuristics, and are then encapsulated as intermediate building blocks so that they can be used multiple times within both one individual and also shared across the population. By identifying and reusing the useful structures, these studies have reported improvement over conventional Genetic Programming.

An example of an ADF can be found in Figure 3.1. When using ADFs, a GP tree has two parts, the result producing branch and a function defining branch. The function defining branch, on the left hand side in Figure 3.1, defines a sub-tree as function so that it can be called multiple times in the result producing branch. The result producing branch is a normal GP tree which can return the value of the entire program, but in addition to using the predefined functions (also called nonterminals in the thesis), it can also refer to the functions defined by the function defining branches.

3.2.5 Non-fixed Complexity

Non-fixed complexity of a GP individual is one of the important properties of GP. In this, it differs radically from GA, which uses a fixed length string to encode individual. GP is aimed at problems where no prior knowledge is available regarding the complexity of their solution. Hence, theoretically, there is no complexity constraint on individual GP trees.

Even if we do impose some limits, such as maximum depth or maximum number of nodes, the individual complexity still varies significantly from individual to individual, from generation to generation. We cannot see how to make a fixed-complexity model, resembling some models in conventional EDA in having a fixed size in terms of number of variables, reflect this variation. Note that variable complexity does not necessarily imply greater complexity. In a fixed complexity model, the initial model must be sufficiently complex to model solutions. Otherwise search will fail. Hence, there is a temptation to start with a high complexity model, leading to high search cost. By contrast a variable complexity system can start with a low complexity model, relying on the learning method to increase its complexity lately.

In order to effectively extend conventional EDA to tree representation, we may take lessons from GP research, which also employs the same representation. From the perspective of current GP research, we have identified the above five properties which are important for probabilistic models of EDA-GP. Based on these properties, a framework for EDA-GP is proposed in the next section.

3.3 Framework of PRODIGY

In this section, we propose our framework, called **Pro**gram **Di**stribution Estimation with **G**rammar Models (PRODIGY). The core of PRODIGY is the use of Stochastic Context Free Grammars (SCFGs). Based on the desirable properties of a probabilistic model that we discussed above in Section 3.2, we argue that SCFGs satisfy

1. Generate a population $\mathcal{P}$ from some initial grammar $\mathcal{M}_0$
2. Select a set of fitter individuals $\mathcal{G}$ from the population $\mathcal{P}$.
3. Estimate an SCFG model $\mathcal{M}$ over $\mathcal{G}$.
4. Sample the SCFG model $\mathcal{M}$ to obtain a set of new individuals $\mathcal{G}'$.
5. Incorporate $\mathcal{G}'$ into population $\mathcal{P}$.
6. If the termination condition is not satisfied, go to 2.

Figure 3.2: High level algorithm of PRODIGY

these criteria well.

3.3.1 Algorithm

The high level algorithm of PRODIGY can be found in Figure 3.2. It is consistent in its essentials with the EDA algorithm. In more detail, assuming an initial model of the problem space $\mathcal{M}_0$, PRODIGY starts with a population $\mathcal{P}$ generated from $\mathcal{M}_0$. A Stochastic Context-free Grammar model (SCFG) $\mathcal{M}$, which will be discussed later, is learnt from the fitter individuals $\mathcal{G}$ selected from this population. A new set of individuals $\mathcal{G}'$ is generated by sampling the learnt model $\mathcal{M}$. The new population is then formed by incorporating $\mathcal{G}'$ into the original population $\mathcal{P}$. The next iteration starts again from this new population.

3.3.2 Grammar Model

The key component in the PRODIGY framework is a Stochastic Context-free Grammar model (SCFG) (Charniak 1993). A Stochastic Context-free Grammar (SCFG), also called a Probabilistic Context-free Grammar, is the probabilistic model which we use in the PRODIGY framework. The formal definition of SCFG can be found in Section 2.1.1. As we foreshadowed in Section 3.2, in this section, we will look at the proposed desirable criteria for an EDA-GP model, to see to what extent SCFG meets them.

Grammars were invented to represent the internal hierarchical structure of natural language in a general way. Consequently, the SCFG model intrinsically respects the tree structure, which is a basic requirement of GP. Therefore, it can represent GP-style tree structure naturally. No extra constraint needs to be introduced.

SCFG grammars can model locality of dependence very well. The probability attached to each SCFG production rule can be regarded as a conditional probability, describing dependencies between parents and children. As mentioned before, this dependence is the primary dependence in tree structures.

A position dependent model, such as in PIPE (Salustowicz and Schmidhuber 1997) or other prototype tree based work, has to learn identical building blocks at different positions separately. The SCFG model does not assume any position dependence. Thus, common structures (building blocks) shared by individuals, even if they are not at the same position, can be represented and preserved by the grammar model. It may be easier to illustrate this point with an example.

Suppose we have selected the two individuals illustrated in Figure 3.3. Between them, one sub-tree occurs in three positions A, B and C. If a model is position dependent, the sub-trees in positions A and C would be considered different because of their different positions in the tree, although they are obviously identical sub-trees. Therefore, a position dependent model cannot gain any learning advantage from the duplication of the sub-trees in positions A and C, while a grammar model, such as Figure 3.4, can reflect this common structure very well. The common structure in Figure 3.3 only occurs once in this grammar, represented through rules 2, 4 and 6 of Figure 3.4. This confirms that the grammar can represent position independent building blocks. Note that the probability component of the SCFG is omitted in Figure 3.4, because it is not relevant to this specific discussion.

The structure constraint imposed by an SCFG is flexible. The SCFG is a constrained probabilistic graphical model, i.e. the common structure can be shared, while the individual generated is guaranteed to be a valid tree structure. This sug-

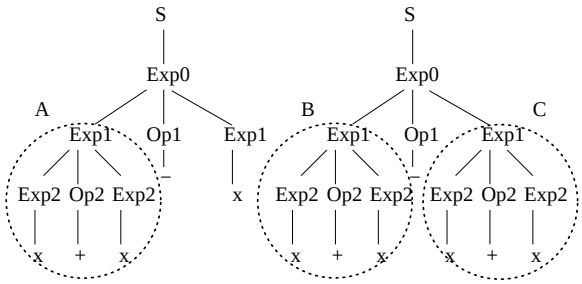


Figure 3.3: An Example of position independence and modularity in tree-shaped individuals

gests that SCFG can represent the modularity in tree representation appropriately. In this respect, the SCFG model differs essentially from models employing a rigid tree structure. With SCFG, a common structure can be used multiple times in an individual, yet not have to be re-learnt each time.

Let us again look at the example used above. The common structure, which is represented by rules 2, 4 and 6, occurs in the grammar model illustrated in Figure 3.4 only once, but it occurs twice in the right-hand individual in Figure 3.3. This suggests that in SCFG, common structure can be shared both within the individual, and of course across the population. It may be easier to understand this from another perspective. If we take the symbols in this grammar as nodes and the rules as edges, it is clear that this grammar is actually a graph. Thus, its components can be shared, while the individual generated is still guaranteed to be a valid tree.

This example clearly demonstrates that a common structure, which is repeated at different positions, can be represented by a grammar model. The common structure represented by a grammar can be easily shared among tree branches, either in one individual or across the population. Furthermore, this makes learning more efficient, because we do not need to re-learn the common structure in different positions. For example, in Figure 3.3, the sub-trees at A, B and C are identical (although at different locations) and therefore they all contribute to learning rules 2, 4 and 6. Hence, the total number of training cases needed is reduced, and this improves learning efficiency. Another issue worth mentioning, is that SCFG does not presume

S	→	Exp0				(0)
Exp0	→	Exp1	Op1	Exp1		(1)
Exp1	→	Exp2	Op2	Exp2	|	(2)
Exp1	→	x				(3)
Exp2	→	x				(4)
Op1	→	−				(5)
Op2	→	+				(6)

Figure 3.4: A trivial grammar model

a particular shape for building blocks, and it appears to be able to represent the various kinds of building blocks previously studied in GP research.

Although an SCFG grammar does not usually refer to position, and we argue that position independence is advantageous in EDA-GP, an SCFG grammar variant can learn position dependence by adding position constraints, such as depth, to limit the rule admission, as in SG-GP (Ratle and Sebag 2001) and PEEL (Shan, McKay, Abbass, and Essam 2003).

Lastly, tree individuals have no fixed complexity. Models with fixed structure, which resemble some models in EDA in having a fixed size in terms of number of variables, cannot reflect this variation. With an appropriate learning method, the grammar model may vary in size so as to reflect the superior individuals (training samples), while also generalising them to an appropriate extent.

Hence, we conclude that an SCFG model has the desirable characteristics for an EDA-GP model proposed in Section 3.2.

3.3.3 Grammar Model Sampling and Learning

In PRODIGY, the new population is generated by sampling an SCFG grammar model. The sampling method for SCFG has been discussed in Section 2.1.1.

Regarding the grammar model learning method, the PRODIGY framework does not have to be bundled with a specific SCFG model learning method. Any SCFG model learning method can be used. In this framework section, we will not describe any particular learning method. In later chapters, two of our methods based on PRODIGY will be discussed, each employing a different grammar learning method. However, when introducing a grammar learning method from NLP to PRODIGY, there are some considerations that one must be aware of, which we discuss in the following subsections. Some aspects of these considerations will also be revisited in Section 3.4.

3.3.3.1 Initial Grammar

PRODIGY generally relies on an initial grammar, $\mathcal{M}_0$, to specify the initial search space; in contrast, an initial grammar is rarely available in NLP. This initial grammar introduces typing and some level of background knowledge, as in GGGP, and it also imposes a constraint on the grammar learning. All later grammar models learnt must be consistent with the initial grammar. Otherwise, infeasible individuals may be generated.

3.3.3.2 Multiple Iterations of Grammar Learning

Grammar learning in NLP is one-shot learning, while PRODIGY involves multiple iterations of grammar learning. One implication is that, in conventional one-shot learning, if the goodness measure for the grammar - such as a metric derived from Minimum Message Length methods - is not very accurate, we may still be able to manually tune the parameters or incorporate fudge factors to largely cancel out the errors caused by the inaccurate measure in the context of this particular situation. However, in multiple iterated learning such as in EDA, it is generally difficult - if not impossible - to find a single set of parameter values which can offset the error at a suitable level for every iteration. The training examples and patterns behind

the samples may change from iteration to iteration. A set of parameters suitable to offset the errors in one iteration may not be suitable for other iterations, particularly when the structure of the learning set changes - as is the deliberate intention in EDA methods - from generation to generation. This issue will be revisited in Section 5.3.3.1.

3.3.4 Search in the Grammar Space

From the perspective of search, EDA methods search a different space from the original problem space, mapping the search from the original problem space to the model space. However different EDA methods have different forms of models and thus different mapping methods. Due to these differences, the connectivities and distance metrics of the model spaces may be entirely different, both from the original problem space, and from each other. This leads to a variation in performance of different EDA methods on different problems.

PRODIGY maps its problems to a grammar space. We anticipate that the search in the new grammar space may be beneficial for some problems. Firstly, by mapping the problem to a new grammar space, infeasible solutions are excluded from the search i.e. the search space shrinks. This has been repeatedly empirically demonstrated in GGGP, which uses the same underlying problem space. Secondly, the distance metric and the connectivity of the grammar space is different from other representations, and in particular, very different from that of methods based on PIPE's prototype tree. For example, two grammars which share some position-independent components will be closer to each other in PRODIGY-based distance metrics than if they did not share those components. There is no corresponding effect in prototype-tree methods such as PIPE.

From this perspective, EDA methods, including PRODIGY, are about finding an appropriate mapping to a new model space, so that the search in the new space is easier than in the original problem space.

3.3.5 Relationship with Other Works

PRODIGY provides a unified framework for grammar based EDA-GP, including SG-GP (Ratle and Sebag 2001), Bosman's work (Bosman and de Jong 2004) and methods to be presented in this thesis. It may also help to understand related work such as grammar-based systems using Ant Colony Optimisation (ACO) (see Section 2.6) and conventional GGGP with a grammar learning component (Tanev 2004; Whigham 1995a). Although these approaches are not strictly EDA-GP, they fit the PRODIGY perspective in using a stochastic grammar model, and in refining this model through learning from fit individuals.

Figure 3.5 gives a conceptual overview of some of the main EDA-GP approaches: PIPE (Salustowicz and Schmidhuber 1997), EDP (Yanai and Iba 2003), vector SG-GP (Ratle and Sebag 2001), ECGP (Sastry and Goldberg 2003), Bosman's work (Bosman and de Jong 2004), PEEL (Shan, McKay, Abbass, and Essam 2003), GMPE (Shan, McKay, Baxter, Abbass, Essam, and Nguyen. 2004). The last two, PEEL and GMPE, will be presented in detail in later chapters of this thesis.

Different approaches are organised according to the degree of constraint imposed on their handling of dependence. This constraint has two dimensions, the type of interactions among nodes (i.e. the complexity of the interaction relationships that can be represented), and position dependence (i.e. whether the dependence between nodes is positional). These two properties are summarised in brackets under the name of each method. The lower part of the figure lists approaches consistent with the PRODIGY framework, while the upper part lists systems based on the PIPE prototype tree.

Among all these approaches, the earliest - PIPE - is the most rigid. In PIPE, probabilities sit at particular positions of the prototype tree and thus cannot be moved around, and the probabilities in each node are independent. GMPE, at the other extreme, is the most flexible, in that its probability model does not have any positional reference, and the probabilities on the grammar productions can

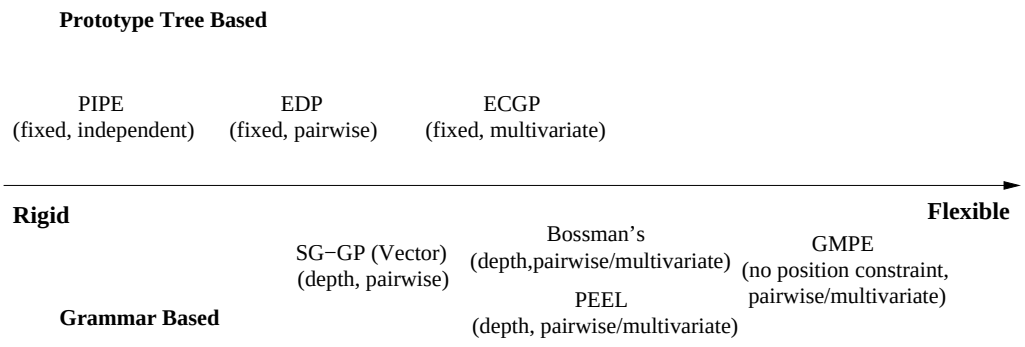


Figure 3.5: Relations among EDA-GP methods

represent pairwise or even multivariate interaction among nodes. Some methods in the middle of the spectrum, such as SG-GP, take only depth, instead of full position, as a reference.

3.4 Learning Issues

The PRODIGY framework proposed in this section is a general framework which can be implemented in various ways depending on the specific emphasis of each implementation. Some related issues, most of which are intertwined with the learning methods, are discussed in this section.

3.4.1 General-to-specific and Specific-to-general Learning Method

In each iteration, the learning may start from a general grammar model and then specialise it, or the other way around, i.e. start from a specific grammar and then generalise it. For example, the grammar in Figure 2.1 is a very general grammar for symbolic regression problems. It covers all the possible solutions for certain symbolic regression problems indiscriminately. If we start building models from this grammar, we would usually want to specialise the grammar so that promising solutions are favoured, i.e. that the probability of deriving these promising solutions increases (consequently the probability of deriving bad solutions decreases). Much

of the time, this will be done by updating the probabilities attached to the rules (originally a uniform distribution), but this process will generally reach a limit, beyond which further learning requires a new, more specialised, probability model (grammar). This new grammar would generally be obtained by replacing one or more general rules with a larger number of more specific rules. While this general-to-specific search can be effective, it is also worth considering the reverse, specific-to-general search. This starts with a very specialised grammar, for example a grammar which covers only the currently selected, fitter individuals (the training examples), and then generalises it to cover a larger area of search space. Again this is done either by changing the probability distribution, or by reducing the number of rules.

Note that the learning being discussed here is the learning in each iteration, not the learning across the entire run. When adopting either approach, we must ensure that the learnt grammar is consistent with the given initial grammar.

3.4.2 Incremental Learning

PRODIGY requires the estimation of an SCFG model in each iteration. This model can be either estimated from scratch based on the currently selected fitter individuals from the population, or learnt in an incremental way. Usually, it would be desirable to make use of previously learnt experience, i.e. to learn incrementally.

This distinction is closely related to one commonly made in EC, between generational and steady-state approaches, discussed in Section 2.2.1.3. In the generational EC approach, an entirely new population is created in each iteration. In contrast, for the steady-state EC approach, once an individual is created, it is incorporated back into the population straightaway. Therefore, there are no clearly identifiable generations.

Using the analogy of these two EC approaches, PRODIGY also has a generational approach (in which the model is learnt from scratch each iteration) and a steady-state approach (in which it is learnt incrementally). However, there is a very impor-

tant difference between the EC and PRODIGY approaches. In PRODIGY, it is the SCFG which conveys information across the run. Therefore, the emphasis is on the model rather than the population. More specifically, generational PRODIGY learns an entirely new model from scratch in each iteration, while steady-state PRODIGY progressively updates its model.

Generational PRODIGY is not an explicit incremental learning method, because at each iteration its model is built from scratch. By contrast, a steady-state PRODIGY builds its current model based on the one learnt in the previous iteration and thus is an incremental method. It is likely that incremental learning would in general be more effective (in terms of fitness evaluations required), but it also introduces more complex issues of probability update and model combination. While there are straightforward Bayesian methods for incrementally updating probabilities within a fixed model, methods for incrementally updating probabilities in the context of changing models are at or beyond the state of the art in statistical theory.

In our experience, generational PRODIGY is relatively straightforward to implement. At each iteration of generational PRODIGY, given all its training samples, a model is built, often using a hill-climbing method, and in this it is similar to a standard grammar learning problem in Natural Language Processing (NLP).

However, steady-state PRODIGY raises problems that do not usually occur in NLP. The most critical issue is how to update the model from the previous iteration. Besides needing to consider the specific learning technique, one must know how to weight evidence from the previous model against evidence from the current training example. An arbitrary learning rate parameter can be used to discount the importance of the previous model, but Bayesian induction would be a preferable alternative. Bayesian induction translates this problem into choosing an appropriate prior. However there is no easy method to make this choice.[2] Not only might

[2]In (Stolcke 1994), there are two implementations proposed for grammar model learning, online and batch versions. They both learn maximum a posteriori probability (MAP) grammar models with respect to some given training data set. However, both of these versions are only directly

different distributions be chosen for the prior, but even if a simple choice such as uniform prior is chosen, there are multiple uniform priors available, one for each grammar (probability model) generated throughout the learning process.

3.4.3 Additional Constraints

Additional constraints can be used in PRODIGY and EDA-GP to limit the rule admission or symbol applicability. For example, although there are no position references in the standard SCFG model, additional constraints, such as the position in the tree structure, may be introduced into SCFG for other reasons, such as efficiency or implementation convenience.

Position constraint is the most common constraint. It can be in the form of the depth of the tree as a vertical coordinate in some reference system, or a full position constraint including both vertical and horizontal coordinates. The PIPE prototype tree appears to be a common and natural reference system.

A number of approaches based on grammar models EDA-GP (Ratle and Sebag 2001; Shan, McKay, Abbass, and Essam 2003; Bosman and de Jong 2004) adopt a depth constraint. The approaches based on the PIPE probabilistic model known to us – PIPE (Salustowicz and Schmidhuber 1997), ECGP (Sastry and Goldberg 2003) and EDP (Yanai and Iba 2003) – can be regarded as using a full position constraint.

Additional position constraints may limit the flexibility of the probabilistic model of EDA-GP. However, it may significantly simplify the learning of the probabilistic model, and hence reduce the computational overhead.

---

applicable to generational PRODIGY. In the batch version, the whole training data set is available for learning. In the online version, although only part of the entire set of training examples is used each time, in order to reduce the computational cost, the entire training set is still available for computing the posterior probability. However, this is not the case in steady-state PRODIGY, i.e. the complete training set is generally unavailable.

3.4.4 Exploration versus Exploitation

The EDA algorithm in general strongly favours exploitation over exploration. To understand this, let us consider an EDA with no selection. Assuming the current model is $\mathcal{M}$, a sample set $\mathcal{G}'$ of infinite size can be drawn from $\mathcal{M}$. Assuming perfect learning methods, and in the limit of infinite population sizes, the new model $\mathcal{M}'$ learnt from $\mathcal{G}'$ should be equivalent to $\mathcal{M}$ from which $\mathcal{G}'$ was generated. In this case, the search is stagnant. If selection is introduced, which is normally the case in EDA, the search space covered by the selected set of individuals $\mathcal{G}$ will not be greater than $\mathcal{G}'$, $\mathcal{G} \subseteq \mathcal{G}'$, consequently the new model $\mathcal{M}'$ learnt from $\mathcal{G}$ covers less search space than the original model $\mathcal{M}$ which generated $\mathcal{G}$. Therefore, $\mathcal{M}'$ is more specialised than $\mathcal{M}$. This reasoning is based on two assumptions:

1. perfect learning method
2. infinite sample size.

Although these two assumptions do not always hold in reality, it does suggest that EDA algorithms, in general, strongly favour exploitation to exploration. In fact, finite sample size increases the bias toward exploitation. Given a sufficient number of training examples, EDA should be able to efficiently shrink the search space and to thus locate the optimal solution.

Therefore, to improve the probability of obtaining an optimal solution, the sample size (population size) has to be examined carefully. This issue of population sizing is well known in conventional EDA (Pelikan 2002). From the perspective of problem decomposition, if the problem only needs to be decomposed once, the time complexity of solving the problem is exponential to the size of the biggest sub-solution, regardless of other sub-solutions.

In conventional EDA, it is usually possible to only decompose the problem once. Although hierarchical problem decomposition is desirable, this is usually primarily an issue of efficiency. By contrast, in EDA-GP, it is rare that only one decomposition

is required, and thus it is often necessary to hierarchically decompose the problem. This is because in EDA-GP, we need to not only find the sub-solutions, but to also combine them to form bigger sub-solutions. More specifically, we hypothesise that EDA-GP has two iterative epochs.

1. discovering sub-solutions - decomposition
2. combining sub-solutions to obtain bigger sub-solutions at some higher level - aggregation.

In the first epoch, of discovering sub-solutions, the problem is decomposed and sub-solutions are identified. This is an exploitation- dominated stage. The next epoch explores the combination of the sub-solutions to obtain bigger sub-solutions. This stage is dominated by exploration. Then, based on the bigger sub-solutions, the next iteration starts. This is a hierarchical decomposition. Therefore, we envisage that it would be desirable to explicitly identify and manage these two epochs in the EDA-GP algorithm.

3.4.5 Sample Clustering

It is possible that the problem at hand may have a number of optimal solutions. Conventional GA usually converges to only one of these solutions. More importantly, it may also be the case, for problems having multiple solutions, that combining good solutions coming from different parts of the search space results in poor solutions.

To overcome this problem, clustering can be used to group the samples into sub-populations, and to then conduct conventional genetic search or EDA search on each of these subpopulations (Hocaoglu and Sanderson 1997; Pelikan and Goldberg 2000; Larrañaga and Lozano 2001).

This is even more important for tree representation based methods, such as EDA-GP and conventional GP. Except for some artificial problems, most problems have

more than one solution - if for no other reason than the prevalence of introns. Unfortunately, those multiple solutions may distract the search. To deal with this, similar clustering techniques can be used in EDA-GP. Another advantage of introducing clustering is to improve efficiency, as EDA is usually computationally expensive, and EDA-GP is even more so. Clustering may reduce the number of training samples while improving EDA-GP learning efficiency because each subpopulation has more consistent samples.

3.5 Two Algorithms: PEEL and GMPE

We present two algorithms under PRODIGY framework in the thesis, PEEL and GMPE, and an extension of GMPE – sGMPE. Although both of these two algorithms are based on the same framework, these two methods have different emphases and thus radically different grammar learning methods are used.

As mentioned before, learning a completely positional independent model of EDA-GP is computationally expensive, Program Evolution with Explicit Learning (PEEL) employs a semi-positional independent grammar model. In this grammar model of PEEL, depth is taken as an extra dimension, which simplifies the learning. Thus, PEEL is a trade-off between computational cost and effectiveness.

Grammar Model based Program Evolution (GMPE) is based on grammar learning methods from the field of Natural Language Processing. It tries to learn a standard SCFG model. The Minimum Message Length principle is used to direct the search for a good SCFG model. It appears that most types of building blocks previously considered in GP research can be explicitly represented and preserved in GMPE.

The characteristics of these two implementations are summarised as follows.

1. Grammar learning approach. Grammar learning is general-to-specific search in PEEL. It starts with a general grammar and rules are added as the search progresses. The added rules make the grammar more specific. In GMPE, in

each iteration of grammar learning, there is a specific-to-general search. It starts with a grammar which only covers the training samples, and a merge operator is imposed on the grammar to generalise it. Note that in GMPE, in general, the grammars learnt at later stages are more specific than grammars learnt earlier but, in each iteration of grammar learning, it follows a specific-to-general approach.

2. Incremental learning. In the current implementation, PEEL uses an incremental learning approach while GMPE learns its grammars from scratch (although in both cases, these were pragmatic choices for simple implementation; either choice could be reversed).

3. Depth constraint. In PEEL, depth is introduced as an extra constraint, in common with a number of other methods of EDA-GP (Bosman and Thierens 1999; Ratle and Sebag 2001). It simplifies the learning, but imposes a disadvantage in that building blocks cannot be shared among positions of different depth due to this constraint. In GMPE, a grammar learning method from the field of Natural Language Processing is used. Based on the Minimum Message Length principle, a standard SCFG grammar model is learnt . Thus no depth constraint is considered. It is more flexible in the sense that there is no limitation on the position of the building blocks.

3.6 Conclusion

Estimation of distribution of tree form solutions, EDA-GP for short, is an interesting research direction not only because it has potential to improve GP performance, but also because it provides a new perspective to understand GP. It is vital to find a proper model for this kind of research. Based on current GP research, we have identified some desirable properties in a model for this purpose.

A framework for grammar-based EDA-GP, known as PRODIGY, has been pro-

posed. The core of this framework is a Stochastic Context-free Grammar model (SCFG). We have argued that SCFG, a probabilistic model with a flexible structure constraint, exemplifies the properties we identified as desirable for EDA-GP. Some important implementation considerations for PRODIGY methods were discussed. Two algorithms under the PRODIGY framework were briefly outlined, and will be discussed in detail in the later chapters.

Chapter 4

Program Evolution with Explicit Learning

We present in this chapter one of our implementations of PRODIGY, which represents a significant extension to our earlier method – Program Evolution with Explicit Learning (PEEL) (Shan, McKay, Abbass, and Essam 2003). We still refer to this significantly extended version as PEEL. In PEEL, we choose a specialised SCFG model in this implementation. The implementation of PEEL is a trade-off between the efficiency and effectiveness.

4.1 Algorithm

The basic algorithm is sketched in Figure 4.1. Obviously, it is consistent with the algorithm of PRODIGY. It starts with a population $\mathcal{P}$ randomly generated by sampling the given initial SCFG model $\mathcal{M}$. A Stochastic Context-free Grammar model (SCFG) $\mathcal{M}$ is learnt from the individuals $\mathcal{G}$ selected from this population. A new set of individuals $\mathcal{G}'$ is generated by sampling the learnt model $\mathcal{M}$. The new population is formed by replacing the original population $\mathcal{P}$ with $\mathcal{G}'$. The next iteration starts again from this new population.

1. Generate a population $\mathcal{P}$ from a given minimum SCFG model $\mathcal{M}$.
2. Select a set of fitter individuals $\mathcal{G}$.
3. Update the SCFG model $\mathcal{M}$ over $\mathcal{G}$.
4. Sample the SCFG model $\mathcal{M}$ to obtain a set of new individuals $\mathcal{G}'$.
5. Replace population $\mathcal{P}$ with $\mathcal{G}'$.
6. If the termination condition is not satisfied, go to 2.

Figure 4.1: Basic algorithm of PEEL

In the experiments in this chapter, we set the size of the selected individual set $\mathcal{G}$ to 2. Hence, $\mathcal{G}$ has two individuals, the best individual in the current generation and the best individual found so far, also called the elite. In other words, we use these two individuals to incrementally learn the grammar in each generation.

4.1.1 SCFG Model in PEEL

Our grammar model is a specialised stochastic parametric Lindenmayer system (L-system) (Prusinkiewicz and Lindenmayer 1990). The reason we use this specialised SCFG is that we want to balance between expressiveness of the grammar model and learning efficiency. This L-system is equivalent to a standard SCFG with two extra conditions on the LHS.

The two conditions we introduce are *depth* and *location*. So the modified form of a production rule is

$$X(d, l) \rightarrow \lambda \quad (p)$$

The first parameter, depth d is an integer, indicating that when generating individuals, this rule may only be applied at level d. The second parameter is a location l, which indicates the relative location where this rule may be applied. (Note that this relative location information is relative to the current location where the rule may be applied. More details will be presented in Section 4.1.2.2). p is the attached probability. From now on, matching the LHS does not mean merely matching the

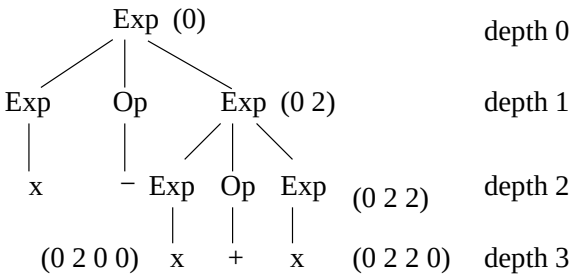


Figure 4.2: Relative coordinates of the nodes in PEEL probabilistic model. A node's position is described by a tuple of n coordinates $(b_1, b_2, \cdots, b_n)$, where n is the depth of the node in the tree, and b_i indicates which branch to choose at level i.

LHS symbol X, but also matching these two conditions on the left hand side.

The location information is encoded as the relative coordinate in the tree, somewhat similar to the approach of (D'haeseleer 1994). The position of each node in a rooted tree can be uniquely identified by specifying the path from the root to this specific node. A node's position can therefore be described by a tuple of n coordinates $T = (b_1, b_2, \cdots, b_n)$, where n is the depth of the node in the tree, and b_i indicates which branch to choose at level i.

For example, in Figure 4.2 to reach the bottom right-most node, from the root (with coordinate 0) we need to choose the 2nd, 2nd and 0-th branch, subsequently. This gives us 0220. Similarly, we can obtain the location of the left x at depth 3 in the left tree as 0200. Note: in this paper the depth of the root is 0, the branch in each sub-tree which connects to the left-most child node is the 0th branch and thus the left-most child is the 0th child.

For any given problem, we assume there is an initial grammar specifying the search space, as in Grammar Guided Genetic Programming (GGGP). The starting grammar, called the primitive grammar, of our implementation is very similar to this grammar, except that this set of rules is duplicated at each depth level. For example, the grammar rules in Figure 4.3 are changed into the rules in Figure 4.4 to generate individuals at depth 1:

S	→	Exp				(0)
Exp	→	Exp	Op	Exp	|	(1)
	→	Pre	Exp	|		(2)
	→	x				(3)
Op	→	+	|			(4)
	→	-	|		(5)
	→	*	|		(6)
	→	/				(7)
Pre	→	sin	|			(8)
	→	cos	|			(9)
	→	e^	|			(10)
	→	ln				(11)

Figure 4.3: A commonly used Context-free grammar for symbolic regression problem

Exp(d=1,l=#)	→	Exp	Op	Exp	$(p = 0.33)$
Exp(d=1,l=#)	→	Pre	Exp		$(p = 0.33)$
Exp(d=1,l=#)	→	x			$(p = 0.33)$
Op (d=1,l=#)	→	+			$(p = 0.25)$
Op (d=1,l=#)	→	-			$(p = 0.25)$
Op (d=1,l=#)	→	*			$(p = 0.25)$
Op (d=1,l=#)	→	/			$(p = 0.25)$
Pre (d=1,l=#)	→	sin			$(p = 0.25)$
Pre (d=1,l=#)	→	cos			$(p = 0.25)$
Pre (d=1,l=#)	→	e^			$(p = 0.25)$
Pre (d=1,l=#)	→	ln			$(p = 0.25)$

Figure 4.4: Example of fragment of initial PEEL grammar model for generating individuals at depth 1

The rules at other depth level can be defined in a similar way. The first condition, $d = 1$, indicates these rules can only be used to generate the 1st level ($d = 1$) of the tree. The probabilities are initialised uniformly, and are then subject to update by the probability learning method. All locations l are initialised to the value #, "don't care" or "match to any value". When necessary, the grammar will be refined by the structure learning method, and location information will be incrementally added. The grammar model learning, which includes learning both probabilities and structures, will be discussed in the following section.

The last issue worth mentioning is the treatment of constants in PEEL. As the probabilistic model in PEEL is a variant of SCFG, constants can be treated in the same way as in GGGP. To enable constants, we just need to add rules with constants in their RHS's. For example, if we want to add constant 1 to the grammar in Figure 4.4, the following rule would need to be added:

$$\text{Exp} \quad \rightarrow \quad 1$$

In all our experiments of PEEL, we decided not to include rules with constants to keep the grammar as compact as possible.

4.1.2 Model Learning

The SCFG grammar model has two components: structure and parameters (probabilities attached to rules). Learning of these two components will be presented in the following two sections.

4.1.2.1 Learning Probability

The basic idea of probability learning is to increase the probabilities of production rules which are used to generate good individuals.

We want to increase probabilities according to individual id. Assume the rules involved in deriving individual id are R^{id}. To update the probability of each rule $p(r_i)$, $r_i \in R^{id}$:

$$inc(r_i) = inc(r_i) + (1 - p(r_i)) \quad (4.1)$$

$$p(r_i) = p(r_i) + (1 - p(r_i)) \times lr \quad (4.2)$$

where lr is the predefined learning rate. $p(r_i)$ is then renormalised among rules with identical LHS (including both the LHS symbols and the two conditions). $inc(r_i)$ is used to accumulate the increments and does not need to be normalised. It will be used later in structure learning.

This grammar model has the potential to recover from over-specialising when the probabilities attached to the rules are not zero. Practically, it is possible that the probabilities of some rules approach zero, with the resultant effect that they may never be selected. To overcome this and encourage exploration, a lower and upper bound of the probabilities can be set as in (Stutzle and Hoos 1997). As the search progresses however, better and, very likely, bigger individuals may be discovered. To alleviate the bias toward individuals of a specific size limit, as implicitly set by the bounds of probabilities, it may be better to employ a proper annealing mechanism to decide these bounds rather than setting fixed values.

4.1.2.2 Learning Structure

The starting grammar (also called primitive grammar) will usually be minimal, representing the search space at a very coarse level. Then, when necessary, rules are added to focus the search to a finer level. The way we add new rules is by splitting existing rules, adding more context information. Once more context information has been added, one production can be split into several.

For example, assume we have the grammar illustrated in Figure 4.3. One of its initial rules in the SSDT representation is:

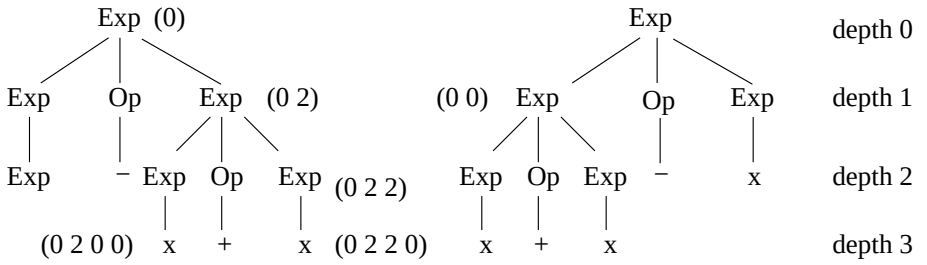


Figure 4.5: Generation of stings x-(x+x) and (x+x)-x from the grammar in Figure 4.3

$$\text{Exp(d=1,l=\#)} \quad \rightarrow \quad \text{Exp} \quad \text{Op} \quad \text{Exp} \quad (p = 0.33)$$

This rule would be applied to rewrite predecessor *Exp* at depth 1. $l = \#$ implies that no location information will be considered. Therefore, at depth 1, as long as predecessor *Exp* is matched, the rule would be applied with probability $p = 0.33$ no matter where the predecessor *Exp* is. As a result, this rule cannot distinguish between the derivations for $x - (x + x)$ and $(x + x) - x$ given in Figure 4.5 as these two trees must have the same probability of being generated for any probability distribution over our original grammar.

Thus, it is necessary for the system to be able to increase the expressiveness of the grammar through refinement. For example, if we split the above rule into three:

$$\begin{aligned}
\text{Exp(d=1,l=0)} &\rightarrow \text{Exp} \quad \text{Op} \quad \text{Exp} \quad (p_0) \\
\text{Exp(d=1,l=1)} &\rightarrow \text{Exp} \quad \text{Op} \quad \text{Exp} \quad (p_1) \\
\text{Exp(d=1,l=2)} &\rightarrow \text{Exp} \quad \text{Op} \quad \text{Exp} \quad (p_2)
\end{aligned}$$

then the two trees in Figure 4.5 may have different probabilities, because at depth 1 ($d = 1$) the *Exp* at location 2 will be rewritten as *Exp Op Exp* with probability p_2 while at location 0, it will be rewritten with probability p_0.

Note that in PEEL, Left Hand Side (LHS) matching is partial matching. The relative location l is matched right to left, with the rightmost symbol matching the current location, the next matching its parent, etc., until the relative location is exhausted.

For example in the left tree in Figure 4.5, the complete location of the right-most *Exp* at depth 1 is (0 2). The rule:

$$\text{Exp(d=1,l=2)} \quad \rightarrow \quad \text{Exp} \quad \text{Op} \quad \text{Exp} \quad (p)$$

can match even if its predecessor location is $l = 2$. In these circumstances, we start matching from the right of the complete location string since it is the closest (or most relevant) information. Therefore, in this example, for the complete location (0 2), both $l = 2$ and $l = 02$ can match, but not $l = 20$.

Therefore, the key idea here is to split the production rule by using more contextual information; it is partially motivated by (Bockhorst and Craven 2001). Two steps are needed in this splitting:

1. detect the *most loaded Left Hand Side (LHS)*,
2. split the most loaded LHS.

The most loaded LHSs are those which are both highly influential, and badly converged. Specifically, we calculate the *load* for each LHS in the grammar. The LHS with the largest load is called the most loaded LHS. The load of a LHS L is defined as:

$$load(L) = \sum_{r_i \ with \ LHS \ L} inc(r_i)$$

where $inc(r_i)$ is defined in Equation 4.1. The idea behind this measure is: if one LHS L keeps being rewritten using different RHS's, it suggests that there is a great degree of uncertainty regarding which RHS should be chosen given LHS L. The following small example may help to understand this. Suppose we have the following set of rules.

$$\begin{array}{llllll}
\text{Exp(d=1,l=\#)} & \rightarrow & \text{Exp} \quad \text{Op} \quad \text{Exp} & (p_1 = 0.33) & (1) \\
\text{Exp(d=1,l=\#)} & \rightarrow & \text{Pre} \quad \text{Op} & (p_2 = 0.33) & (2) \\
\text{Exp(d=1,l=\#)} & \rightarrow & \text{x} & (p_3 = 0.33) & (3)
\end{array}$$

When structure learning happens, let's study two scenarios. The first scenario is that in good individuals the first rule is always chosen. The second scenario is that each of the first and last rules are chosen half of the time. Obviously, in these two scenarios, it is more certain in the first scenario than the second one in choosing the RHS for LHS $Exp(d = 1, l = \#)$. We can substantiate this example with more detail. For example, in the first scenario, the first rule could be used twice. If the learning rate = 0.4, the load would be calculated as follows:

$$\begin{aligned} load(Exp(d = 1, l = \#)) &= inc(r_1^1) + inc(r_2^1) + inc(r_3^1) \\ &\quad + inc(r_1^2) + inc(r_2^2) + inc(r_3^2) \\ &= 0.400 + 0 + 0 + 0.526 + 0 + 0 \\ &= 0.93 \end{aligned} \tag{4.3}$$

where r_i^j is the increment of the rule i at time step j.

In the second scenario, each of the first and last rule could be used once each sequentially. The load is then calculated as follows:

$$\begin{aligned} load(Exp(d = 1, l = \#)) &= inc(r_1^1) + inc(r_2^1) + inc(r_3^1) \\ &\quad + inc(r_1^2) + inc(r_2^2) + inc(r_3^2) \\ &= 0.400 + 0 + 0 + 0 + 0 + 0.737 \\ &= 1.14 \end{aligned} \tag{4.4}$$

Thus, the load of LHS $Exp(d = 1, l = \#)$ is greater in the second scenario than in the first one, which means that there is more uncertainty in the second scenario. Note that it is obvious that this definition of load is, to some extent, related to the entropy of the LHS. However, the reason that entropy was not chosen is that it cannot represent the dynamics of the search. Entropy can measure which LHS is badly converged, but cannot determine which rule is highly influential at a particular moment of a search. For example, at one point in time, one LHS could be badly

converged, i.e. all of its RHS's have the same probabilities, but this LHS might be very rarely used in constructing good individuals. In this case, this LHS should not be chosen for splitting if there are much more frequently used LHS's with the same convergence. The measure of entropy cannot distinguish these different kinds of LHS's. Therefore, in the PEEL algorithm, load is a better measure than entropy.

To reduce above-mentioned uncertainty, we need to add more contextual information, which is represented by the relative location. By adding more context information, rules become distinguishable. Once the context information has been added, one production can be split into several. For example, the rule

$$\text{Exp(d=1,l=\#)} \quad \rightarrow \quad \text{Exp} \quad \text{Op} \quad \text{Exp} \quad (p = 0.33)$$

can be split into three rules by adding more relative location information according to the grammar in Figure 4.3:

$$\begin{aligned}
\text{Exp(d=1,l=0)} &\rightarrow \text{Exp} \quad \text{Op} \quad \text{Exp} \quad (p = 0.33) \\
\text{Exp(d=1,l=1)} &\rightarrow \text{Exp} \quad \text{Op} \quad \text{Exp} \quad (p = 0.33) \\
\text{Exp(d=1,l=2)} &\rightarrow \text{Exp} \quad \text{Op} \quad \text{Exp} \quad (p = 0.33)
\end{aligned}$$

Initially, the new rules each have the same probability as the original. However, after further learning, i.e., probability learning, these three rules may have different probabilities.

More specifically, in this example, originally the rule

$$\text{Exp(d=1,l=\#)} \quad \rightarrow \quad \text{Exp} \quad \text{Op} \quad \text{Exp} \quad (p = 0.33)$$

will be applied with equal probability anywhere in depth 1 as long as the LHS symbol "Exp" is matched, no matter what precedes it. In other words, it cannot recognise its surrounding context. By adding relative context, this one rule becomes three. Thus, at different relative locations, it may have different probabilities. This

means that it is now possible to recognise the context to a certain extent. Note that the location information is added incrementally – each time, only the minimum location information is added. The location information describes the location of the current node relative to its ancestor node. Thus the location information will be accumulated upward, toward the ancestor of all nodes – the root. In the above example, with added minimum relative location, this rule now depends on its immediate parent. If this is not enough, in the next iteration of learning, more relative location information may be added so that this rule can recognise not only its immediate parent but also its more distant ancestors. In the current implementation, this structure learning happens at predefined intervals, i.e. after predefined number of generations, the load is calculated and the rules are split accordingly.

As can be seen, our implementation is an incremental learning approach. It starts from a minimum initial L-system, a special kind of SCFG in this context, which describes the search space very roughly. Then, when necessary, rules are added to the L-system to focus the search to a finer level.

4.1.3 Model Mutation

A primary exploration mechanism is mutation. We want to explore the search space surrounding good individuals. Therefore, we consider individuals in defining grammar mutation. Assume we want to do mutation with respect to individual id. R^{id} are all rules involved in deriving id. Q^{id} are all rules which have an identical LHS with any of the rules in R^{id}. We mutate all the rules in $R^{id} \cup Q^{id}$. The mutation probability is p_m. n_r is the total number rules of $R^{id} \cup Q^{id}$. There is a risk that the number of rules mutated may increase too fast as the size of individuals increases, so we make p_m depend inversely on the n_r:

$$p_m = c_m / n_r$$

where c_m is a predefined *mutation rate*.

The purpose of mutation is to allow the search to escape from the areas of previously

learnt knowledge, and thus explore different parts of the search area. Consequently, our mutation will force converged LHSs to partially lose their convergence, thus encouraging exploration. If rule r_i is mutated, its probability $p(r_i)$ is changed as follows.:

$$p(r_i) = p(r_i) + mi \times (1 - p(r_i))$$

where $p(r_i)$ is the probability of rule r_i which is being mutated, and mi is a predefined *mutation intensity*. $p(r_i)$ is renormalised after mutation.

4.1.4 Model Sampling

Grammar model sampling, i.e. generating individuals from the grammar, is basically consistent with SCFG grammar sampling, – keep rewriting nonterminals according to the production rules until no nonterminal is left. This has been illustrated in the previous chapter. There is only one minor difference in matching the LHS. In this implementation, matching the LHS includes matching both the LHS symbol and the two conditions on the left hand side, depth and location.

4.2 Experiment

In this section, to demonstrate PEEL's relevance to general problem solving, we present two symbolic regression problems and one time series problem. The first problem has some degree of regularity, which may be useful for algorithms which can handle build blocks, such as PEEL. The second one was used by a very related method and thus serves as a good benchmark for comparison. The last problem is a real world problem with a noisy data set. This is useful for testing the generalisation of the algorithm. Furthermore, these problems have similar initial grammars and thus have reduced the implementation effort. Due to the variety of the test problems, we expect the results may generalise to other problems.

Population size	100
Generation	50
Maximum depth	10
Crossover rate	0.9
Mutation rate	0.1
Tournament size	3
Total number of runs	50

Table 4.1: Parameter settings for GGGP on simple symbolic regression problem $f(x) = x^4 + x^3 + x^2 + x$

4.2.1 Simple Symbolic Regression Problem

Symbolic regression problems are often used in the GP literature for evaluating the performance of an algorithm. In this experiment, we adopt a widely used function, originating from (Koza 1992). The function is:

$$f(x) = x^4 + x^3 + x^2 + x$$

The fitness cases were sampled at 20 equidistant points in the interval [-1,1). Fitness was the sum of absolute value of error. 5,000 program evaluations were set for both PEEL and GGGP. The grammar for this problem can be found in Figure 4.7, which is the reduced grammar of Figure 4.3 with unary operators removed. The major settings for GGGP and PEEL can be found in Table 4.1 and 4.2.

Fifty runs were generated for both systems. We measured the number of successful runs. A successful run is a run which found a perfect solution. The cumulative number of successful runs were plotted in Figure 4.6. The x-axis is the number of individual evaluations and the y-axis is the cumulative number of successful runs. As can been seen, 38 out of 50 PEEL runs found the perfect solution while 32 GGGP runs succeeded. Briefly, on this problem, the performance of our implementation of PEEL is at least comparable to GGGP.

To understand the difficulty of this problem, we also conducted random search.

Population size	10
Generation	500
Maximum depth	10
Mutation rate C_m	0.4
Mutation intensity mi	0.2
Learning rate lr	0.4
Number of elitists	1
Learning interval	10 generations
Total number of runs	50

Table 4.2: Parameter settings for PEEL on simple symbolic regression problem $f(x) = x^4 + x^3 + x^2 + x$

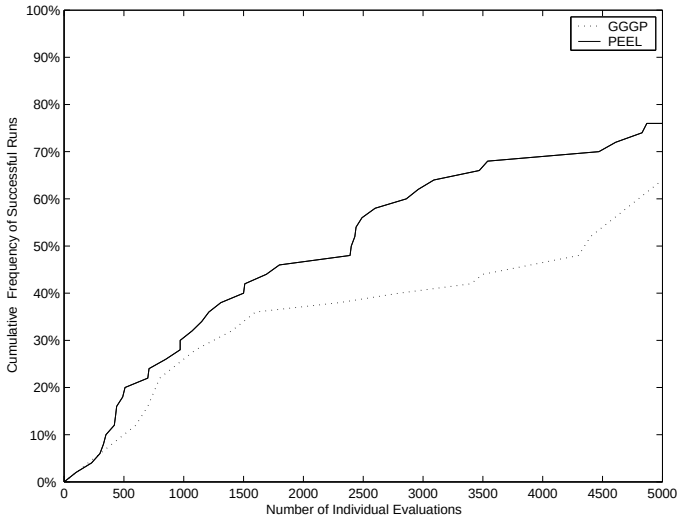


Figure 4.6: Comparison of PEEL and GGGP on cumulative frequency of successful runs on the problem $f(x) = x^4 + x^3 + x^2 + x$.

S	→	Exp				(0)
Exp	→	Exp	Op	Exp	|	(1)
	→	x				(3)
Op	→	+	|			(4)
	→	−	|		(5)
	→	×	|		(6)
	→	/				(7)

Figure 4.7: Context-free grammar for the simple symbolic regression problem $f(x) = x^4 + x^3 + x^2 + x$

This was done by turning off the grammar learning module of our system. In total, 250,000 individuals (equivalent to 50 runs × 5000 individuals per run) were generated and evaluated. No solution was found.

4.2.2 Second Regression Problem

This is a more complicated symbolic regression problem. In this experiment, the function to be approximated is adopted from (Salustowicz and Schmidhuber 1997). This complicated function was chosen to prevent the algorithm from simply guessing it. The function is:

$$f(x) = x^3 \times e^{-x} \times cos(x) \times sin(x) \times ((sin(x))^2 \times cos(x) - 1)$$

The following results compare PEEL and Grammar Guided Genetic Programming (GGGP). The results of PIPE in (Salustowicz and Schmidhuber 1997) over the same problem are also briefly mentioned.

The fitness cases were sampled at 101 equidistant points in the interval [0,10], 100,000 program evaluations were set for both PEEL and GGGP, the same as in (Salustowicz and Schmidhuber 1997). The major settings for GGGP and PEEL can be found in Table 4.3 and 4.4. The errors (sum of absolute value of error) of the

Population size	2,000
Generation	50
Maximum depth	15
Crossover rate	0.9
Mutation rate	0.1
Tournament size	3
Total number of runs	30

Table 4.3: Parameter settings for GGGP on complex symbolic regression problem $f(x) = x^3 \times e^{-x} \times cos(x) \times sin(x) \times ((sin(x))^2 \times cos(x) - 1)$

best individual in each run are summarised in Table 4.5. The grammar for both systems can be found in Figure 4.3.

As can be seen from Table 4.5, when considering the mean and median statistics, PEEL gets much better results than GGGP, although both the standard deviation and range of results for PEEL is larger than for GGGP. We conducted a two-tailed student *t-test* to see whether the difference of means between these two systems is statistically significant. We found that at $p < 0.01$, the difference is statistically significant.

We also presented the size in term of the number of terminal nodes (i.e., GP symbols - nonterminal grammar symbols are not counted) in Table 4.5. The difference of the sizes is not statistically significant. It is interesting to see that PEEL found better solutions than GGGP did, but the sizes of these solutions were similar.

Comparing PEEL with the results of PIPE and GP in (Salustowicz and Schmidhuber 1997), we find that PEEL is no worse than GP and PIPE in terms of mean of fitness of the best individuals, even though we use less tree depth and have no ephemeral constants in PEEL. Overall, in terms of the general accuracy, on this specific problem, PEEL has roughly similar performance to PIPE.

The best solution found by PEEL is presented in Figure 4.8. It has 77 nodes with

Population size	10
Generation	10,000
Maximum depth	15
Mutation rate C_m	0.4
Mutation intensity mi	0.2
Learning rate lr	0.4
Number of elitists	1
Learning interval	10 generations
Total number of runs	30

Table 4.4: Parameter settings for PEEL on complex symbolic regression problem $f(x) = x^3 \times e^{-x} \times cos(x) \times sin(x) \times ((sin(x))^2 \times cos(x) - 1)$

Algorithm	mean±std dev	max	median	min	size
PEEL	5.44±4.55	20.32	3.78	0.89	75.68±24.14
GGGP	7.87±3.54	14.00	7.56	0.95	74.07±32.94

Table 4.5: Comparison of PEEL and GGGP on the symbolic regression problem $f(x) = x^3 \times e^{-x} \times cos(x) \times sin(x) \times ((sin(x))^2 \times cos(x) - 1)$

(((e^(((e^(ln (x))) - (x)) - (ln ((sin ((cos (x)) + (cos ((cos ((e^(e^(x)))) / (x))) + (ln ((e^((x) / (x))) * (sin ((x) + (x))))))))) + (e^(x)))))) / ((sin (ln (e^(x)))) * (((x) - (x)) - (x)))) * ((x) * ((sin (((cos ((cos (x)) * (sin ((cos (x)) + (x))))) + (cos (sin ((sin (cos (x))) + (x))))) - (e^(ln (x)))))) * (((x) - (x)) + (ln (e^(x)))))))

Figure 4.8: Best solution found by PEEL on regression problem $f(x) = x^3 \times e^{-x} \times cos(x) \times sin(x) \times ((sin(x))^2 \times cos(x) - 1)$

fitness 0.893954.

4.2.3 Time Series Prediction

For this problem, we used the sunspot benchmark series for years 1700-1979 (The data can be obtained from ftp://ftp.santafe.edu/pub/Time-Series/data/). It contains 280 data points, divided into three subsets: the data points from years 1700-1920 are used for training, while 1921-1955 and 1956-1979 are for testing.

The task of prediction is to map data points from lag space to an estimate of the future value:

$$\hat{x}_{t+1} = f(x_t, x_{t-1}, \ldots, x_{t-\delta})$$

where δ is the order of the model, which is set to 5 in this work, and t is the time index.

The accuracy of the model is measured by Average Relative Variance (ARV) (Weigend, Huberman, and Rumelhart 1992). Given data set S, ARV is defined as:

$$\begin{aligned} ARV(S) &= \frac{\sum_{t \in S} (target_t - prediction_t)^2}{\sum_{t \in S} (target_t - mean)^2} \\ &= \frac{1}{\sigma^2} \frac{1}{N} \sum_{t \in S} (x_t - \hat{x}_t)^2 \end{aligned}$$

where $target_t$ or x_t are the actual values, $prediction_t$ or $\hat{x}_t$ are the predicted values, N is the total number of data points, and σ^2 is the variance of the total dataset (i.e. from year 1700-1979). It is set to 0.041056, as in (Jäske 1996).

Population size	200
Generation	3000
Maximum depth	N/A
Crossover rate	0.2
Mutation rate	0.3
Reproduction rate	0.01
Total number of runs	10

Table 4.6: Parameter settings for GP on sunspot prediction problem

Population size	10
Generation	60,000
Maximum depth	15
Mutation rate C_m	0.4
Mutation intensity mi	0.2
Learning rate lr	0.4
Number of elitists	1
Learning interval	50 generations
Total number of runs	27

Table 4.7: Parameter settings for PEEL on sunspot prediction problem

Twenty-seven PEEL runs were generated using the same settings as in the function regression experiment, except that the number of generations was set to 600,000 to make it comparable with the GP runs in (Jäske 1996), and the grammar structure learning interval was increased from 10 to 50 to reduce computational time. The major settings are listed in Table 4.6 and 4.7. The results are summarised in Table 4.8.

The GP results, which are the statistics of ten runs, are adopted from (Jäske 1996), which is a widely cited comparison work for GP on the sunspot data. The first row is the statistics of all PEEL runs. The second and third rows are the top 80% of PEEL runs and GP runs respectively, measured by training accuracy. The fourth

Algorithm	Training ARV 1700-1920	Generalisation ARV 1921-1955	Generalisation ARV 1956-1979	size
PEEL(all)	0.122±0.033	0.196±0.098	0.324±0.187	44.3±13.7
PEEL(top 80%)	0.108±0.010	0.163±0.037	0.307±0.197	47.5±12.5
GP(top 80%)	0.155±0.045	0.235±0.090	0.43±0.10	52.5
t-test	$p < 0.01$	$p < 0.05$	$p < 0.05$	
PEEL(top 60%)	0.104±0.007	0.153±0.034	0.259±0.123	47.7±10.0
GP(top 60%)	0.125±0.006	0.182±0.037	0.37±0.06	66.5
t-test	$p < 0.001$	$p < 0.15$	$p < 0.01$	

Table 4.8: The comparison of PEEL and GP on sunspot prediction problem

(sin ((sin (e^(((e^(ln ((x2) + (x5)))) / (sin ((((x2) * ((((x1) * ((x1) * (x1))) + (x1)) + (x6))) + (cos (ln (x3)))) + (((x2) / ((x3) + (x3))) * ((ln (sin (cos (x6)))) + ((x2) + (x2))))))) * (x5)))) / (((sin ((x6) * (cos (((x6) + (x1)) + ((x6) + (x4))))))) - (e^(sin (sin (x3))))) / (x1))))

Figure 4.9: Best solution found by PEEL on sunspot prediction problem

row is a two-tailed student *t-test* of the above two rows. The fifth and sixth rows are accuracies of the top 60% of PEEL runs and GP runs respectively. The fourth row is two-tailed student *t-test*. The right-most column, size, is the number of nodes in the best individuals. The reason we report the top 80% and 60% runs is that this is the only comparison data reported in (Jäske 1996).

The best individual among all PEEL runs, in terms of ARV on the training data set, can be found in Figure 4.9. This best individual has 61 nodes and attains an error rate of 0.099 on the training dataset, 0.092 on the first testing dataset and 0.143 on the second testing dataset.

As can be seen from Table 4.8, PEEL's mean error rates on both the training

and the two testing datasets are all smaller than their GP counterparts. Most of the differences are statistically significant, as suggested by the p-value from our *t-test*. Therefore, on this problem, PEEL, one of our implementations of PRODIGY outperforms conventional GP.

We notice that PEEL generalises no worse than the GP system. Note that the GP in (Jäske 1996) was carefully designed to alleviate overfitting, while neither overfitting control nor explicit parsimony pressure were used in PEEL.

Much research has been done on parsimony pressure and control of individual complexity in GP (Bleuler, Brack, Thiele, and Zitzler 2001; Rosca 1997b; Iba, de Garis, and Sato 1994; Zhang and Mühlenbein 1995). It is notable that, in our work, although no complicated parsimony mechanism is used, the results still maintained low complexity, yet the concomitant problem in GP, i.e., premature convergence, was not observed as severe as in GP because better generalisation was obtained.

This leads to the question, where does the parsimony pressure in PEEL come from? We hypothesise that PEEL, as well as PRODIGY in general, has a built-in Occam's Razor because of its probabilistic nature. Given a particular probability distribution on the production rules, it is harder to generate specific larger structures (i.e., their overall probabilities are lower) than it is to generate smaller. Thus learning a larger structure requires greater convergence of the probability distribution, requiring more learning time. This explains why the PEEL results tend to be compact.

We note in passing that more accurate models for predicting sunspots have been obtained by GP (Nikolaev and Iba 2001). However Nikolaev's method uses highly specialised heuristics for non-linear function fitting, and hence have limited value for comparison of generalised GP methods.

4.3 Empirical Analysis

As discussed in the previous chapter, there are five important properties that an appropriate model should have for EDA-GP. We argue that an SCFG model has all these five properties. In this chapter, we empirically confirm the importance of these properties and investigate their relationship to the PRODIGY framework using implementation of PEEL.

Of the five properties, internal hierarchical structure and non-fixed complexity are generally accepted as important for EDA-GP; since SCFG representation is able to handle them, we will not pursue them further here. Locality of dependence is difficult to measure directly. However, it is closely related to position independence and modularity. Therefore, the purpose of this section is to examine the impact of position independence and modularity on the behaviour of PRODIGY. We intend to show how these two properties can be controlled and measured in our implementation of PRODIGY, and measure the correlation between them and search performance.

4.3.1 Methodology

In PEEL's implementation of PRODIGY, position independence and modularity are embodied in the use of rules. More specifically, position dependence and modularity are respectively embodied in the amount of context and the level of reuse of rules, respectively.

We firstly examine how position dependence is related to the amount of context. Recall that, in the PEEL implementation of PRODIGY, the context is represented as a unique path from the root to the specific node. If this path is fully defined, the rule can only be used in the position defined by this path. There is no other place where this rule can be applied. If the path is only partially defined, the rule can be shared among tree branches. In the PEEL implementation, the context is added

incrementally. So the more context is added, the more restricted are the locations where the rule can be applied, and thus the less the position independence. The amount of context can be measure by the length of context.

The modularity is reflected by the rule reuse. Recall that modularity emphasises that some sub-tree structures are used multiple times in constructing one individual. In PEEL, this is translated into rule reuse, because individuals are generated by repeatedly applying rules to nonterminals. If every rule is only used once in constructing one individual, this individual does not have any component which is shared within itself. The number of times that a rule is used can be directly measured.

In PEEL, these two factors, position independence and modularity, are controlled by the grammar structure learning, or more precisely, by the frequency of grammar structure learning. As we saw, in this implementation in Section 4.1.2.2, the grammar structure learning happens at predefined intervals - that is, grammar structure learning happens a fixed number of generations apart. The effect is that when grammar structure learning occurs, contextual information is added, and thus the contexts in which a rule may be applied become more restricted. This leads to reduced position independence. A second effect is that the rule is less likely to be shared within one individual, due to the increasing contextual information, and thus the individual becomes less modular. Therefore, the shorter the learning interval is, the more frequently contextual information is added. With more contextual information, the number of places that one rule may apply will be reduced very quickly. In other words, modularity and position independence are reduced.

Because the interval of grammar structure learning controls the degree of position independence and modularity, we can control the grammar structure learning interval to examine the impact of position independence and modularity on the behaviour of PRODIGY. By observing the performance change with different levels of modularity and position independence, we may analyse the interaction between them.

We expect that, with an increase in modularity and position independence, the performance will increase in general. However, this issue is intertwined with the learnability of PRODIGY and the expressiveness of the grammar. If the grammar is an extremely simple general grammar, it has very high modularity and position independence, but it does not help us to find the solution, because it covers too many individuals indiscriminately. We need to make the grammar more specific, to bias toward promising areas of the search space. In PRODIGY, this is done by grammar learning. Therefore, we would expect the performance would start to deteriorate again after reaching a certain length of interval of grammar structure learning.

In summary, in this section, we study how the grammar learning interval influences the position independence and modularity, and then in turn how these two impact on PRODIGY behaviour. We have formed the following three hypotheses.

Hypothesis 1 *With increased length of the grammar learning interval, the position independence will increase.*

Hypothesis 2 *With increased length of the grammar learning interval, the modularity will increase.*

Hypothesis 3 *With an increase in modularity and position independence, the performance will increase in general. However, the performance will start to then deteriorate after reaching a certain length of interval of grammar structure learning.*

The following three sets of experiments corresponding to above three hypotheses are reported in this section.

- Experiments on the correlation between grammar structure learning interval and position independence. In these experiments, the positional independence is measured by the amount of context. These experiments are to verify Hypothesis 1

- Experiments on the correlation between grammar structure learning interval and modularity. Modularity is measured by the amount of rule reuse. These experiments are to verify Hypothesis 2
- Experiments on the performances of PRODIGY with different levels of modularity and position independence. These experiments are to verify Hypothesis 3

4.3.2 Benchmark Problem

In this study, we chose as a test problem the first symbolic regression problem which we previously discussed in Section 4.2.2.

The total number of program evaluations was 100,000. The other settings were population size 10, maximum depth 15, mutation rate $c_m = 0.4$, and mutation intensity $mi = 0.2$, and the number of elitists was 1. We generated 6 sets of runs. Twenty runs were generated for each set. Among these 6 sets of runs, the structure learning happened every 1, 5, 10, 50, 100 and ∞ generations, respectively. The last setting ∞ means there is no structure learning at all. The fitness is measured by the sum of the absolute values of the error. Hence, this is a minimisation problem.

4.3.3 Results

4.3.3.1 Grammar Learning Interval and Positional Independence

In this section, we examine Hypothesis 1, i.e. the correlation between the length of the grammar structure learning interval, and position independence. We expect to see an increase of position independence with an increase of the length of learning interval. Position independence is measured by the amount of context in the grammar. The amount of context is the length of location l defined in Section 4.1.1. For example, if l only refers to the parent of current node, the length of l is 1 and if l refers to both the parent and grandparent, the length is 2, etc.

	Interval 1	Interval 5	Interval 10	Interval 50	Interval 100	Interval ∞
mean	7.6380	6.1141	5.1799	3.4730	2.6204	0
std	0.3615	0.5045	0.4342	0.4755	0.3206	0

Table 4.9: Average context length of last generations of runs for different learning intervals

The average context of the grammar can be found in Table. 4.9. The data in this table is also presented in Figure 4.10. The first data row is the mean of average length of context and the second one is their standard deviation. The columns represent different learning intervals. The average context length is around 7 with the length of interval 1 (the left most data column) while it decrease to around 2 with length of interval 100 (the second right most data column). The right most column represents the scenario with no grammar structure learning, and thus no context information is added. Therefore, it remains 0. It is clear that the amount of context decreases as learning interval increases. Less context information means more positional independence, as mentioned before. Hence, we can see from Table 4.9 that position independence increases as the learning interval increases.

4.3.3.2 Grammar Learning Interval and Modularity

In this section, we examine Hypothesis 2, i.e. the correlation between the length of the grammar structure learning interval, and modularity. We expect to see an increase of modularity with an increase of the length of learning interval. Modularity is measured by the average number of times that a rule is reused.

The average number of times that a rule is used among elitists can be found in Figure 4.11. The six plots represent the different learning intervals. The y-axis is the average number of times that a given rule is used, for simplicity called *average reuse*. Each plot represents the average rule reuse in the best individuals in each generation of a single run. The x-axis is the generation. Note that because the average reuse is under 8 in most cases, the y-axis is truncated to 7. The only

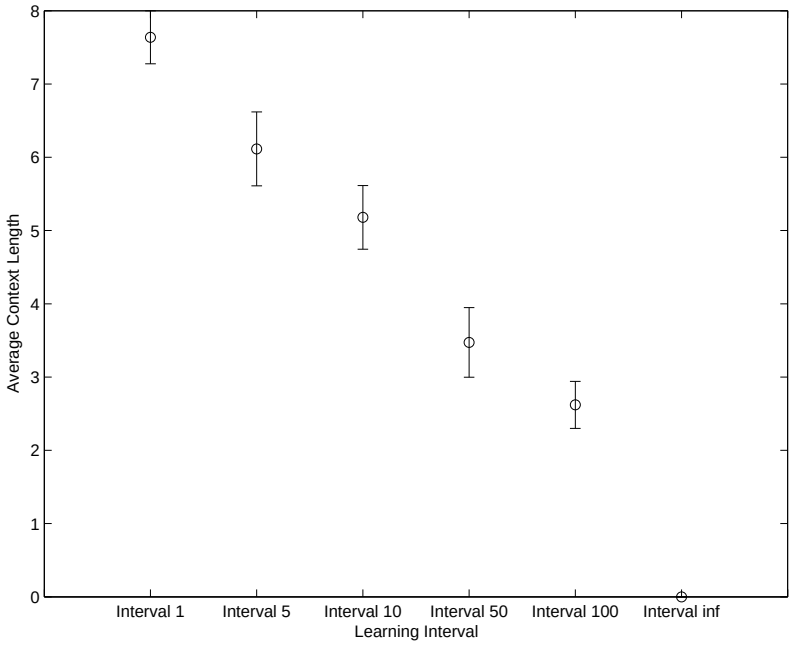


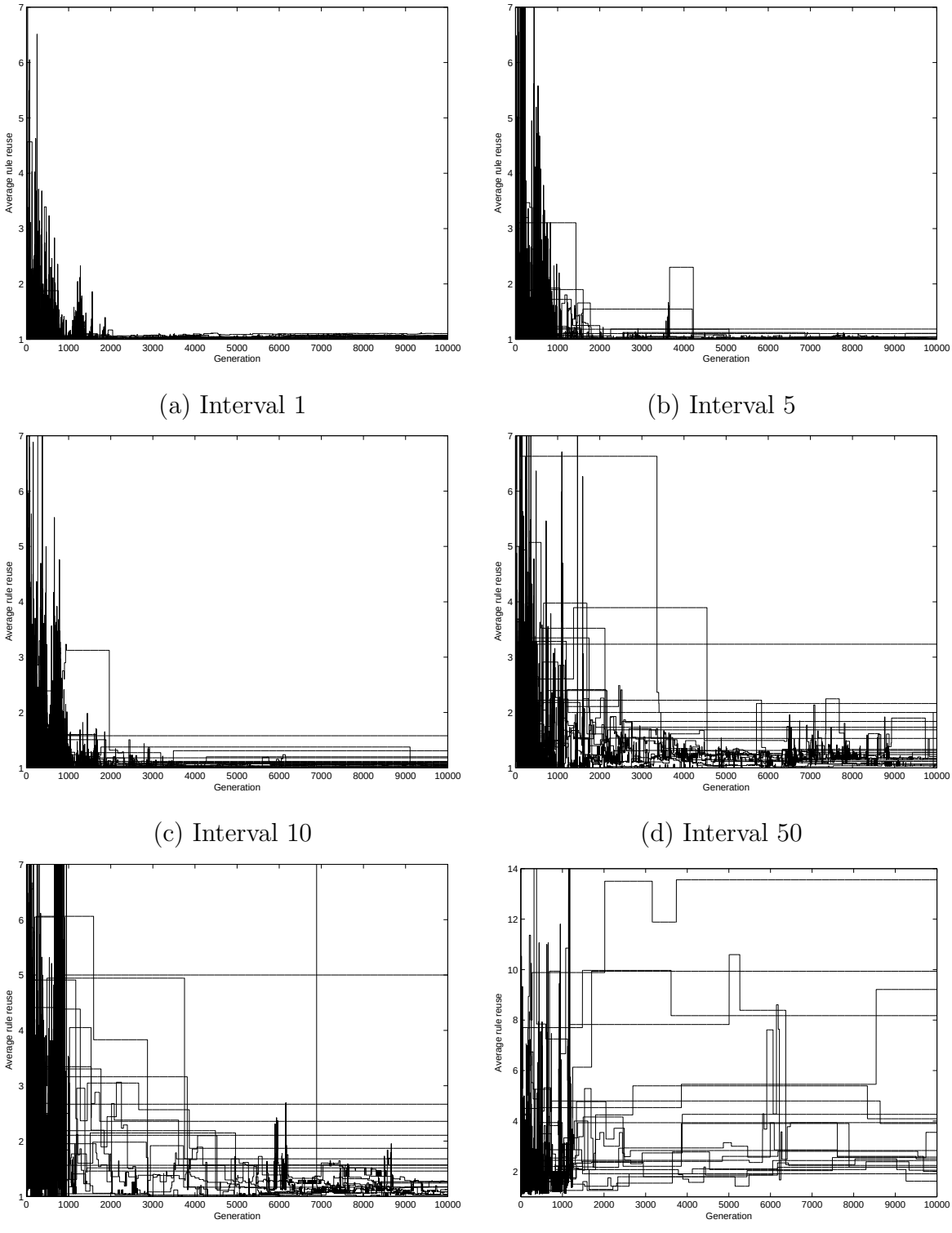
Figure 4.10: Average context length of last generations of runs at six different learning intervals

exception is Figure 4.11.f, in which the maximum y-axis value is 14.

The pattern in Figure 4.11 is clear. In Figure 4.11.a and b, which have intervals 1 and 5 respectively, rules are used only slightly more than once. When the interval increases to 10, average reuse also increases. When the interval reaches 50 and 100, the average reuse rises to an average of around 2. When the interval reaches ∞ (Figure 4.11.f), which means no structure learning, runs with an average reuse of 4 or more can be frequently observed (note that the y-axis is on a different scale in Figure 4.11.f). Figure 4.11 confirms our argument, that with an increase in the length of the learning interval, modularity also increases.

Another pattern clearly visible in all six figures in Figure 4.11 is that average reuse decreases as learning progresses. This also confirms our basic assumption, that the more learning happens, the more restricted are the locations where the rule can be applied and thus the less the modularity.

In Figure 4.11, all the runs are plotted in one figure. To better comprehend the



(a) Interval 1

(b) Interval 5

(c) Interval 10

(d) Interval 50

(e) Interval 100

(f) Interval ∞

Figure 4.11: Average reuse of rules among elitists at different learning intervals. Note the y-axis of last figure is on different scale.

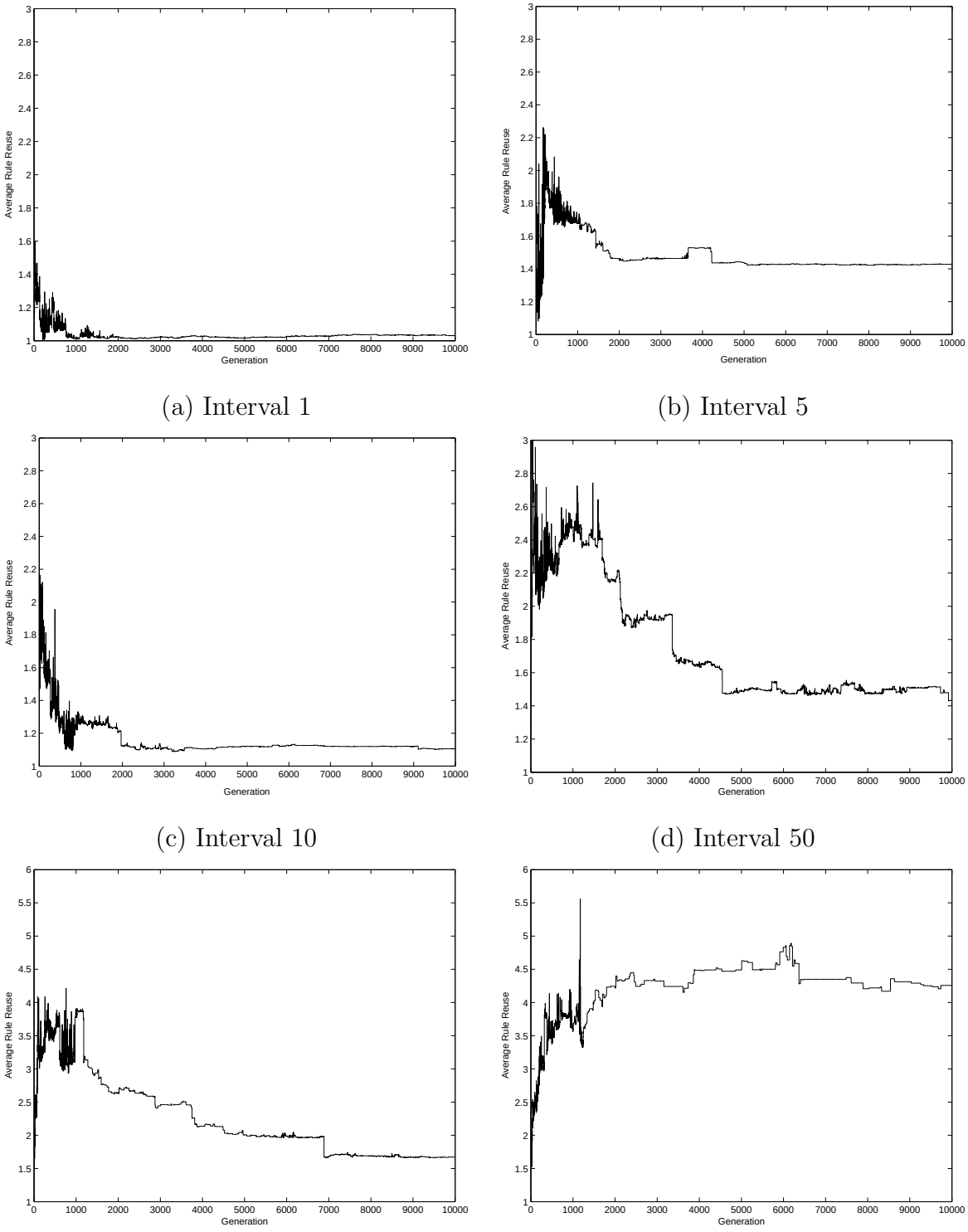
	Interval 1	Interval 5	Interval 10	Interval 50	Interval 100	Interval ∞
mean	3.3358	4.1301	4.0063	6.151	7.916	13.555
std	2.9581	5.8254	3.7165	6.169	10.4524	13.3019

Table 4.10: Maximum number of reuses of rules among elitists

pattern, we plot the means of all the runs according to its intervals in Figure 4.12. It shows a similar pattern to Figure 4.11. Figure 4.12.b should perhaps be ignored because it is severely affected by one abnormal run in which average reuse was more than 8 throughout the run, while the other runs all have less than 1.2 on average. Note that Figure 4.12.a – Figure 4.12.d are on one scale while the last two plots are on another.

To gain a more intuitive impression of rule reuse, we present the maximum reuse among elitists in Table 4.10. The columns represent different learning intervals. The first row is the mean of maximum reuse among elitists and the second one is their standard deviation. This table shows a similar pattern to Figure 4.11. In the runs with minimum learning interval (interval 1 at the first column), maximum rule reuse is only a little over 3, while when the interval reaches ∞ (last column), one rule can be used over 13 times.

We also notice the increase of standard deviation with increase of length of learning interval in Table 4.10. The data of table is also presented in Figure 4.13. This is consistent with our arguments about the interaction between length of interval and learnability/grammar expressiveness. If the length of interval increases too much, learning simply does not happen frequently enough to represent all the information. Consequently, the search exhibits strong instability (exploration) since the knowledge accumulated is not enough to focus the search in promising areas of the search space.



(a) Interval 1

(b) Interval 5

(c) Interval 10

(d) Interval 50

(e) Interval 100

(f) Interval ∞

Figure 4.12: Mean of average reuse of rules among elitists. Note the y-axes of last figures are on different scale.

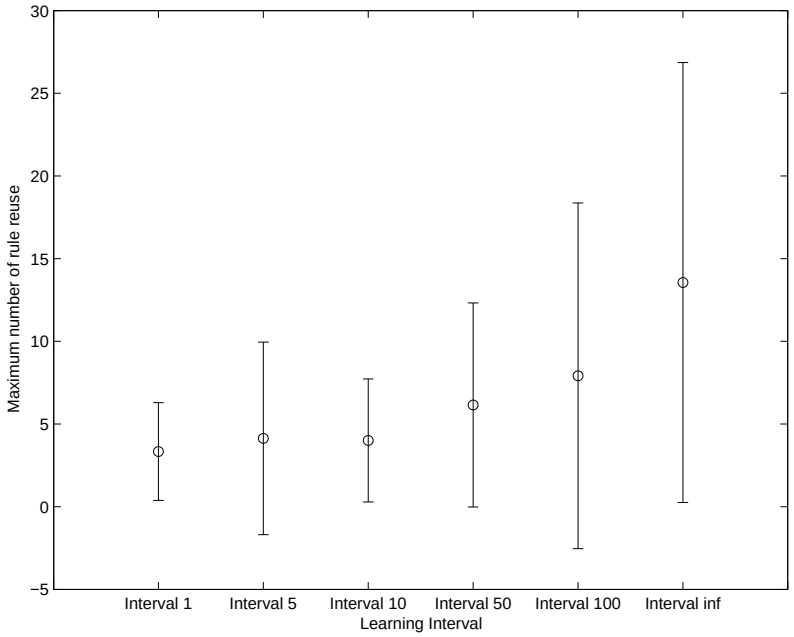


Figure 4.13: Maximum number of reuses of rules among elitists

4.3.3.3 Impact of Positional Independence and Modularity on Performance

We have demonstrated in previous sections the interaction between length of learning interval and positional independence/modularity where the positional independence and modularity can be controlled by the length of the learning interval. In this section, we examine Hypothesis 3, i.e. the impact on PRODIGY performance of the degree of positional independence and modularity, which are controlled by, once again changing the length of learning interval.

The fitnesses of the best individuals for six different learning intervals are plotted in Figure 4.14. In Figure 4.14, the x-axis is generation and the y-axis is fitness. The six curves represent the average best fitnesses at particular learning intervals. Obviously, this is a minimisation problem, where the smaller the value of fitness, the better.

In Figure 4.14, we see that the best performance is not obtained in the scenario with the most frequent structure learning, that is, the scenario with its length of

learning interval being 1. In that scenario, structure learning occurs every generation. However, such frequent grammar structure learning does not help in obtaining good performance. Furthermore, due to the overhead of grammar structure learning, this is the most computationally expensive scenario, in terms of both memory and computational time. This validates the hypothesis, that severely limiting position independence and modularity, by frequently adding contextual information, actually has a negative impact on performance. This is consistent with our previous theoretical analysis.

If we increase the interval, the performance initially increases as well. The best performance is obtained when grammar structure learning happens every 10 generations. This further confirms that by allowing position independence and modularity, search becomes more efficient.

However after this, if we further increase the length of the interval, the performance starts to drop. This is simply a result of insufficient structure learning. Without enough grammar structure learning, the grammar cannot represent all the information needed to refine the search. The extreme case is the top curve in Figure 4.14, where there is no grammar structure learning. In this case, the system degrades to a simple SCFG system with only probability learning. We cannot expect such a system to conduct sufficient search, due to the limited expressiveness of the SCFG with fixed structure.

4.3.4 Summary

The above experiments show that sharing and re-use of common sub-structures is important in EDA-GP and therefore should be encouraged. This may be explained from two perspectives. The first is that good solutions may have different forms but they all share some common structure – building blocks. Therefore, building blocks should be allowed to occur at different places, both in one individual and across the population. This leads to efficient discovery of good individuals. The second is

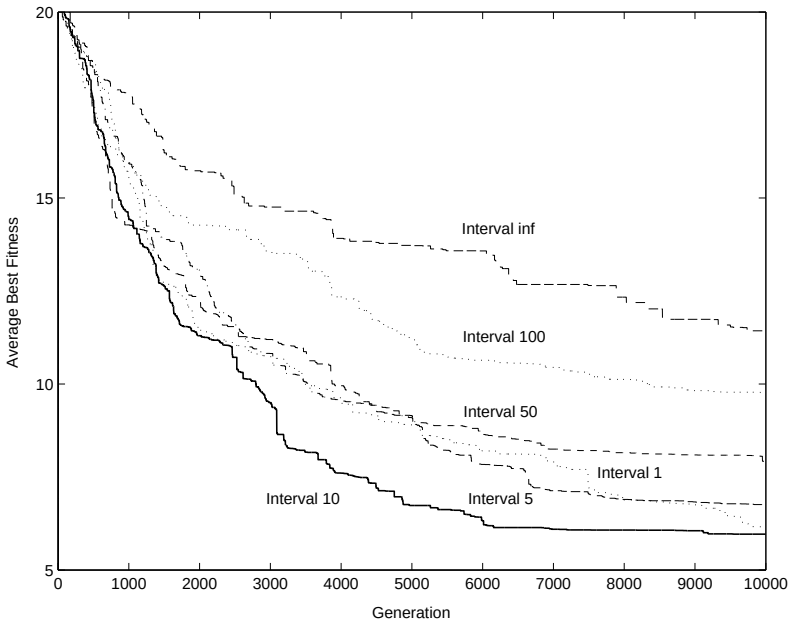


Figure 4.14: Average best fitnesses of runs at six learning intervals.

that, by allowing building blocks at various places, building blocks with the same structure but in different places can be treated as one building block by the learning algorithm. Thus they all contribute to learning of this one structure. The total number of training cases needed is thereby reduced, improving learning efficiency.

In this section, we firstly validated that position independence and modularity are determined by the length of the structure learning interval in our implementation of PRODIGY. Secondly, by changing the length of the learning interval, we obtained different levels of position independence and modularity. We also observed that the performance changes with the level of position independence and modularity.

During the PEEL search, it is expected that the specificity of the PEEL grammar model will increase. Although there is no known method to us to measure the specificity (or the size of the search space) directly in this context, the modularity and positional dependence, i.e. the number of rule reuse and the length of context, provide us an indirect indication of degree of specificity. That is, as the search progresses, the number of times rules are reused decreases, and the length of context increases. Both of these imply that specificity of the PEEL grammar model

specificity is increasing.

The empirical results of this section show the close interaction between performance and position independence and modularity. By encouraging position independence and modularity, the performance improves. This is consistent with the common belief in GP, which also employs trees to encode solutions. We also observed the phenomenon that insufficient structure learning has a negative impact on performance.

The experiments in this section support the view that the SCFG, the core of the PRODIGY framework, is an appropriate model, in the sense that it is capable of handling positional independence and modularity, and can (through changing the structure learning interval) provide a graduated trade-off between structure and probability learning.

4.4 Related Work

The first work in this area was PIPE (Salustowicz and Schmidhuber 1997), which we discussed earlier. PIPE differs from our system in two main aspects. Firstly, the probability at each node is independent from that at other nodes, while in our SCFG, the model probability is a conditional probability with respect to its parent. Secondly, the probability sits on the tree structure and thus no common structure can be shared, i.e. PIPE is absolute-position dependent while PEEL may share rules which are at the same depth.

Subsequently to PIPE, several other authors have based distributional GP approaches on the PIPE prototype tree, such as Estimation of Distribution Programming (EDP) (Yanai and Iba 2003) and Extended Compact Genetic Programming (ECGP) (Sastry and Goldberg 2003).

The PIPE prototype tree does not have a palpable problem handling individuals with varying complexity. However, the analysis in Section 4.3 has confirmed that,

in EDA-GP, building blocks should be allowed to move around and be reused, instead of being fixed at one position and then consequently used only once. Hence, when considering probabilistic models for EDA-GP, we need to take into consideration whether the model has this flexibility. We cannot see an obvious way for methods based on the prototype tree to handle position independence and modularity.

Grammar model based work, such as (Abbass, Hoai, and McKay 2002) and (Ratle and Sebag 2001), are similar to our implementation in that a grammar is used for EDA-GP; but the grammar structure is fixed, and only the attached probabilities are re-estimated in every iteration. It is noteworthy that both PEEL and method proposed in (Ratle and Sebag 2001) use depth as an additional constraint.

We have shown in Section 4.3 that learning the probability of a grammar with fixed structure is not sufficient for complicated problems. Learning the probability of a grammar with fixed structure is equivalent to runs with length of structure learning interval ∞ in our experiments in Section 4.3. This setting showed inferior performance in our experiments. This phenomenon is not hard to explain. Because what we are searching is a huge search space, grammars with fixed structure cannot represent all the accumulated information obtained from the continuous searching experience. The search must then stop when a grammar with fixed structure exhausts its capability to represent accumulated information from its searching experience. We expect methods of this kind, such as (Ratle and Sebag 2001; Abbass, Hoai, and McKay 2002), to suffer from this problem, in having bounded expressiveness.

Subsequently to our work (Shan, McKay, Abbass, and Essam 2003), Tanev (Tanev 2004) and Bosman (Bosman and de Jong 2004) independently proposed similar ideas, namely using a grammar for program evolution. Tanev added a fixed amount of learnt context to the grammar, to make the grammar more specific so that some part of the search space could be investigated more intensively. There are some important differences. In (Tanev 2004), grammar learning is used as an aid and the main search mechanism is conventional genetic search. More importantly, in (Tanev 2004) only a fixed amount of context information is added while in PEEL the context

information is incrementally added and thus the amount of context is adaptable.

The algorithm in (Bosman and de Jong 2004) started with a minimum SCFG and grammar specialisation was achieved by expanding production rules. It also used depth as an additional constraint as PEEL does. However, its learning methods are very different from PEEL.

4.5 Conclusion

We presented our PEEL implementation of PRODIGY. Experimental study confirmed the relevance of PEEL to problem solving. More importantly, using PEEL implementation, we empirically verified the significance of some of the properties of the model identified as important for EDA-GP.

Chapter 5

Minimum Encoding Inference of Grammars

In this chapter, we discuss Stochastic Context-free Grammar (SCFG) inference, which allows the inference of both the structure and the probability components of SCFG. We emphasise the use of minimum encoding metrics, as used in Minimum Description Length (MDL) and Minimum Message Length (MML) learning, as the quality measure of the inferred grammar. We argue that careful treatment of minimum encoding is critical in the PRODIGY approach to EDA-GP. In this research, we carefully derive a more accurate approximation of SCFG encoding length than that which is generally used in grammar inference. We argue that the differences between Natural Language Processing (NLP) and EDA-GP negate the use of tuning parameters which are common in NLP.

5.1 Introduction

Inferring an SCFG from derivation tree instances is a difficult problem. Usually, greedy search is used for the inference. In greedy search, the SCFG is altered using some variation operators. If after a given number of variations, a better SCFG is

found, it is accepted as the basis for the next iteration. Otherwise, the search usually stops. During the search, a metric is needed to compare the competing SCFGs.

One common approach to defining such a metric is through minimum encoding inference, such as Minimum Message Length (MML) (Wallace and Boulton 1968; Wallace and Freeman 1987; Wallace and Dowe 1999) or Minimum Description Length (MDL) (Rissanen 1989). In the remainder of this chapter, the term MML is employed to refer to minimum encoding inference.

In Machine Learning, when inferring a model from the training data (the observations), in addition to the performance of the model on training data, we are particularly interested in its generalisation, i.e. the performance of the model on unseen data drawn from the same distribution as the observations. These are not necessarily the same thing. It is often easy to find a model which performs perfectly on training data, but which might not work at all well on unseen validation data. One simple example is a model which simply enumerates the entire training data set.

Regarding generalisation, it is well known that simple models tend to generalise better on unseen data. Therefore, in this respect, we prefer a simpler model over models enumerating the entire training data set. On the other hand, an over-simplified model may not have good performance, even on the training data. Hence, the overall preferred model should not only have low complexity, but should also cover training samples well.

Given these intuitive understandings, a theoretically sound framework is needed. MML provides such a framework for inference. Grammar inference under the MML framework is discussed in this chapter, with emphasis on the approximation of the minimum coding length of SCFGs.

This chapter is organised as follows. A brief introduction to minimum encoding inference is presented in Section 5.2. Then, under the minimum encoding inference framework, our MML metric for grammar learning in PRODIGY is proposed in

Section 5.3. Related work is discussed in Section 5.4 and the final section is the conclusion.

5.2 Minimum Encoding Inference

In order to describe a set of data, if we know a priori that the data possesses some systematic pattern, standard techniques derived from Shannon's information theory may be used to encode the data more briefly than if they were purely random. Even if we don't know such a pattern a priori, if we can estimate the pattern, a briefer encoding may still be possible. We may first estimate the pattern, and then encode the data under the assumption that this is the true pattern. The encoded string thus has two parts: a description of the model, and the coding of the data given that model (this second part can also be viewed as a description of the errors of the model on the data).

A model reduces the information required to represent the data, but at the cost of also representing the structure and parameter values of the model. Therefore, a model is only worth considering if the shortening that the encoded data string achieves is more than compensated by the cost of encoding the model. This establishes a simple trade-off between the complexity of a model and its quality of fit. The preferred model, among a class of competing models, will be the one with the shortest total length. This inference method is called minimum encoding inference (Wallace and Boulton 1968; Wallace and Freeman 1987; Wallace and Dowe 1999; Rissanen 1989). As previously mentioned, in the remainder of this chapter, the term MML is employed to generally refer to minimum encoding inference.

MML gives a theoretically sound framework to balance these two factors, i.e. the complexity and the accuracy of the model. From the perspective of MML, the model we are looking for should minimise the cost of coding the given data. The cost of coding the data, $L(D)$ is the sum of the cost of coding the model $L(G)$ and the cost

of coding the data with the help of the model $L(D|G)$. Thus, we want to minimise

$$L(D) = L(G) + L(D|G) \tag{5.1}$$

where D is the data, G is the model and $L(X)$ is the cost of coding X.

For a more detailed discussion of MML and MDL please refer to (Wallace and Boulton 1968; Wallace and Freeman 1987; Wallace and Dowe 1999; Rissanen 1989).

5.3 MML Metric for Grammar Learning in PRODIGY

The MML metric for grammar learning is proposed in this section. Although this metric is specifically derived for grammar learning in PRODIGY, it is also applicable to other grammar learning and probabilistic graph learning applications.

5.3.1 MML Metric for SCFG

In the context of SCFG inference, the two-part message coding, specified in Equation 5.1, receives a specific interpretation: the total cost of coding training examples D (which are a set of sentences in this case), $L(D)$, is the sum of the cost of coding the SCFG model G, $L(G)$, plus the cost of coding the training samples given the SCFG model, $L(D|G)$.

Since the training examples are a set of sentences derived from the grammar, the message length of $L(D|G)$ is the negative logarithmic product of the probabilities of deriving these sentences from the grammar, as discussed in Section 2.1.1.2. That is, the probability of a sentence, p_i, is the probability of its derivation from the grammar D. Given a set of sentences (training samples) D, the probability of this entire set of sentences, p_D, is the product of the probabilities of all sentences:

$$p_D = -\log \prod_{i}^{D} p_i = -\sum_{i}^{D} \log p_i \tag{5.2}$$

The $L(G)$ component requires a measure for the cost of encoding the inferred SCFG. Although there are some rough estimates of $L(G)$ in the literature, they are not adequate for our purpose, as they over-estimate the grammar complexity (Stolcke 1994; Chen 1995). In the literature, this over-estimation is compensated by a fudge factor α, which is tuned to fit the data. However, direct application of this method in GMPE led to instability in the algorithm, with the size of the generated grammar fluctuating wildly from generation to generation. Theoretical analysis led us to recognise that the source of the problem was the inaccurate estimation of grammar complexity. We have derived a more accurate estimate of $L(G)$, eliminating the need for a fudge factor.

To encode an SCFG, we need to encode the names of its terminal symbols, the number of terminals, and the number of nonterminals. For each rule, we need to encode the LHS, the RHS, and the probability of the rule. In our coding scheme, to reduce the coding cost, we do not explicitly encode the LHSs. We only encode RHSs and then order them based on their LHSs. Thus, we just need to encode a way to partition this list of RHSs so that the first partition of the RHSs corresponding to the first LHS, the second partition of the RHSs corresponding to the second LHS, etc. Formally, we have

$$
\begin{aligned}
L(G) = {} & L(\text{names of terminal symbols}) && \text{(5.3a)}\\
& + L(N) + L(\Sigma) && \text{(5.3b)}\\
& + L(\text{grouping of LHS}) && \text{(5.3c)}\\
& + L(\text{RHS of production rule}) && \text{(5.3d)}\\
& + L(\text{probability of RHS for each nonterminal}) && \text{(5.3e)}
\end{aligned}
$$

where:

- The first term $L(\text{names of terminal symbols})$ (Equation 5.3a) is the length

of the coding names of the terminal symbols. For our purposes, this will be omitted from the calculation, because the same cost is incurred by all grammars under consideration. [1]. Once the problem is specified, the grammar terminal symbols are fully defined and are hence independent of any grammar structure variation.

- The terms $L(N)$ and $L(\Sigma)$ (Equation 5.3b) are the cost of coding the number of terminals and non-terminals, respectively. They can be calculated using Rissanen's methods of coding integers (Rissanen 1989). The length of coding integer N is $\log^*(N) = \log N + \log\log N + \log\log\log N + ...$ (only positive terms are included).

- Since we do not code the LHS of each rule, we need to state which RHSs correspond to the same LHS (Equation 5.3c). For example, in the grammar presented in Figure 5.1, we need to specify the following sequence:

  1, 2, 4

  which represents that the first rule has the first LHS symbol S, the following 2 rules share the second LHS symbol Exp and the last 4 rules share the last LHS symbol Op. Effectively, this means we partition all the rules into three partitions, 1, 2, 4. How can we code this partition? Suppose, for P items ($P \geq N$), we have $partition(P)$ unique ways to partition these P items. We just need to specify one of them, i.e. we need to specify an integer between 1 and $partition(P)$. Hence, the upper bound for this is the cost of coding the value of $partition(P)$. We use the upper bound of the Hardy-Ramanujan

[1] In this thesis, we follow traditional formal language terminology, in which the terminal symbols include all the directly expressive symbols of the grammar, both function symbols and symbols for variables (and thus include what would be referred to in traditional GP terminology as both terminals and non-terminals)

asymptotic formula (Andrews 1984) for $partition(P)$: [2]:

$$partition(P) = \frac{\exp(\pi\sqrt{\frac{2P}{3}})}{4P\sqrt{3}} \tag{5.4}$$

Besides this, we also need to specify the number of items (total number of rules in this case) P. Therefore, the total cost of L(grouping of LHS) is the sum of these two, i.e. $L(P) + L(partition(P))$. Because both are integers, Rissanen's method for coding integers can again be used.

- Equation 5.3d is the length of coding for the RHS of each rule. We use N for both the set of non-terminals and the size of this set. We use Σ in the same way for terminals. Thus, the total number of distinct symbols is $N + \Sigma$. Using a fixed coding scheme, each of them needs $\log(N + \Sigma)$ bits to code. Suppose the total number of occurrences of symbols on RHSs is m, then the total cost of coding them is $m \log(N + \Sigma)$. Alternatively, if the frequencies of the symbols are known, then variable length coding may be used. For a symbol Y whose probability, approximated from its frequency, is p_Y, the cost of coding an occurrence of this symbol is $-\log p_Y$. In our application, the difference between these two encodings is minimal, so the simpler fixed coding scheme is used.

- The last term, Equation 5.3e, reflects the encoding of the probabilities of production rules. Some LHSs have more than one RHS. When deriving sentences, we need to know with what probability we replace a given LHS with each RHS. Hence, we need to record this probability. More precisely, we record the frequencies, which will then be normalised, in order to obtain the probabilities. We prefer skewed probability distributions, with probability more strongly approaching either 0 or 1, as opposed to the uniform distribution, because this means we have less uncertainty. This can be better explained by taking an simple example. Suppose we only learn the probabilities of a model and leave

[2] In this case, the order of partitions does matter. For example, composition $1, 1, 2$ differs from $2, 1, 1$ or $1, 2, 1$, whereas they are the same partition. However, we can impose an ordering for coding purposes (say, ascending) and so we need distinguish only partitions.

the structure fixed. Initially, due to lack of knowledge, the probabilities of the model follow a uniform distribution. We expect, with an appropriate learning method, after iterations of learning, that the probabilities will converge to 0 and 1. This newly learnt model certainly describes the solution space better, covering the fitter individuals with higher probability, while the initial model covers all individuals (with the same number of derivation steps) indiscriminately. This tells us that the model we prefer is a model with probabilities approaching 0 and 1. This preference for a skewed probability distribution can be modelled by a symmetric Dirichlet prior (Buntine 1990). The last term, Equation 5.3e, can be calculated as the sum of the following equation over all nonterminals. Since the detailed derivation of this term is rather complex, we leave it to Section 5.3.2.

$$\begin{aligned} & L(\text{probability of RHS for one nonterminal}) \\ =& MML(\hat{\theta}, D_n, \alpha_i) \\ =& -\sum_{i=1}^{C} \log \hat{\theta}_i^{\alpha_i + n_i - \frac{1}{2}} + \log B_C(\alpha_1, \ldots, \alpha_C) \\ & + \frac{1}{2}(C-1)\log n + \frac{C}{2}(1 + \log \frac{1}{12}) \end{aligned} \tag{5.5}$$

where D_n represents the given data set (a set of rules, with associated counts, which share a given LHS), C is the number of different RHSs with this LHS, n_i is the frequency of the i-th RHS, $n = \sum_{i=1}^{C} n_i$, and $B_C()$ is the Beta function defined later. $\alpha_i (i \neq 0)$ are predefined parameters, $\alpha_0 = \sum_{i=1}^{C} \alpha_i$, and $\hat{\theta}_i = \frac{n_i + \alpha_i}{n + \alpha_0}$.

5.3.1.1 Example

Given the grammar illustrated in Figure 5.1, we derive the encoding cost, $L(G)$, following Equation 5.3, as follows:

- Equation 5.3a. This term is omitted because it is the same for all admissible models.

S	→	Exp				(0)
Exp	→	Exp	Op	Exp	|	(1)
	→	x				(2)
Op	→	+	|			(3)
	→	-	|		(4)
	→	*	|		(5)
	→	/				(6)

Figure 5.1: A simple grammar example

- Equation 5.3b. The terms $L(N)$ and $L(\Sigma)$ are the cost of coding the number of terminals and non-terminals. There are 3 non-terminals and 5 terminals. This gives $N = 3$ and $\Sigma = 5$. Thus, $L(N) = \log^* 3 = \log 3 + \log\log 3 = 1.099 + 0.094 = 1.19$ and $L(\Sigma) = \log^* 5 = \log 5 + \log\log 5 = 1.609 + 0.476 = 2.09$.

- Equation 5.3c. The calculation of term $L(\text{groupings of LHS})$ is as follows. In this example, there are 7 RHSs, $P = 7$ and thus the cost is $L(P) = \log^* 7 = 2.61$. The cost of coding the partition is $L(\mathit{partition}(P)) = L(\mathit{partition}(7)) = \frac{\exp(\pi\sqrt{\frac{2\cdot 7}{3}})}{4\cdot 7\sqrt{3}} = 11.04$. Thus, $L(\text{grouping of LHS}) = L(P) + L(\mathit{partition}(P)) = 2.61 + 11.04 = 13.65$

- Equation 5.3d. The term $L(\text{RHS of production rule})$ is the cost of coding the RHS of each rule. We have 7 distinct symbols (omitting the start symbol since it does not occur on the RHS; there are 2 nonterminals and 5 terminals). There are 9 occurrences of these symbols. Using the fixed coding scheme, the cost is $9 \cdot \log 7 = 17.51$.

- Equation 5.3e. The last term is the cost of coding the probability distributions of the rules. This will be discussed in the following Section 5.3.2.

5.3.2 Cost of Coding the Probability Distribution of an SCFG

5.3.2.1 Multinomial Distribution

Given a LHS symbol, the probability distribution of choosing the RHS can be described by the multinomial distribution. The multinomial distribution is a discrete distribution which gives the probability of choosing a given collection of n items from a set of C items with repetitions, where the probabilities of each choice are given by $\theta_1, \ldots, \theta_C$.

5.3.2.2 Dirichlet Prior for Multinomial Distribution

For the multinomial distribution, the Dirichlet distribution has a number of practical advantages as a prior. A symmetric Dirichlet prior has the form:

$$p(\hat{\theta}) \propto \prod_{i=1}^{C} \hat{\theta}_i^{\alpha_i - 1} \tag{5.6}$$

where $\alpha_1, \ldots, \alpha_C$ are parameters of the prior.

A Dirichlet prior is a conjugate prior because it has the same form as the likelihood. This makes the posterior distribution a simple product of likelihood and prior:

$$p(\hat{\theta}|D_n) \propto \prod_{i=1}^{C} \hat{\theta}_i^{n_i + \alpha - 1} \tag{5.7}$$

where D_n are the n observations.

Notice that the above two equations are proportional. To generate a distribution, we need as a normalising term a C-dimensional beta function:

$$B_C(\alpha_1, ..., \alpha_C) = \frac{\prod_{i=1}^{C} \Gamma(\alpha_i)}{\Gamma(\alpha_0)} \tag{5.8}$$

where $\alpha_0 = \sum_{i=1}^{C} \alpha_i$ and

$$\Gamma(n+1) = n! \tag{5.9}$$

where n is an integer.

The Dirichlet distribution is then:

$$p(\hat{\theta}) = \frac{1}{B_C(\alpha_1, ..., \alpha_C)} \prod_{i=1}^{C} \hat{\theta}_i^{\alpha_i - 1} \tag{5.10}$$

The parameters α_i can be interpreted as "prior observation counts" for events governed by θ_i. For $0 < \alpha_i < 1$, the prior is biased toward the extremes, $\theta_i = 0$ and $\theta_i = 1$. $\alpha_1 = \alpha_2 = \ldots = \alpha_C = 1.0$ corresponds to the uniform prior. In our application, we make the underlying assumption that the problem is learnable with an SCFG, corresponding to the assumption that the system will eventually be able to generate highly biased probabilities; this influences our choice of α values for the Dirichlet prior. The Dirichlet priors for given α values are illustrated in Figure 5.2. As can be seen, for $0 < \alpha < 1$ this distribution imposes bias toward the extremes of the parameter space, i.e. toward $\theta = 0$ and $\theta = 1$. For $\alpha = 1$, the distribution becomes noninformative and thus imposes a uniform prior. For $\alpha > 1$, this distribution becomes a bell shape and the variances will get smaller as the parameter α grows.

5.3.2.3 Cost of Coding Probability Distribution with the Dirichlet Prior

For C classes and n data points, the cost of coding this data is (Wallace and Freeman 1987):

$$MML(\hat{\theta}, D_n, \alpha_i) = -\log \frac{p(\hat{\theta})p(D_n|\hat{\theta})}{\sqrt{F(\hat{\theta})}} + \frac{C}{2}(1 + \log \frac{1}{12}) \tag{5.11}$$

where $F(k)$ is the Fisher Information term.

For Dirichlet distributions, the Fisher Information term (Allison 2003) is

$$F(\theta) = \frac{n^{C-1}}{\prod_{i=1}^{C} \theta_i} \tag{5.12}$$

where n is the number of data items.

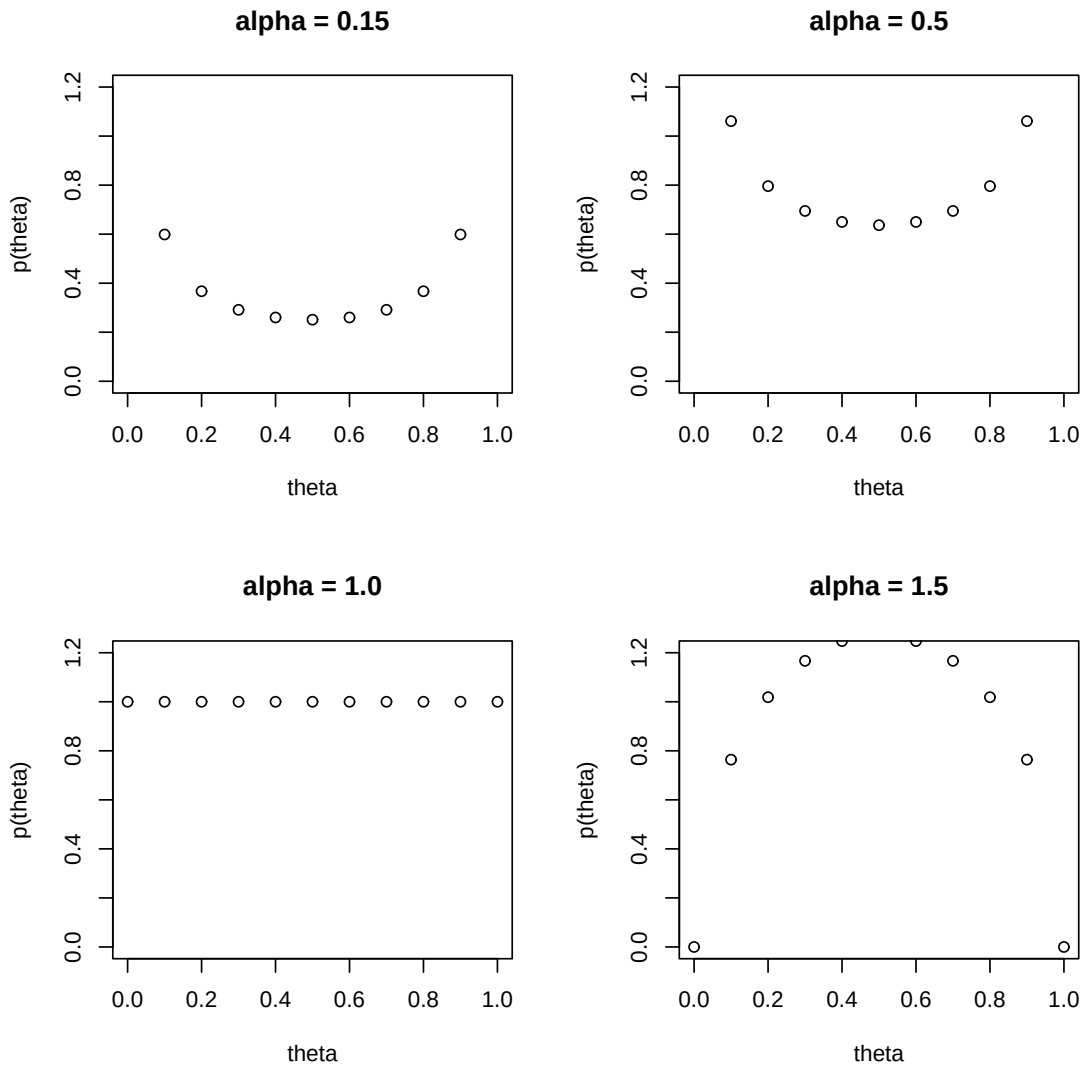


Figure 5.2: Dirichlet prior at different α values

Putting it all together gives:

$$\begin{aligned} MML(\hat{\theta}, D_n, \alpha_i) = &- \sum_{i=1}^{C} \log \hat{\theta}_i^{\alpha_i + n_i - \frac{1}{2}} + \log B_C(\alpha_1, \ldots, \alpha_C) \\ &+ \frac{1}{2}(C-1)\log n + \frac{C}{2}(1 + \log \frac{1}{12}) \end{aligned} \tag{5.13}$$

where $\hat{\theta}_i = \frac{n_i + \alpha_i}{n + \alpha_0}$.

5.3.2.4 Example

Assume we have the following grammar fragment:

Exp	$\rightarrow$	Exp	Op	Exp	|	1
	$\rightarrow$	x				3

where the number after each rule is the count of rule usage. These can be normalised to obtain probabilities. The cost of recoding this probability distribution is calculated using Equation 5.13 as follows.

There are two alternatives rules in this grammar and thus $C = 2$. A skewed distribution is preferred, so we take $\alpha_i = 0.15$. The cost of coding this probability distribution is[3]:

$$\begin{aligned} MML(\hat{\theta}, D_n, \alpha_i) &= -\log \left(\frac{1 + 0.15}{4 + 0.3}\right)^{0.15+1-0.5} - \log \left(\frac{3 + 0.15}{4 + 0.3}\right)^{0.15+3-0.5} \\ &\quad + \log B_2(0.15, 0.15) + \frac{2-1}{2}\log 4 - \frac{2}{2}(1 + \log \frac{1}{12}) \\ &= 0.85725 + 0.82471 + 2.55983 + 0.69315 - 1.48491 \\ &= 3.4500 \end{aligned}$$

[3]Throughout this chapter, we use base e for logarithms.

5.3.3 MML Metrics in Natural Language Processing and PRODIGY

Minimum encoding inference, in particular the MML metric, can be used in SCFG inference in both NLP and PRODIGY. Although MML is used in similar ways in both fields, the differences between them bring up some subtle issues that have to be considered when transferring the grammar inference ideas from NLP to PRODIGY and EDA-GP. The difference in learning procedures of these two is one of the important issues. Grammar learning in NLP is one-shot learning, while in PRODIGY it requires multiple iterations. This issue was previously mentioned briefly in 3.3.3.2; we now present the more MML-specific aspects.

5.3.3.1 Multiple Iterations of Grammar Learning

In most NLP applications, the carefully calculated estimation for $L(G)$ developed above is usually replaced with a less accurate approximation. To compensate, Equation 5.1 is re-formulated, and a fudge parameter λ is introduced to balance the error from the $L(G)$ term:

$$L(D) = \lambda L(G) + L(D|G) \tag{5.14}$$

Because grammar learning in NLP is one-shot learning, i.e. learning only happens once, the λ factor can be experimentally determined to accurately balance the two terms in Equation 5.14.

However, EDA is a multi-shot learning algorithm, i.e. it is an iterated search of grammar learning and grammar sampling. It may not be possible to find a λ which is appropriate for all the grammar learning phases.

A simple transformation of Equation 5.14 can make this clearer. Because $L(D|G)$ can be accurately measured, Equation 5.14 effectively suggests:

$$\lambda L(G) = L(G)_{true} \tag{5.15}$$

where $L(G)_{true}$ is the true cost of encoding grammar G. But, $L(G)$ can be rewritten as

$$L(G) = L(G)_{true} + error(L(G)) \tag{5.16}$$

where $error(L(G))$ is the extra cost introduced by an inaccurate encoding method. By substituting Equation 5.16 into Equation 5.15:

$$\begin{aligned} \lambda(L(G)_{true} + error(L(G)) &= L(G)_{true} \\ \lambda &= \frac{1}{1 + \frac{error(L(G))}{L(G)_{true}}} \end{aligned} \tag{5.17}$$

Equation 5.17 shows that assuming a fixed λ implies that $error(L(G))/L(G)_{true}$ is constant. Unfortunately, this is unlikely to be true in multi-shot learning. $L(G)$ is the sum of two encoding costs: of the structure and of the probabilities. There is no reason to suppose that the error structure of these two components is similar. Hence as the grammar complexity increases during learning (as we expect), it is unlikely that the error will increase proportionately. This analysis highlights the importance of the accuracy of the MML metric in the context of multi-shot learning.

5.4 Related Work in Natural Language Processing

Related work on MML-based grammar inference is reviewed in this section. Since this review is from the perspective of the MML metric, the specific grammar learning methods of these works are omitted. The approximation to the MML metric in most of these works is very different from ours. Of them, (Stolcke 1994) and (Chen 1995) are the most directly relevant to our approach.

5.4.1 Chen

Chen proposed a MML metric for SCFG learning in (Chen 1995). Chen uses a restricted SCFG representation, in which the valid forms of rules are restricted to

a number of types. The coding of each rule is as follows. The LHS does not need to be encoded because in (Chen 1995) each LHS nonterminal symbol only has one expansion, and thus we just need to order them (the LHS of the first rule is the first nonterminal symbol, of the second rule is the second symbol, etc.) For the RHS, we need to encode its type and RHS symbols. There are 8 types, requiring log(8) bits to encode. For encoding a RHS symbol, each symbol can be assigned a unique code and thus encoding RHS symbols is simply listing the code of each symbol. For rules which need to record probabilities, the count is recorded and then normalised to obtain the associated probability. The count n is encoded using the universal MDL prior over the natural numbers $p_{MDL}(n)$ (Rissanen 1989), where

$$p_{MDL}(n) = \frac{6}{\pi^2} \frac{1}{n[log_2(n+1)]^2} \tag{5.18}$$

The length $l(n)$ of an integer n is $log_2 \frac{1}{p_{MDL}(n)}$.

This work differs from ours in two aspects. The first lies in the classification of rules into 8 types with the corresponding encoding. The second lies in the encoding of counts of rule usage as natural numbers, while we treat them as a probability distribution. This means that we are able to impose an appropriate prior, which is important in the context of PRODIGY. In PRODIGY, a certain probability distribution is preferred. Without treating the counts of rule usage as a probability distribution, this preference cannot be expressed.

5.4.2 Stolke and Omohundro

The method proposed in (Stolcke 1994) is similar to (Chen 1995). They are both based on a Bayesian framework. Thus, Stolke and Omohundro also employs the MDL/MML principle. Stolke uses a Dirichlet prior to code the probabilities of the SCFG, which is similar to ours but different from (Chen 1995) and less operators are used.

Although both of (Stolcke 1994) and our work use a Dirichlet prior, different prior observation counts, α, are chosen, reflecting different preference of the model. Fur-

thermore, in our work, only the RHS of the rules are encoded. We expect this would reduce the total cost of coding a grammar.

5.4.3 Grunwald

(Grunwald 1994) uses partial grammars, a subset of SCFG. This very brief paper, which abbreviates crucial details, claims that $DL(C)$ (the description length of the grammar structure) can be ignored. Unfortunately, the encoding for the probability component is not explained in detail.

5.4.4 Keller and Lutz

Keller and Lutz (Keller and Lutz 1997) used a Genetic Algorithm to learn the probabilities of SCFGs. $L(G)$ is encoded in (Keller and Lutz 1997) as follows. The SCFG is specified by a set of pairs of the form (r, w) where r is a rule and w is a weight:

$$L(G) = \sum_{i \in G} (L(r_i) + L(w_i)) \tag{5.19}$$

Weight w is encoded using Rissanen's prefix code and thus its length is:

$$L(w) = log_2(w) + 2log_2(log_2(w)) + 1$$

Keller and Lutz depart from the common practise of measuring $L(r)$ as the length required to explicitly encode the symbols of rules. Instead, $L(r)$ is calculated as the negation of the logarithm of the probability of a rule. This probability is calculated as follows.

A probability distribution $\hat{p}(r)$ is defined over the rules. The method for computing the probability of each r_j is to look at the probability of the symbol in each position of the set of rules in the covering grammar. In particular, we look at the set of symbols

allowed at a given position in the rules, given the preceding symbols. Assuming that all legal symbols at that position can occur with equal probability, $\hat{p}(r)$ can be computed as follows.

Let N be the set of nonterminal symbols in the covering grammar G, and let n be the length of the RHS of the longest rule. Now write each rule r_j in the form $s_{j_0} \rightarrow s_{j_1} s_{j_2} \ldots j_n$, where s_{j_m} is a special "blank" symbol for all m greater than the length of the right hand side of rule r_j (i.e. in effect, each rule is padded to the maximum length). Define $S_{j_k}(0 \geq k \geq n)$ to be that set of symbols given by:

$$S_{j_k} = \begin{cases} N & \text{if } k = 0; \\ (S_{j_k} | \exists r_i \in G \text{ and } s_{i_l} = s_{j_l} (0 \geq l \geq k)) & \text{otherwise} \end{cases}$$

The probability $\hat{p}(r_j)$ of rule r_j is then given by

$$\hat{p}(r_j) = \sum_{0 \geq k \geq n} p_{j_k}$$

where $p_{j_k}(0 \geq k \geq n)$ is defined as $\frac{1}{|S_{j_k}|}$.

In summary, the probabilities of particular symbols are computed position by position. Assuming each symbol has equal probability, the probability of choosing a particular symbol at a given position (or the probability of this position) is the inverse of the number of possible symbols at this position. The probability of a complete rule is the product of the probabilities of each position.

5.4.5 Osborne

(Osborne 1999) studied the induction of Definite Clause Grammars (DCGs). DCGs are treated in much the same way as SCFGs. The rules (except for the probabilities) are encoded in a similar way to (Keller and Lutz 1997).

Rissanen's prefix coding is used to encode a rule's frequency (where the probabilities are not directly encoded, following Keller and Lutz (Keller and Lutz 1997)). However, instead of coding frequency i, the cost is calculated using $log^*(Z - i)$, where

Z is a number larger than any frequency, on the argument that higher frequency values should be favoured.

5.5 Conclusion

In this chapter, we have presented a brief introduction to minimum encoding inference and have reviewed the current work in minimum encoding grammar inference in the field of NLP. A careful derivation of the MML metric for the Stochastic Context-free Grammar inference in the context of PRODIGY has been presented. Although this MML metric is derived for PRODIGY, it is also applicable to learning of SCFG in other applications and is generally relevant to the inference of probabilistic graph models. The possible applications might include plots and story lines for interactive fiction, structured questions for on-line tutoring systems, generative artworks in the blind watchmaker tradition and perhaps even architectural design.

Chapter 6

Grammar Model-based Program Evolution

This chapter presents one of the possible approaches under PRODIGY framework, Grammar Model-based Program Evolution (GMPE). Due to the flexibility of the grammar model, and the effective grammar learning method, GMPE significantly outperforms GP on some of the benchmark problems we have studied. In terms of the number of individuals evaluated, it is both faster and more reliable. More importantly, GMPE removes some constraints which are not necessary parts of PRODIGY, such as the depth constraint of admissibility of grammar rules used in PEEL, and it is capable of inferring a full SCFG grammar, with both probabilities and structure. As a result, GMPE can show a number of the properties of PRODIGY more clearly than other PRODIGY implementations, and consequently it provides an effective tool for studying these properties.

6.1 Introduction

Grammar Model-based Program Evolution (GMPE), as an approach under the PRODIGY framework, adopts a different grammar learning method from the PEEL

approach. Its learning method is heavily motivated by grammar learning methods from the field of Natural Language Processing (NLP). This provides a strong theoretical basis, and avoids re-inventing the wheel.

The basic algorithm of GMPE is consistent with the PRODIGY framework. GMPE employs a fully SCFG model, which makes it more generally applicable than other methods. The SCFG model of GMPE is a standard SCFG with no extra constraints. Both the structure and the probability distribution of the SCFG are inferred in GMPE.

This chapter is organised as follows. In Section 6.2, an alternative view of EDA is provided. The grammar learning method of this research is heavily motivated by Bayesian grammar induction from Natural Language Processing. Background on this topic is provided in Section 6.3. GMPE is presented in Section 6.4, and is experimentally studied in Section 6.5. Its relationship with other works is discussed in Section 6.6. The last section is the conclusion.

6.2 Problem Decomposition

EDA is closely related to problem decomposition. This view is consistent with the perspective of estimation of distribution of potential solutions, but with a different emphasis. In the iterative process of model learning and sampling, EDA in effect attempts to decompose the problem into sub-problems, such that closely interacting genes can be understood as sub-solutions for these sub-problems. Some EDA methods are explicitly designed to solve problems that can be decomposed, as for example (Pelikan, Goldberg, and Cantú-Paz 2000; Mühlenbein and Mahnig 1999).

Similarly, in EDA-GP, especially GMPE, we are particularly interested in problems which can be solved constructively in a bottom-up fashion. In other words, we are interested in problems having the following two properties:

- it has sub-trees as partial solutions;

- its final solution can be obtained by growing or combining partial solutions.

A partial solution is defined as a sub-tree which positively contributes to the overall fitness. Therefore, the individuals which contain partial solutions are, on average, favoured in the selection stage. This is consistent with the common belief about building blocks in EC, that the number of building blocks grows as search progresses. Consequently, the frequency of a sub-tree, in particular the frequency of a sub-tree among different individuals, indicates the quality of that sub-tree as a partial solution.

From the perspective of problem decomposition, the expected behaviour of EDA-GP is far more complicated than that of conventional EDA. In EDA-GP, the small sub-solutions have to be discovered, and then assembled in some particular order in order to obtain bigger sub-solutions at a higher level. Finding a sub-solution is not enough. EDA-GP needs to be able to preserve these sub-solutions, and explore their combinations, in order to find the sub-solutions of the higher level.

6.3 Bayesian SCFG Grammar Induction

There are several studies in Natural Language Processing (NLP) on learning an SCFG from a corpus using minimum encoding inference (Stolcke 1994; Chen 1996). In this research, we are mainly inspired by (Stolcke 1994). This uses a specific-to-general search method, where the inference of the SCFG model is regarded as searching for the optimal model among the possible model solutions. A model which only covers training samples is initially constructed. Search operators are then imposed on the model to vary it, usually to generalise it. Since we cannot enumerate all the possible sequences of operations, a hill-climbing method is adopted. The metric from Minimum Encoding Inference determines which search operations to accept, and also when to stop the search.

An alternative potential approach (Chen 1996) starts from a general grammar which

covers the entire search space, and then specialises it by adding more rules. This does offer potential computational efficiencies, and like most specialising learning methods offers some advantages in handling noisy data. However, there are potential difficulties in ensuring that the learnt grammars remain consistent with the initial grammar. These issues will be discussed in more detail in the conclusions and future work sections, but will not be further considered in the body of this thesis.

6.3.1 The Method

There are two basic components in SCFG inference:

1. search operators (model variation operators),
2. scoring metric.

Regarding search operators, there are two essential operators – merge and chunk (Stolcke 1994). They are used to variate the current model in searching for an optimal one. Merge changes the coverage of the model, as well as its probability distribution. Chunk helps to construct the hierarchical structure which is then subject to merge. Regarding the scoring metric, which is used to compare the quality of the model, it is derived within a Bayesian inference framework, which provides a principled method to trade off model accuracy for model simplicity.

The method in (Stolcke 1994) is a hill-climbing search which may be trapped in local optima. However, in our research this has not been as serious an issue as might be anticipated, because the grammar model is only an intermediate outcome in the iterative search process. Even if a local optimum is found in one generation of the algorithm, it is possible that the stochastic variation induced by sampling from the SCFG in the next generation may help to dislodge it from this local optimum. In fact, given the search operators and scoring metric, the search does not have to be hill-climbing – more reliable search methods could be substituted, though probably

at the cost of computational efficiency. We have not pursued this further here, in the light of the satisfactory results achieved by hill-climbing.

6.3.1.1 Merge operator

If we are given a derivation tree, the only operator needed to construct an SCFG model is the merge operator. The merge operator takes two rules and unifies their LHSs to one symbol.

$$\begin{aligned} X_1 &\rightarrow \lambda_1 \\ X_2 &\rightarrow \lambda_2 \\ &\Downarrow \quad \text{merge}(X_1, X_2) = Z \\ Z &\rightarrow \lambda_1 \\ Z &\rightarrow \lambda_2 \end{aligned}$$

After merge, all the occurrences of X_1 and X_2 in the grammar are replaced by Z. As can be seen, before merge, X_1 can only be rewritten with λ_1, while after merge, X_1 and X_2 are not distinguishable, so that the places where X_1 appears can be rewritten with either λ_1 or λ_2. The same applies to X_2. It is not hard to see that the merge operator usually generalises the grammar. Before the merge, the two rules had different LHSs; after the merge, the two rules share LHSs, so the merged grammar can cover more strings.

The probability of each rule is approximated by its frequency. Therefore, each rule is attached with a count, which can later be normalised to obtain the probability. The merge operator may lead to adjustments of the count. For example, two previously distinguishable rules may become identical after merge. In this case, the redundant rules are removed and the count is accumulated. For example, take the following hypothetical grammar fragment,

$$
\begin{array}{lcll}
S & \rightarrow & Y & 1 \\
S & \rightarrow & B & 1 \\
Y & \rightarrow & X_1 & 1 \\
Y & \rightarrow & X_2 & 1 \\
B & \rightarrow & X_2 & 1 \\
X_1 & \rightarrow & \lambda_1 & 1 \\
X_2 & \rightarrow & \lambda_2 & 1
\end{array}
$$

where the number at the end is the count of the rule (which is often omitted if it is not germane to the discussion). After imposing merge$(X_1, X_2) = Z$, we obtain

$$
\begin{array}{lcll}
S & \rightarrow & Y & 1 \\
S & \rightarrow & B & 1 \\
Y & \rightarrow & Z & 1 \\
Y & \rightarrow & Z & 1 \\
B & \rightarrow & Z & 1 \\
Z & \rightarrow & \lambda_1 & 1 \\
Z & \rightarrow & \lambda_2 & 1
\end{array}
$$

By then removing the redundant rules and re-adjusting the count, we obtain

$$
\begin{array}{lcll}
S & \rightarrow & Y & 1 \\
S & \rightarrow & B & 1 \\
Y & \rightarrow & Z & 2 \\
B & \rightarrow & Z & 1 \\
Z & \rightarrow & \lambda_1 & 1 \\
Z & \rightarrow & \lambda_2 & 1
\end{array}
$$

The relevance of the EC conception of building blocks to GMPE can also be illustrated through this example. The string λ_1 can be regarded as a sub-tree with X_1 as its root. Initially, B can only be followed by sub-trees with X_2 as their root. After

merge$(X_1, X_2) = Z$, a sub-tree with X_1 as its root can also follow B. So in this example the sub-tree with X_1 as its root is moved around. This is similar to what happens in GP crossover. Through merge, if it is beneficial to fitness, the whole sub-tree can be relocated and promoted. Thus we can see a clear analogy to the desired dynamics of building blocks in EC.

6.3.1.2 Chunk Operator

While the chunk operator is important in learning stochastic context free grammars in natural language processing and many other applications of grammar learning, it is not necessary in GMPE. We include it here for the sake of completeness. To learn an SCFG model from a set of strings without structural information (as is the case, for example, in learning a natural language grammar from sentences), we have to construct the hierarchical structure. The chunk operator is designed for this purpose. Recall that, in an SCFG, the function of nonterminals is mainly to represent the hierarchical structure. The chunk operator constructs this hierarchical structure by introducing nonterminals. It takes an ordered sequence of symbols $X_1 X_2 \ldots X_k$, and creates a new nonterminal Y that expands to $X_1 X_2 \ldots X_k$. It then replaces occurrences of $X_1 X_2 \ldots X_k$ with Y.

$$\begin{aligned} Z &\rightarrow \lambda X_1 X_2 \ldots X_k \mu \\ &\Downarrow \quad \text{chunk}(X_1 X_2 \ldots X_k) = Y \\ Y &\rightarrow X_1 X_2 \ldots X_k \\ Z &\rightarrow \lambda Y \mu \end{aligned}$$

The chunk operator does not change the probability distribution specified by the original model. The simple justification is as follows. The newly introduced symbol Y in the above definition is a new symbol and thus is not referred to in any other rules except

$$Z \rightarrow \lambda\ Y\ \mu.$$

Thus it has no impact on any other rules beside this. For this rule, Y has only one right hand side. Hence, the probability of rewriting Y with $X_1X_2 \ldots X_k$ is 1. Consequently, the rule is a simple substitution, and will not change the probability of deriving any string involving the symbol Z.

Because the chunk operator does not change the probability distribution specified by the grammar, its only function being to construct the hierarchical structure, it is not necessary in GMPE. In EDA-GP, we obtain individuals by generating them from the current model (grammar). Thus, as in GGGP, the individuals are derivation trees and already have a complete hierarchical structure. Consequently, we have no need of a chunk operator to restore the hierarchical structure, unlike the situation in NLP. In NLP, commonly a set of sentences, i.e. a set of linear strings of words, are given as training examples. The internal hierarchical structure of these sentences are initially unknown. In this case, the chunk operator is needed to make explicit the hidden hierarchical structure.

It is interesting to look at the chunk operator from some other perspective. The chunk operator may be reminiscent of ADF to some extent. The chunk operators replace a string of symbols with one new symbol and then this new symbol can be used to refer to this string. Automatically Defined Functions (ADF) of GP (refer to Section 3.2.4) encapsulate a sub-tree as intermediate building blocks so that they can be used multiple times. Thus, it is not hard to see the some similarities between chunk operation and ADF in GP.

6.3.2 Scoring Metric

The scoring metric is based on MML, as previously discussed in Chapter 5 and hence omitted here.

1. Generate a population $\mathcal{P}$ from a given initial SCFG model $\mathcal{M}_0$.
2. Select a set of fitter individuals $\mathcal{G}$ from $\mathcal{P}$.
3. Construct the SCFG model $\mathcal{M}$ from $\mathcal{G}$.
4. Sample the SCFG model $\mathcal{M}$ to obtain a set of new individuals $\mathcal{G}'$.
5. Replace population $\mathcal{P}$ with $\mathcal{G}'$.
6. If the termination condition is not satisfied, go to 2.

Figure 6.1: Basic algorithm of GMPE

6.4 Grammar Model-based Program Evolution

6.4.1 Algorithm

GMPE is consistent with the basic PRODIGY algorithm. It starts with a randomly generated population. An SCFG model is learnt from selected individuals from this population. A new generation is generated by sampling the learnt model. Subsequently, the next iteration will start from this new population. The high level algorithm is illustrated in Figure 6.1.

6.4.2 Learning the Grammar Model

The algorithm used for learning the grammar in step 3 in Figure 6.1 is a simplified stochastic hill-climbing method. The merge operator is repeatedly applied to the grammar until the grammar does not improve any more with respect to the MML metric. The basic algorithm is as follows:

1. Retrieve rules to obtain the primitive grammar $\mathcal{M}$
2. Merge (IGC merge) two randomly chosen rules from grammar $\mathcal{M}$ to obtain new grammar $\mathcal{M}'$
3. If the MML metric decreases, accept the new grammar, i.e. $\mathcal{M} = \mathcal{M}'$, else restore the previous grammar $\mathcal{M}$
4. If there is no improvement in the MML metric in n_t consecutive trials, exit else go to 2

In this algorithm, there are two issues which need further refinement. The first is the retrieval of rules from the set of individuals to obtain the primitive grammar. The second is the Initial-Grammar-Consistent merge operator (IGC merge). This is a variant of the standard merge operator, modified to enforce consistency between the learnt grammar and the initial grammar. We refer to the grammars constructed at the start of each iteration, to cover the selected individuals, as the *primitive grammars*. The minimal grammar $\mathcal{M}_0$, which is specified to define the search space at the start of the run, often incorporates problem-dependent knowledge supplied by the user. It is referred to as the *initial grammar*.

Although the grammar learning method uses stochastic hill-climbing to reduce the computational cost, it is still very computationally expensive. It is not feasible to enumerate all the possible merges and choose the most beneficial one when the number of rules is large. The number of merges is quadratic in the number of rules (and hence in the population size). Moreover, for each merge, the scoring metric has to be calculated, and this is itself expensive for big grammars. Instead, we set the number of trials n_t, limiting the number of unproductive merges examined, as a parameter of the algorithm, although this potentially hinders the accuracy of the model, resulting in models which may be over-specialised (i.e. fewer merges are conducted than are justified by the data). There is potential for varying the algorithm to reduce computational cost, but most such variations carry a concomitant risk of under- or over-generalisation. Examination of how to determine appropriate trade-offs of efficiency versus accuracy in the inner, grammar-learning stage of the algorithm is

S	→	Exp				1
Exp	→	Exp	Op	Exp	|	1
	→	x				1
Op	→	+	|			1
	→	-	|		1
	→	*	|		1
	→	/				1

Figure 6.2: Grammar for simple symbolic regression problem $f(x) = x^4 + x^3 + x^2 + x$

viewed as beyond the scope of this thesis, whose main purpose is to introduce a general framework and investigate its ramifications. In this initial investigation of the framework, we have necessarily erred on the side of caution, sacrificing potential computational efficiencies to try to ensure algorithm correctness. Therefore, in the experimental section, we examine some toy problems whose scale is within reach of the algorithms as proposed here, in order to understand, and try to characterise the behaviour of the system, rather than tackle the efficiency problem in order to handle hard real world problems.

6.4.2.1 Obtaining the Primitive Grammar

The Primitive Grammar for a particular generation is obtained by retrieving rules from the individuals in the corresponding selected population. These individuals are derivation trees. The retrieval involves two steps.

1. Traverse the tree and assign a unique ID to each of the nonterminals.

2. Take each sub-tree of depth 1 (i.e. two levels), and re-form it as a single rule. The left hand side of this rule is the label of the root of the sub-tree, and the right hand side consists of the leaves of the sub-tree in left-to-right order. Attach to each rule the original grammar symbol associated with it, to indicate its origin from the initial grammar (this information will be used in the IGC

merge).

Example Assume we have the initial grammar illustrated in Figure 6.2 and that given this initial grammar, the two individuals shown in the upper part of Figure 6.3 are generated. Since all the individuals share the same starting symbol S, only one S appears in the figure.[1] In the bottom part of the figure, we assign a unique ID to each of the nonterminals. Thus, we obtain the following set of rules as the primitive grammar.

S	→	R1(Exp)			1
S	→	R2(Exp)			1
R1(Exp)	→	R3(Exp)	R4(Op)	R5(Exp)	1
R2(Exp)	→	R6(Exp)	R7(Op)	R8(Exp)	1
R3(Exp)	→	x			1
R5(Exp)	→	R9(Exp)	R10(Op)	R11(Exp)	1
R4(Op)	→	-			1
R6(Exp)	→	x			1
R7(Op)	→	-			1
R8(Exp)	→	x			1
R9(Exp)	→	x			1
R10(Op)	→	+			1
R11(Exp)	→	x			1

[1]Generally speaking, a grammar can be represented as an AND/OR graph, where all the symbols on the RHS of a single rule are related through AND logic, while the different RHSs sharing the same LHS are related through OR logic. In this specific figure, all the nodes are related through AND logic except the children of root S. S can either go to $R1$ or $R2$. That is, it uses OR logic. Although it would be more appropriate to express this difference in the figure, omission of this detail does not jeopardise our discussion. Therefore, for brevity, this difference will usually be omitted in figures.

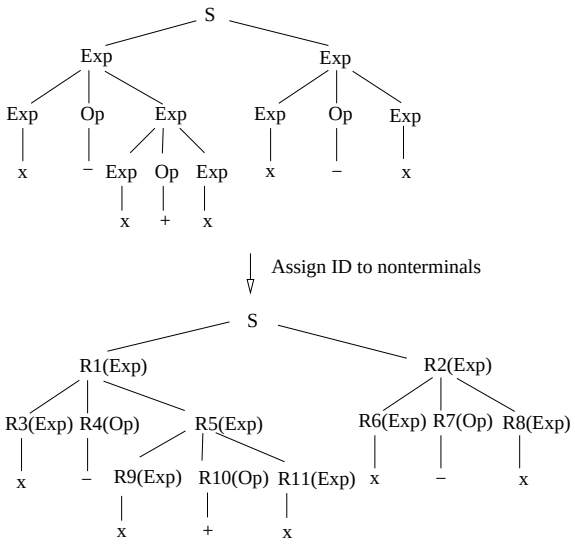


Figure 6.3: Retrieval of rules from a set of individuals to construct primitive grammar for GMPE

6.4.2.2 Initial-Grammar-Consistent Merge Operator

The Initial-Grammar-Consistent merge operator (IGC merge) is a straightforward modification of the conventional merge operator, with the restriction that only rules with the same origin (i.e. that come from the same rule of the initial grammar) can be merged. We impose this restriction so that the learnt grammar always generates a sub-language of the initial grammar, respecting the initial grammar's role in defining the search space. That is, the rules learnt should be consistent with the initial grammar $\mathcal{M}_0$.

The strings in parentheses in figure 6.3 are the original strings (origins), which are retained so as to ensure the legitimacy of IGC merge operation. Only rules sharing the same origins can be merged. For example, in the above grammar, $R3$ cannot merge with $R4$ because they have different origins (Exp and Op respectively). If they were to merge, the new grammar could generate individuals which may not have been able to be generated by the initial grammar in Figure 6.2.

Example Given the primitive grammar, the following are some examples of IGC merges. For example, by imposing merge $(R8, R9) = R12$ on the above primitive grammar, we obtain:

S	→	R1(Exp)			1
S	→	R2(Exp)			1
R1(Exp)	→	R3(Exp)	R4(Op)	R5(Exp)	1
R2(Exp)	→	R6(Exp)	R7(Op)	R12(Exp)	1
R3(Exp)	→	x			1
R4(Op)	→	-			1
R5(Exp)	→	R12(Exp)	R10(Op)	R11(Exp)	1
R6(Exp)	→	x			1
R7(Op)	→	-			1
R10(Op)	→	+			1
R11(Exp)	→	x			1
R12(Exp)	→	x			2

Note that the count of $R12$ is adjusted after the merge. In this case, the RHSs of the two rules involved in the merge are identical, so that this merge does not generalise the grammar. Since this kind of merge does not change the sentences covered by grammar or their probability distributions, it guarantees the decrease of the MML metric (the accuracy of this model does not change while the complexity reduces, thus the whole cost is reduced. Please refer to the Equation 5.1). For brevity, we refer to merge operations which merge rules with identical RHSs, and which therefore do not change the distribution of individuals generated from the grammar as *unifications*.

With the exception of unifications, merges usually generalise the grammar. For example, if we apply merge $(R2, R5) = R13$ to the above grammar, we obtain:

S	→	R1(Exp)			1	
S	→	R13(Exp)			1	
R1(Exp)	→	R3(Exp)	R4(Op)	R13(Exp)	1	
R3(Exp)	→	x			1	
R4(Op)	→	-			1	
R6(Exp)	→	x			1	
R7(Op)	→	-			1	
R10(Op)	→	+			1	
R11(Exp)	→	x			1	
R12(Exp)	→	x			2	
R13(Exp)	→	R6(Exp)	R7(Op)	R12(Exp)	1	|
	→	R12(Exp)	R10(Op)	R11(Exp)	1	

This grammar becomes more general because it covers more sentences than before. Originally R2(Exp) could only be rewritten with R6(Exp) R7(Op) R12(Exp) and R5(Exp) could only be rewritten with R12(Exp) R10(Op) R11(Exp). After merge, the places where these two symbols appear can be rewritten either way. Therefore, the new grammar extends its coverage. For example, the string

x + x

can be generated by applying the following rules from the above grammar

S	→	R13(Exp)			1
R10(Op)	→	+			1
R11(Exp)	→	x			1
R12(Exp)	→	x			2
R13(Exp)	→	R12(Exp)	R10(Op)	R11(Exp)	1

However this string was not covered by the grammar before applying the merge operator. Hence, the new grammar is more general. As previously mentioned, because the merge operator considers the origin of rules, strings generated from the newly learnt grammar, like this one, are still consistent with the initial grammar.

6.4.2.3 Merge and Crossover

Interestingly, the merge operator in grammar learning shares some of the effects that crossover in conventional GP brings to individuals. By moving sub-trees, both of them generate new structures that did not exist before. In the previous example, imposing merge $(R2, R5) = R13$ on the grammar illustrated in Figure 6.3, is effectively similar to imposing a crossover which swaps the sub-tree with root $R2$ and the sub-tree with root $R5$, while also retaining the original individuals in the population. From this perspective, merge resembles some kind of generalised crossover.

However, despite the similarity, they are intrinsically different. One of the most important differences between these two operators is the subject that they are imposed on. The merge operator is imposed on a grammar model, and thus it is used to search for some optimal model with respect to some metric. Crossover is imposed on the individuals of a population to introduce genetic variation. Thus, the merge operator progressively generalises the grammar model, while crossover creates new individuals and consequently moves the population to a new spot in the search space.

More specifically, crossover generates a limited number of new individuals. For example, one crossover operator produces two individuals, which are usually different from their parents. However, merge makes models more general. The new grammar after merging might cover many more new individuals if the merged symbols are the LHSs of multiple rules. Therefore, in this occasion, the merge operator generates many more variations than crossover does. It is even possible that the new grammar might cover an infinite number of individuals, if merge creates a recursive structure.

6.4.3 Sampling the Learnt Grammar

Sampling SCFG grammars has been discussed earlier in Section 2.1.1. To summarise, the rules are repeatedly stochastically sampled based on their probabilities, and applied to the nonterminals until no nonterminals are left.

Since the nonterminal symbols are simply an indication of structure, it is not necessary to record their names when generating new individuals. Only their origins need be recorded.

For example, the two individuals in the upper part of Figure 6.3, may be examples generated from the learnt grammar above. Note that only the origin appears in the individuals, while the names of nonterminals are removed.

It is possible that the straightforward sampling of the learnt grammar may produce the individuals exceed the depth limit. The problem has been studied in GGGP research (Ross 2001; Whigham 1995b). The same methods can be used to enforce depth limit in GMPE. For problems with simple initial grammar, each of whose symbols has at least one rule deriving from this symbol which has only terminal symbols on RHS, the depth limit can be easily imposed by forcing to randomly choose one of these rules when reaching the depth limit.

6.4.4 Restoration of the Initial Search Bias

A population-based search method may lose its population diversity before the solution is found, especially when the population (the set of training examples) is small. This phenomenon, known as premature convergence, occurs when the search becomes trapped in a local optimum.

In this research, a predefined number of individuals generated from the generation 0 initial grammar are introduced at each generation, to partially restore the initial (uniform) prior. This has similar effects to the mutation operator in EC.

6.5 Experimental study

In this section we describe three experiments we conducted to verify the proposed method, and to study its performance. Among these three problems, the first two

$$
\begin{array}{lcl}
\mathrm{S} & \rightarrow & \mathrm{N} \\
\mathrm{N} & \rightarrow & \mathrm{N} \quad | \\
 & \rightarrow & \mathrm{N}\ \mathrm{N} \quad | \\
 & \rightarrow & \mathrm{N}\ \mathrm{N}\ \mathrm{N} \quad | \\
 & \rightarrow & \mathrm{N}\ \mathrm{N}\ \mathrm{N}\ \mathrm{N} \quad | \\
 & \rightarrow & \mathrm{N}\ \mathrm{N}\ \mathrm{N}\ \mathrm{N}\ \mathrm{N} \quad | \\
 & \rightarrow & \mathrm{x}
\end{array}
$$

Figure 6.4: Initial grammar for Royal Tree Problem

have unique solutions and are thus well studied in the GP community because of their ability to illustrate certain properties of the GP methods, while the last is a commonly used symbolic regression problem. These will be further discussed in Section 6.5.4.

One of the major performance measures of GMPE in this section is the number of evaluations, which is a common measure widely used in the EC literature. One reason is that this is an implementation independent measure across the different EC algorithms, unlike the actual execution time. Another reason is that number of evaluations is important if the evaluation cost is high (with respective to the computational cost on the other parts of the algorithm), which is almost always the case in real world applications.

6.5.1 Royal Tree Problem

6.5.1.1 Problem Description

The Royal Tree Problem (Punch, Zongker, and Goodman 1996) was designed as a difficult problem for GP. In this experiment, we use the level-E Royal Tree Problem.

This problem has one terminal x and a series of nonterminals A, B, C, D, E etc. with increasing arity. For example, these five nonterminals have arity 1, 2, 3, 4 and

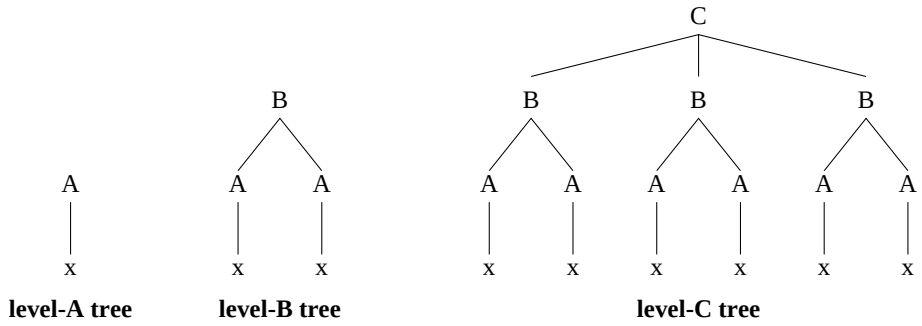


Figure 6.5: Examples of perfect trees of Royal Tree Problem

5 respectively. The perfect solution for this problem is a complete full tree, with nonterminals of the biggest arity at the root and the nonterminals of second biggest arity at the next level, etc. The fitness reflects the resemblance to this perfect solution, and was designed to encourage searching for the solution in a bottom up manner. The fitness of the perfect solution of the level 5 Royal Tree Problem is 122,880.

More specifically, for any depth, we define a "perfect" tree as shown in Figure 6.5 adopted from (Punch, Zongker, and Goodman 1996). A level-A tree is an A with a single x child. A level-B tree is a B with two level-A trees as children. A level-C tree is a C with three level-B trees as children, and so on. A level-E tree has depth 5 and 326 nodes, while a level-F tree has depth 6 and 1927 nodes. The raw fitness of the tree (or any sub-tree) is the score of its root.

Each node calculates its score by summing the weighted scores of its direct children. If the child is a perfect tree of the appropriate level (for instance, a complete level-C tree beneath a D node), then the score of that sub-tree, times a FullBonus weight, is added to the score of the root. If the child has the correct root but is not a perfect tree, then the weight is multiplied by a PartialBonus. If the child's root is incorrect, then the weight is multiplied by Penalty. After scoring the root, if the function is itself the root of a perfect tree, the final sum is multiplied by CompleteBonus. Typical values used are: FullBonus = 2, PartialBonus = 1, Penalty = 1/3, and CompleteBonus = 2. The score base case is a level-A tree, which has a score of 4 (the A—x connection is worth 1, times the FullBonus, times the CompleteBonus).

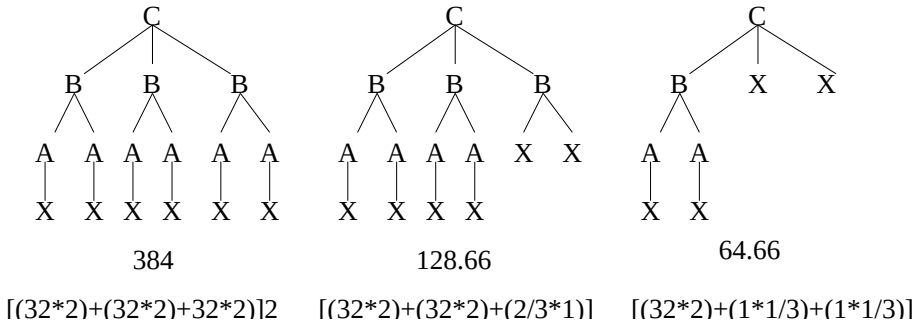


Figure 6.6: Example trees with scores of Royal Tree Problem

The reasoning behind this "stairstep" approach is to give a big jump in evaluation credit to each proper combination, so that the problem can be solved by progressively discovering sub-solutions and combining them. The FullBonus is provided to give a large credit to those trees that find the correct, complete royal tree child. The PartialBonus is used to give credit for finding the proper, direct child for a node, even if that direct child is not the root of a royal tree. This pressure is not as great as the FullBonus, but it is an effective incentive, since the score is determined recursively down the tree, and thus each node receives some credit when if finds its proper, direct children. If a node does not have the correct, direct children, it is penalised by Penalty, making the FullBonus and PartialBonus even more effective. Finally, if the resulting tree is itself complete, then a very large credit is given. The reasoning behind the increase in arity required at each increased level of the royal tree is to introduce tunable difficulty.

Some examples of complete and partial royal trees are shown in Figure 6.6 adopted from (Punch, Zongker, and Goodman 1996). In this figure, the method of scoring is given. For example, in the leftmost tree, the score of a complete level B tree is 32, so the score of a complete level C tree is $[(32 \cdot FullTree) + (32 \cdot FullTree) + (32 \cdot FullTree)] \cdot CompleteTree$.

In this experiment, the settings for GMPE are population size 60, selection rate 50%, maximum depth 5 (i.e. the fitness of the perfect solution within this depth limit is 122,880), and $\alpha = \alpha_i = 0.15$ for all i. We use truncation selection, i.e. in each

Population size	60
Generation	20,000
Maximum depth	5
Mutation rate	0.005
Prior observation counts α_i	0.15
Selection	Truncation
Total number of runs	55

Table 6.1: Parameter settings for GMPE on Royal Tree Problem

generation, the top 50% of the population are selected to build the model and the second 50% are replaced by the individuals generated from the model. To maintain diversity, ten randomly generated individuals are introduced in each generation, i.e. randomly generate 10 individuals from initial grammar and insert them into the population (these individuals are not subject to selection procedure). We conducted 55 runs. The major setting can be found in Table 6.1

The grammar for GMPE can be found in Figure 6.4. This grammar is designed so that it does not impose any more bias to the search than conventional GP, so that we can compare the performance of these two. This is achieved by having only one nonterminal in the grammar beside the starting symbol. Consequently, any pair of rules can potentially be merged. Thus this grammar does not introduce any syntax constraints.

6.5.1.2 Results

The cumulative frequency of successful runs is illustrated in Figure 6.7, where the horizontal axis is the number of individual evaluations, and the vertical axis is the percentage of successful runs. As we can see, around 80% of runs succeed before 2×10^5 individual evaluations, and overall around 90% percent of all runs succeed within 6×10^5 individual evaluations. The average best fitnesses of all runs are plotted in Figure 6.8. The error bar is deceiving. From Figure 6.7, it is easy to

see that majority of runs succeed after 2×10^5 individual evaluations. However, the error, standard deviation of best fitnesses of all runs, is still big even at the later stage of the search. The reason is that the fitness function of this problem is designed so that the fitness increase is exponential as search progresses. Thus, at later stage even if there is only a small proportion of runs having bad fitness, they are enough to dominate the standard deviation.

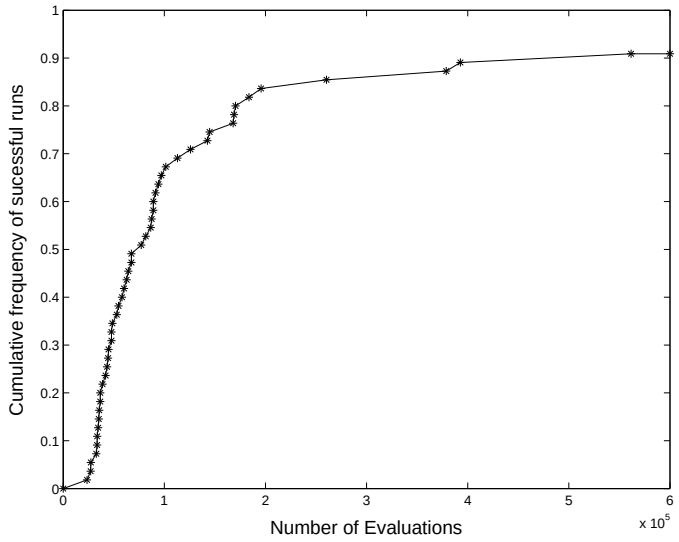


Figure 6.7: Cumulative frequency of successful runs of GMPE on Royal Tree Problem.

The Royal Tree Problem is a very difficult problem for GP. We present a comparison in Table 6.3 to show the relative performance of GMPE. The GP statistics for the problem are drawn from (Punch, Zongker, and Goodman 1996). Several GP settings were tried in (Punch, Zongker, and Goodman 1996). We compare GMPE with the best GP settings, population size 3500, crossover probability 0.875, reproduction probability 0.075, mutation probability 0.05, maximum depth 17, generation 500, over-selection. The major settings are listed in Figure 6.2.

Table 6.3 shows that with 1,750,000 evaluations (*population size* $3500 \times$ *generation* $5000 = 1,750,000$), only 50% of GP runs succeed. To achieve the similar rate of successful runs, GMPE only needs 77,220 ($30 \times 2574 = 77,220$) evaluations. We define a simple measure *speedup*, similar to the measure in (Michalski 2000), to give a clearer

Population size	3500
Generation	500
Maximum depth	17
Crossover rate	0.875
Mutation rate	0.05
Reproduction rate	0.075
Section	Over-selection
Total number of runs	16

Table 6.2: Parameter settings for GP on Royal Tree Problem

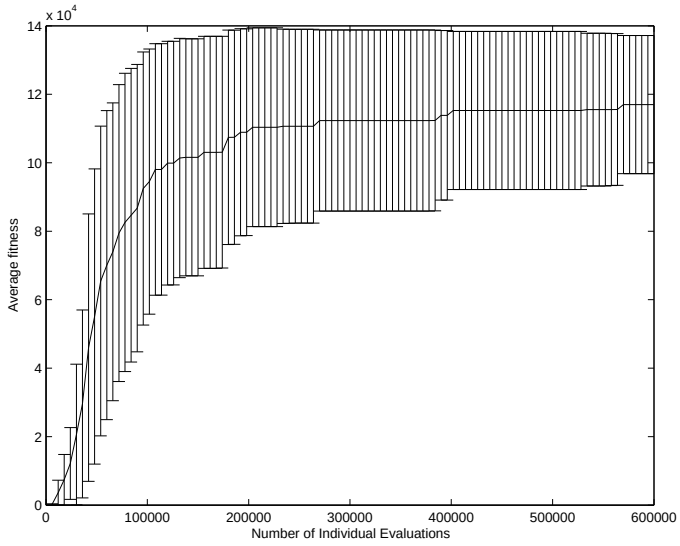


Figure 6.8: Average best fitness of all runs of GMPE on Royal Tree Problem.

impression of the relative performance of GMPE to GP. Given the rate of successful runs δ,

$$speedup_{\delta} = \frac{Number\ of\ Evaluations\ of\ GP}{Number\ of\ Evaluations\ of\ GMPE} \tag{6.1}$$

In this experiment, $speedup_{50\%} = \frac{1750000}{77220} \approx 22.7$. Under this measure, we can say that GMPE is 22.7 times faster, in terms of number of evaluations, for a 50% rate of successful runs.

We have used a different maximum depth in the GMPE experiments than those used in the GP experiments. The GP experiments used a relatively large maximum depth

of 17. Because of the relatively high space complexity of GMPE relative to the tree depth, this maximum depth is infeasible for us. This may compromise comparability. However the nature of the problem makes it highly likely that solutions can only be discovered by building from small partial solutions and then combining them into bigger partial solutions until the final solution is found. Certainly, the failed GP runs were trapped in local optima with small partial solutions. That is, for GP the difficulty of this problem is to escape from local optima, not the exploration of a large search space. This suggests that a smaller maximum tree depth would not have had a significant positive impact on the performance of the GP algorithms. To validate this, we duplicated the GP experiments with depth 6 on this Royal Tree Problem. The other settings are similar to (Punch, Zongker, and Goodman 1996): crossover probability 0.875, reproduction probability 0.075, mutation probability 0.05, fitness proportionate selection, ramped half-and-half initialisation. Twenty-five runs were conducted. None of these runs succeeded and majority of these runs had fitnesses less than 100.

Although it is not feasible to use a large maximum depth, such as 17, which is used in (Punch, Zongker, and Goodman 1996), it is possible to employ slightly larger depths for GMPE. We did conduct a number of runs using larger depths, such as 6. At this higher depth limit, our algorithm frequently discovered the perfect solution while it was reported in (Punch, Zongker, and Goodman 1996) that the perfect solution of this depth had never been found by GP. Thus, from this perspective, setting maximum depth at 5 makes the comparison more meaningful. This is the second reason why we chose this particular depth limit.

Table 6.3: Comparison of GP and GMPE on Royal Tree Problem.

Method	No.Eval/Gen	Gen	No. Eval./Run	Succeed	Speedup
GMPE	30	2574	77,220	50.9%	22.7
GP	3500	500	1,750,000	50%	

S	→	Add		|					
	→	Mul								
Add	→	Add	Add	|	Mul	→	Add	Add	|
	→	Mul	Mul	|		→	Mul	Mul	|
	→	Add	Mul	|		→	Add	Mul	|
	→	Mul	Add	|		→	Mul	Add	|
	→	x	Mul	|		→	x	Mul	|
	→	x	Add	|		→	x	Add	|
	→	Add	x	|		→	Add	x	|
	→	Mul	x			→	Mul	x		

Figure 6.9: Initial grammar for Max problem

6.5.2 Max Problem

6.5.2.1 Problem Description

The Max problem (Langdon and Poli 1997) has only one terminal x with value 0.5, and two nonterminals, *Add* (addition) and Mul (multiplication). The purpose is to find a tree with maximum fitness under some tree size constraint. In this experiment, we use the maximum depth as constraint, and set it to 6 (root is 0). The maximum fitness is 65536.

The initial grammar of GMPE for this problem can be found in Figure 6.9. The widely used grammar for problems of this kind resembles the grammars for regression problems. We choose this particular grammar to make a comparison with the GP counterpart feasible, and also to facilitate our use of the theoretical analyses of the Max problem available in GP literature, such as (Langdon and Poli 1997).

The results of the GP runs are taken from (Langdon and Poli 1997). Briefly, the GP settings are population size 200, generation size 500, tournament selection, 99.5% crossover, no mutation. In this experiment, the setting for GMPE is population size 60, truncation selection, selection rate 50%, $\alpha_i = 0.15$. Ten randomly generated

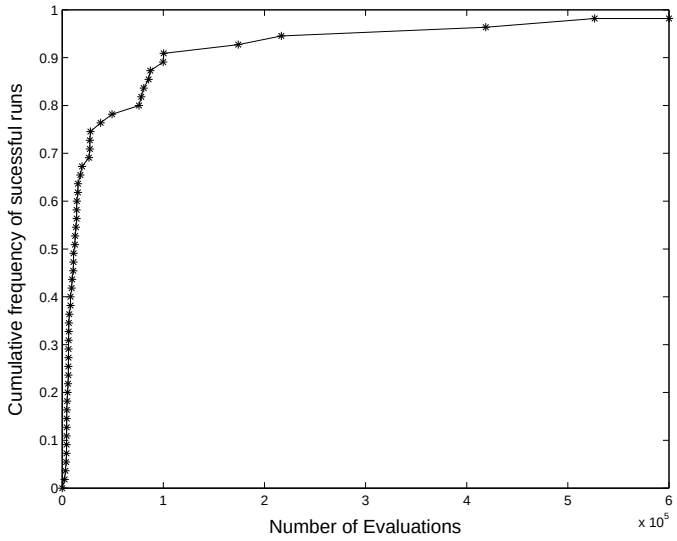


Figure 6.10: Cumulative frequency of successful runs of GMPE on Max Problem.

Population size	60
Generation	20,000
Maximum depth	7
Mutation rate	0.001
Prior observation counts α_i	0.15
Selection	Truncation
Total number of runs	55

Table 6.4: Parameter settings for GMPE on Max Problem

individuals are introduced in each generation. We conducted 55 runs. These settings for GMPE and GP are listed in Table 6.4 and 6.5.

6.5.2.2 Results

The cumulative frequency of successful runs is illustrated in Figure 6.10, where the horizontal axis is the number of generations, and the vertical axis is the percentage of successful runs. The comparison with GP is shown in Table 6.6. The majority of GMPE runs (over 90% of runs) found the perfect solution within 100,000 individual evaluations. In contrast, less than 60% of GP runs succeeded within the same

Population size	200
Generation	500
Maximum depth	7
Crossover rate	0.995
Mutation rate	0
Section	Tournament
Total number of runs	50

Table 6.5: Parameter settings for GP on Max Problem

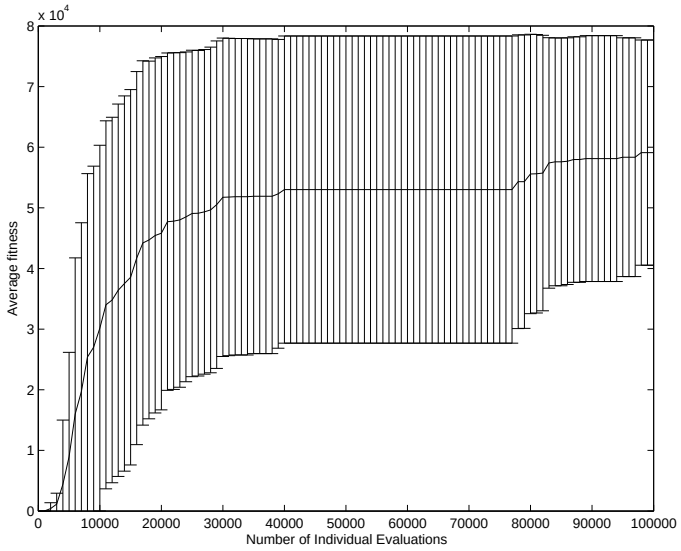


Figure 6.11: Average best fitness of all runs of GMPE on Max Problem.

number of individual evaluations. The average best fitnesses of all runs is plotted in Figure 6.11. We plot the fitness change up to 1×10^5 number of evaluations, i.e. the maximum on x-axis is 1×10^5, because the majority of runs succeed by then. Due to the similar reason as discussed in previous experiment, the error bar is large.

The $speedup_{60\%}$ defined in the previous subsection, is also presented in Table 6.6. According to this measure, GMPE is 7.4 times faster than GP, in terms of the number of individual evaluations, in this experiment.

It is worth mentioning that the Max problem has also been tested by a related

Table 6.6: Comparison of GP and GMPE on MAX problem

Method	No.Eval/Gen	Gen	No. Eval./Run	Succeed	Speedup
GMPE	30	453	13,590	60%	7.4
GP	200	500	100,000	$< 60\%$	

EDA-GP method – EDP (Yanai and Iba 2003). It reported that EDP significantly outperformed GP on this problem in terms of the number of individual evaluations. Although it is not possible to do a thorough comparison between EDP on GMPE given the limited results provided in (Yanai and Iba 2003), it appears that EDP performs better than GMPE – with greater reported average fitness and less standard deviation. The reason for this is not entirely clear at this stage and is certainly worth further investigation. One possibility is that it may be related to diversity. A small population was chosen in the GMPE experiment to reduce the computational cost, while also the truncation selection is known to produce high selection pressure. The combination of these two may lead to overly quick convergence. The very high variation showed in Figure 6.11 may also suggest that GMPE may heavily depend on random variation of its grammar (Section 6.4.4) to escape from local optimal after its quick convergence. However, no definite answer can be given without a thorough study. At this moment, we will leave this to future work.

6.5.3 Simple Symbolic Regression Problem

6.5.3.1 Problem Description

Symbolic regression problems are often used in the GP literature for evaluating the performance of an algorithm. In this experiment, we adopt a widely used function, originating from (Koza 1992). This problem has been used in evaluating PEEL in

Population size	40
Generation	250
Maximum depth	10
Mutation rate	0.001
Prior observation counts α_i	0.15
Selection	Truncation
Total number of runs	50

Table 6.7: Parameter settings for GMPE on simple symbolic regression problem $f(x) = x^4 + x^3 + x^2 + x$

Chapter 4. For convenience, we briefly re-state it here. The function is:

$$f(x) = x^4 + x^3 + x^2 + x$$

The fitness cases were sampled at 20 equidistant points in the interval [-1,1). Fitness was the sum of the absolute value of error. 5,000 program evaluations were set for both GMPE and GGGP. The maximum depth of an individual is 10. The grammar is the same as that in the previous PEEL experiment and is re-stated in Figure 6.2. The settings for GGGP were population size 100, generations 50, maximum depth 10, tournament size 3, crossover rate 0.9, mutation rate 0.1. These settings for GGGP are identical to those in the PEEL experiments. The other settings for GMPE are population size 40, maximum generations 250, $\alpha_i = 0.15$. Truncation selection is used in all the GMPE experiments in this section, i.e. the top half of the population is deterministically selected for grammar model learning. These major parameter settings for GMPE and GGGP are also listed in Table 6.7 and 6.8.

6.5.3.2 Results

The comparison of GMPE and GGGP can be found in Figure 6.12, where the horizontal axis is the number of generations and the vertical axis is the percentage of successful runs. GMPE found perfect solutions in 34 runs while GGGP found perfect

Population size	100
Generation	50
Maximum depth	10
Crossover rate	0.9
Mutation rate	0.1
Section	Tournament
Total number of runs	50

Table 6.8: Parameter settings for GGGP on simple symbolic regression problem $f(x) = x^4 + x^3 + x^2 + x$

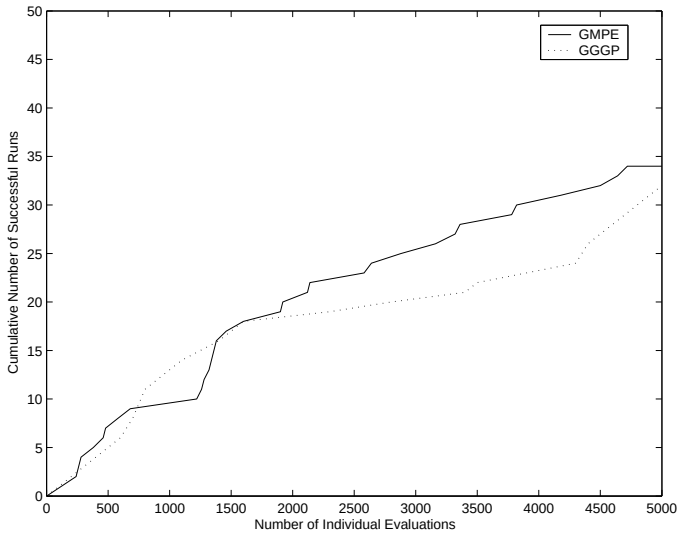


Figure 6.12: Cumulative frequency of success runs of GMPE and GGGP on problem $x^4 + x^3 + x^2 + x$.

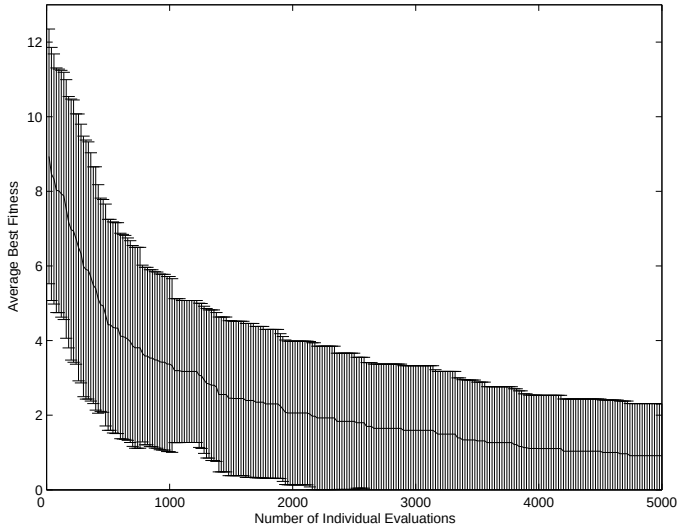


Figure 6.13: Average best fitness of all runs of GMPE on Problem $x^4 + x^3 + x^2 + x$.

solutions in 32 runs out of 50 runs. The average best fitnesses of all runs is plotted in Figure 6.13. On this problem GMPE performs equivalently to conventional GGGP.

As reported earlier, to understand the difficulty of this problem, we also conducted random testing. In total, 250,000 individuals (equivalent to 50 runs $\times$ 5000 individuals per run) were generated and evaluated. No solution was found. This confirms that both GGGP and GMPE significantly out-perform random search on this problem.

This symbolic regression problem was also chosen to present a comparison between GMPE and PEEL. The cumulative frequency of successful runs of PEEL, GMPE and GGGP is presented in Figure 6.14. On this particular problem, PEEL performs better than GMPE and GGGP is the worst. This result is not entirely consistent with what was expected. This problem seems to have some degree of regularity. Thus we expected GMPE, which can handle building blocks more effectively, would outperform PEEL. This interesting result is worth further investigation. One explanation may be that PEEL and GMPE may adopt different ways to solve this problem, which does not make use of the regularities perceived by humans. We will leave further investigation of this result for future work.

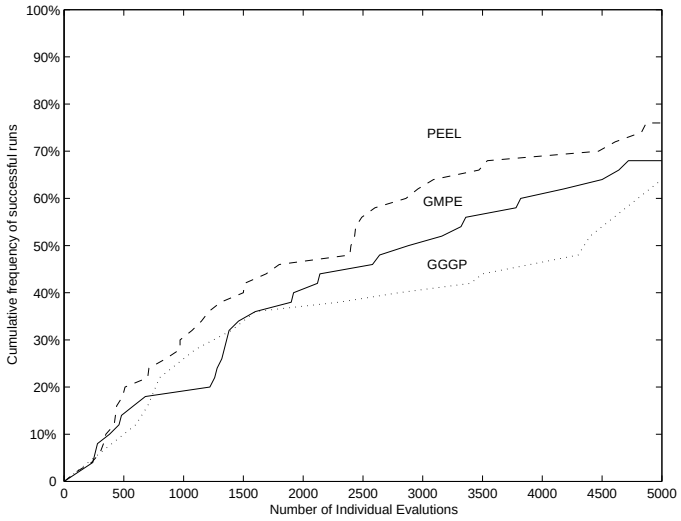


Figure 6.14: Cumulative frequency of successful runs of PEEL, GMPE and GGGP on regression problem $x^4 + x^3 + x^2 + x$.

6.5.4 Discussion

6.5.4.1 GMPE on benchmark problems

In this section, three benchmark problems were tested against GMPE. The Royal Tree Problem and Max Problem were chosen because they have clear and identifiable sub-solutions, i.e. the problem has sub-trees as partial solutions and its final solution can be obtained by growing or combining partial solutions. GMPE is designed to solve such problems constructively, i.e. discovering sub-solutions and then combining them. Therefore, we expected that GMPE should have superior performance on these two problems. These experiments verify that GMPE can efficiently solve problems of this kind. The Simple Symbolic Regression Problem has less clear structure. Although the most compact solutions for this problem may have some regularity, it is not clear whether this regularity helps solving the problem. Furthermore the problem has a large number of solutions which makes rigorous analysis extremely hard. It was chosen not for its help with analysis, but because it is a simple example of a large class of important general problems to which GP is often applied.

The Royal Tree Problem is a perfect demonstration of constructive problem solving. Given nonterminals A, B, C, D, E of arity 1, 2, 3, 4, 5 and terminal x, the fitness is deliberately designed so that the solution has to be constructed in a bottom-up fashion. The minimum partial solution (A x) has to be firstly identified, and then, based on this, a bigger partial solution (B (A x) (A x)) can be constructed, and so on. Note that due to the carefully chosen fitness function, the solution of the problem is constrained to follow this path, as previously explained. The superior performance of GMPE demonstrates that GMPE can solve problems with sub-solutions more efficiently than conventional GP.

The Max Problem has a similar property. One way of solving it is to construct a solution in bottom-up fashion. For example, a minimum partial solution is (*Add* (0.5 0.5)). By combining this minimum partial solution, we can then obtain bigger partial solutions (*Add* (*Add* (0.5 0.5))(*Add* (0.5 0.5))).

On the Simple Symbolic Regression Problem, GMPE does not show significant speedup. One of the reasons may be that the problem is so simple that both conventional GGGP and GMPE can work very well. More than 60% of runs succeed within only 5,000 individual evaluations for both GGGP and GMPE. Another reason may be that, because this problem has multiple solutions, these solutions are phenotypically different. The different solutions may drag the search into an oscillating state, where the search moves back and forth among potentially promising areas of the search space, but cannot effectively commit to any of these areas. This effect has been observed in conventional EDA (Hocaoglu and Sanderson 1997; Pelikan and Goldberg 2000; Larrañaga and Lozano 2001). Further investigation is needed in the case of EDA-GP.

The GMPE implementation of the PRODIGY framework is intended to promote building blocks of a particular form, namely complete sub-trees, with terminal symbols at the frontier. In GMPE, identical sub-trees can be easily identified through the merge operator, because of the MML measure. Merging corresponding rules from identical sub-trees only decreases the complexity of the grammar, while all

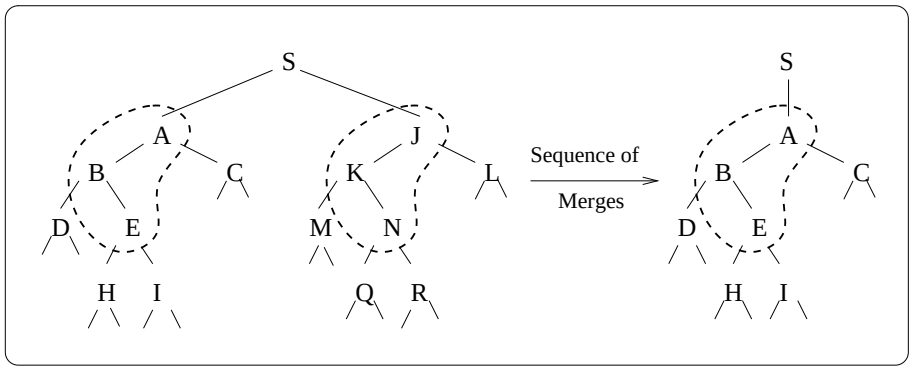


Figure 6.15: Complex building blocks which might not be able to be found by hill-climbing search

the other terms in the MML measure are kept constant. Thus it is possible to discover most or all of the identical sub-trees, using very simple search methods such as stochastic hill-climbing, or potentially even deterministic search methods. Since identical sub-trees (building blocks) occur multiple times in the population, either in the same individual or among individuals, the rules involved have greater frequencies. Rules with higher frequencies have greater chances to be preserved in GMPE. Hence rules which describe frequently-occurring sub-trees can be preserved well in GMPE. Then GMPE will conduct moderate search and variation around these identical sub-trees to discover more complex building blocks. This analysis of behaviour of GMPE is clearly validated by the Royal Tree Problem and Max problems we have studied.

6.5.4.2 Hill-climbing and Other Search Methods

The simple hill-climbing search method may limit the types of building blocks that can be discovered. Only merges which immediately improve the MML metric are accepted. However there are some forms of building blocks which can only be identified by a sequence of merges, where each single merge except the last does not improve the MML metric – only the entire sequence does. An example may help to clarify this. Suppose we have the following grammar, and S is the starting

symbol.

S	→	A	|	J
A	→	B	C	
B	→	D	E	
C	→	F	G	
E	→	H	I	
J	→	K	L	
K	→	M	N	
L	→	O	P	
N	→	Q	R	

This grammar can be represented as a tree, as shown in in Figure 6.15. Let us further assume that the ABE block on the left is identical to the JKN block on the right (in the sense of using identical productions from the initial grammar). Currently, these two blocks are represented by two sets of rules. Because they are identical, it is possible to use only one set of rules to represent both. Consequently, if these two rule sets are reduced to one set, the MML metric may improve, because the total number of rules would shrink dramatically – i.e. the complexity of the model would decrease – while the accuracy of the model may not change much. However, it is likely that this can only be achieved by a sequence of merges, where the early merges in the sequence may not improve the metric. In simple hill-climbing, whether to accept a merge or not is decided by measuring the improvement of the metric caused by that single merge. Therefore, although a sequence of merges may be beneficial, it is very unlikely to happen in simple hill climbing search, if the early merges of this sequence are of no benefit on their own.

In this example, in order to reduce the two sets of rules, four merges need to happen: $\text{merge}(D, M) = D$, $\text{merge}(H, Q) = H$, $\text{merge}(I, R) = I$, $\text{merge}(C, L) = C$. Each of these merges may not improve the metric, since by reducing the number of symbols they only reduce the complexity of the model slightly, but the accuracy of the model

may decrease. However, this sequence of merges transforms the original grammar into:

S → A | J
A → B C
B → D E
E → H I
J → K C
K → D N
N → H I

Given this grammar, merge$(E, N) = E$ becomes definitely beneficial (the grammar complexity decreases and accuracy does not change). After this merge, we obtain.

S → A | J
A → B C
B → D E
E → H I
J → K C
K → D E

For the same reason, merge$(B, K) = B$ could be subsequently imposed and the follow grammar would be obtained:

S → A | J
A → B C
B → D E
E → H I
J → B C

Similarly, merge$(A, J) = A$ could be subsequently imposed:

$$
\begin{array}{lcll}
S & \rightarrow & A & \\
A & \rightarrow & B & C \\
B & \rightarrow & D & E \\
E & \rightarrow & H & I
\end{array}
$$

This new grammar is also represented on the right of the Figure 6.15. This new grammar is much simpler than the original one and the accuracy might not change much. Hence, it may be the preferred model based on the MML metric but in this case, the current simplified stochastic hill-climbing search method would fail to find this simpler grammar.

However, there is no intrinsic difficulty in introducing a more sophisticated searching method under that same GMPE framework. There are several possibilities.

The current hill-climbing method only considers the next immediate step. Therefore, one of the options may be to add a number of steps of look-ahead. A sequence of multiple merges could be evaluated as a whole. However, the computational cost would increase exponentially with the number of steps. Thus it is not usually feasible to increase the number of steps substantially.

A second possibility is to introduce more stochasticity into the algorithm, by accepting deleterious merges with some probability. However this approach also suffers from an inability to handle complex merge sequences, due to the exponentially reducing probability of retaining longer deleterious merge sequences.

The most promising approach is to identify common sub-trees. If common sub-trees can be identified, merges around those structures will cause the common structures to be represented by the same set of rules. As illustrated in Figure 6.15, merges which significantly reduce the number of rules are very likely to improve the MML metric. There are several potential approaches to efficiently identifying common sub-structures. We could use a hash table to identify and count the common structures. Alternatively, tree compression algorithms, especially those based on the Lempel-Ziv algorithm (Chen and Reif 1996) may provide clues on how to efficiently detect

duplicate sub-structures.

6.6 Related work

The relationships between GMPE and other EDA-GP works reviewed in earlier chapters are presented in this section. The most important difference between GMPE and PIPE (Salustowicz and Schmidhuber 1997) – and other work based on the PIPE prototype tree , such as EDP (Yanai and Iba 2003) and ECGP (Sastry and Goldberg 2003) – is the probabilistic model used. The building blocks in PIPE-based work sit in fixed locations in the prototype tree, while in GMPE, because it uses a grammar for the probabilistic model, location references are removed, and thus building blocks occurring at a variety of locations can still be identified and preserved.

GMPE differentiates itself from most other work based on grammar models through its use of the standard form of SCFG (i.e. without depth and other positional constraints as in PEEL and in (Ratle and Sebag 2001)). More importantly, GMPE incorporates a full grammar-inference method, permitting it to infer any CFG-based probability structure, while works such as (Ratle and Sebag 2001; Abbass, Hoai, and McKay 2002) only infer the probability distribution, leaving the SCFG structure fixed. GMPE also differs from PEEL in these and other respects, as previously discussed.

One other subsequent work is quite closely related to GMPE (Bosman and de Jong 2004). The similarities lie in the use of SCFG as the probabilistic model, and in the inference of the complete SCFG from the population in each generation. However there are also major differences. Unusually for grammar-based GP approaches, this system does not treat the grammar derivation tree as the individual; rather, the individual is simply the string of terminals generated by the derivation tree. As a consequence, the individuals must be re-parsed in each generation, leading to severe restrictions on the grammars which can be handled (they must be non-ambiguous, with at most one recursive nonterminal). A further consequence is the unique and

highly specialised grammar inference method, radically different from that in GMPE. The approach used in (Bosman and de Jong 2004) differs from GMPE, and resembles other systems such as PEEL and (Ratle and Sebag 2001), in one other respect. It imposes tree depth as a reference for rule admission, which means the building blocks can only be shared at the same level of depth, unlike GMPE.

6.7 Conclusion

In this chapter, an approach implementing the PRODIGY framework – GMPE – is proposed and investigated. It employs a standard SCFG as its probabilistic model, and it infers both the structure and the probability distribution of the SCFG.

Because the SCFG model can better present, preserve and promote the common structures in superior individuals (usually called building blocks) than conventional GP, the GMPE method significantly outperforms conventional genetic programming on some of the standard benchmark problems we studied. More importantly, the experiments suggest that GMPE can solve problems constructively in a bottom-up way. In other words, GMPE can decompose a difficult problem into tractable smaller sub-problems.

Chapter 7

Swift GMPE

Following on from the GMPE approach proposed in Chapter 6, this chapter will discuss some implementation considerations, with an emphasis on reducing the computational cost of GMPE. A new implementation, Swift GMPE (sGMPE) will then be proposed.

This chapter is organised as follows. The time complexity of the original GMPE is given in Section 7.1. Then a modified version of GMPE, Swift GMPE (sGMPE) is proposed in Section 7.2. Some other aspects of sGMPE are discussed in Section 7.3 and Section7.4. The empirical studies are in Section 7.5. The last section is the conclusion.

7.1 Time complexity of GMPE

Ideally, it would be desirable to analyse the $O(n)$ complexity of GMPE with respect to problem size, and base simplifications of the GMPE algorithm on the analysis. Unfortunately the complicated nature of the GMPE algorithm renders this difficult. However although we are unable to estimate how much computation is needed to solve a problem of size n with a given probability, practically, it is still possible to

examine the cost of the individual steps of the GMPE algorithm and to reduce the computation time, which translates to reductions in the underlying time complexity of GMPE.

It is almost always the case that individual evaluation is the dominant factor in the computational cost of GMPE, as in other Evolutionary Computation approaches. However, this problem-specific cost is not related to the search method, such as GMPE, so it is not the focus of this research.

Aside from the cost of individual evaluation, the other major cost of the GMPE search method is the cost of grammar model learning. Recall the basic procedure of GMPE presented in Figure 6.1. Its grammar learning method is a hill-climbing search (see Section 6.4.2). The merge operator is used to variate the grammar model in seeking a good model. At each step, a number of possible merges have to be tried, and only the best merge will be accepted, manipulating the current grammar model to obtain the next one. For each possible merge, the MML metric has to be calculated to measure the quality of the merge. Ideally, at each step, all the possible merges would be tried, and only the best chosen. However, this is very computationally expensive, because the number of possible merges is quadratic in the number of non-terminals in the grammar. More specifically, given N non-terminals, there are $C_N^2 = \frac{N!}{(N-2)!2!} = \frac{N(N-1)}{2}$ possible merges. Since it is too computationally expensive to examine all the possible merges, we set n_t as the number of trials conducted before accepting the best one, i.e. the number of merges to be tried at each step of hill climbing search. In summary, the computational cost of GMPE, T, is

$$T = g \cdot (T_{evaluate} + n_{step} \cdot n_t \cdot (T_{merge} + T_{MML}) + T_{sample}) \tag{7.1}$$

where g is the number of generations of the GMPE algorithm, $T_{evaluate}$ is the cost of individual evaluation, n_{step} is the number of accepted merges (number of steps in hill climbing search), n_t is the number of trials before accepting a merge, T_{merge} is the cost of one merge operation, T_{MML} is the cost of calculation of the MML metric, and T_{sample} is the cost of sampling the learnt grammar model. In the Equation 7.1, g is predefined, $T_{evaluation}$ is not directly related to GMPE, n_{step} is not controllable

1. MML metric improvement merge.
2. If there are more than $p_n\%$ non-terminals left, keep doing 3, else terminate.
3. Random merge.

Figure 7.1: Grammar learning method of sGMPE

and T_{sample} is only a minor factor. Therefore, it is not hard to see that the terms in the middle $n_t \cdot (T_{merge} + T_{MML})$, the cost of learning the grammar model, is the place where we may seek to reduce the computational cost. T_{merge} and T_{MML} may be optimised separately. However, it would be more significant to reduce n_t. We are convinced that the key to reducing the computational cost is to reduce the number of trials of merges n_t, and consequently reduce the number of MML metric evaluations.

7.2 Algorithm

In this section, we discuss a modification of the GMPE algorithm to make it less computationally expensive, pursuing the idea of reducing the number of trials n_t. Reducing n_t does help to decrease the computational time, but it also reduces the probability of finding the optimal grammar, or lowers the quality of the grammar found. In order to avoid this dilemma, we replace the hill-climbing approach of grammar learning presented in Section 6.4.2 with the algorithm defined in Figure 7.1. We call this modified version of GMPE, Swift GMPE (sGMPE). On the problems studied, sGMPE typically completed a given number of evaluations in approximately half of the time required for GMPE. Since dedicated computing facilities required for comparison were not available for this research, an accurate resource comparison between these two systems is not be presented.

There are two major phases in this algorithm. The first phase concentrates on merges giving MML metric improvement, while the second incorporates random merges. There are simple criteria enabling us to select merges which definitely improve the

MML metric without actually calculating it. These merges happen in the first phase. The random merges randomly merge non-terminals. There is no calculation of MML metric involved. However if we remove calculation of the metric, we lose the indicator which determines when to stop merging. There are a number of ways to solve this problem. To keep the algorithm as simple as possible, we set a stop threshold p_n, which is the percentage of remaining non-terminal symbols with respect to the total number of symbols in the primitive grammar. This threshold is set empirically. Random merges generalise the grammar model, encouraging exploration. These two merges, MML metric improvement merge and random merge, will be further discussed in the following subsections.

Note that this algorithm does not introduce a new merge operator, all the merges must still be Initial-Grammar-Consistent merges (IGC merges) as presented in Section 6.4.2.2, the standard merge operator in GMPE. What is presented here is just a way of selecting IGC merges from among the possibilities.

7.2.1 Phase I: MML Metric Improvement Merge

For some merges, we can be certain the MML metric will improve without actually calculating the metric. In the phase of MML metric improvement merges, such merges are identified and executed.

The most readily identifiable merge that guarantees an improving MML metric is unification, defined in Section 6.4.2.2. Unifications merge two nonterminals which have an identical set of RHSs. In this phase, all the unifications are identified and executed using exhaustive search.

7.2.2 Phase II: Random Merge

The merges in the first phase guarantee to improve the MML metric. However these merges do not change the probability distribution of the grammar, and thus do not

generalise the grammar. In the second phase, ideally, we need to try all the possible merges, and choose that which most improves the MML metric. However, to reduce the cost, we instead do random merge, randomly choosing non-terminals to merge. We can justify this as follows.

Firstly, at this second stage, we need to generalise the grammar. All of the non-unification merges generalise the grammar, whether in GMPE or sGMPE. In GMPE, merges which generalise the grammar to a justifiable extent will be accepted, while in sGMPE every merge is accepted, with no calculation of the MML metric. The risk that may arise from a random merge is that it may over-generalise the grammar; this might not be as undesirable as it seems at first glance, for the following reasons.

Secondly, after the unifications of the first phase, the important building blocks (the building blocks occurring frequently in the population) are unlikely to be disrupted even by random merges. After the first phase of merges, the building blocks occurring many times in the population are represented by only one set of rules in the grammar model. Thus the total number of symbols representing a building block decreases and therefore its proportion of the total number of symbols in the grammar decreases. Consequently, even when choosing to merge non-terminals with uniform distribution, the probability that these building blocks are disrupted is much lower. This issue will be elaborated on in Section 7.3. Note that we will not make a conclusion on the overall effect, solely due to this probability change, on building blocks. The reason is that after unification, multiple copies of certain build block will be reduced to only one copy. Therefore, the consequence of disruption is more severe than before unification.

Thirdly, there should be only a small number of random merges, since too many random merges will lead to over-generalisation. Over-generalisation is very harmful in GMPE because it leads to inefficient learning. In the extreme, if over-generalisation balances the specialisation resulting from selection, the search will stagnate. Therefore, we avoid excessive random merges by setting an appropriate p_n.

Lastly, one-step look-ahead, as employed in the current implementation of GMPE, will miss those building blocks which need a sequence of merges. Even in the case of one-step look-ahead, some degree of randomness, such as accepting merges which do not improve the metric, might help alleviating this problem. Therefore, some degree of randomness may not have undesirable effects. This also relates to encouraging exploration, which will be discussed in Section 7.4

7.3 Disruption of Building Blocks by Random Merge

This section studies the degree of disruption caused by random merge, and in particular the change in disruptiveness caused by performing unifications.

7.3.1 Definition of Schema in GMPE

Definition 1 *A schema H for the GMPE is the derivation tree $A \stackrel{*}{\Rightarrow} \theta$, where $A \in N$ and $\theta \in (N \cup \Sigma)^*$.*

Similar definition is proposed in (Whigham 1995c). In our particular implementation of GMPE, the last condition $\theta \in (N \cup \Sigma)^*$ is restricted to $\theta \in \Sigma^*$, which means that the schema H is defined as a complete sub-tree with no open node, i.e. every leaf node is a terminal. This is due to simple hilling climbing search method used in GMPE, which has been discussed in Section 6.5.4.2. However, the discussion in this section, assuming the effective search method, is therefore applicable to the schema which is both complete and non-complete tree structure.

7.3.2 Disruption from Merge

In a given grammar, if there are i copies of schema H, there are in total $i * size(H)$ nodes representing this schema. If a merge happens among these nodes, i.e. inside

the schema, it may disrupt the schema. Thus, in a given grammar with i copies of schema H, before any unifications have been conducted, the probability of disruption of H by a random merge is

$$p_{before} = \frac{i * size(H)}{\sum_j^{POP} size(I_j)} \tag{7.2}$$

Actually, this probability should be better understood as the probability of non-beneficial merge, i.e. it is an upper bound of disruption, because some merges may happen inside the schema but not necessarily disrupt the schema.

We would like to know the change in the probability of these non-beneficial merges after unification. Unification is a merge which simply merges identical structures and thus improves the MML metric, as defined in Section 6.4.2.2.

The total number of occurrences of symbols in the SCFG model after unification of all instances of the schema $H$is

$$N = \sum_j^{POP} size(I_j) - size(H) \cdot (i-1) \tag{7.3}$$

where POP is the size of the population, $size(tree)$ is the size of the derivation tree $tree$, and I_j is the j-th individual of the population. Starting symbols in the individuals and the SCFG are ignored.

After unification of H, all the copies of H have been combined into one. Thus, the total number of nodes representing H is reduced from $i * size(H)$ to $size(H)$. Therefore, after unification, the probability of a schema H being disrupted by a random merge is

$$\begin{aligned} p_{after}(H) &= \frac{size(H)}{N} \\ &= \frac{size(H)}{\sum_j^{POP} size(I_j) - size(H) \cdot (i-1)} \end{aligned} \tag{7.4}$$

This probability is also an upper bound for the probability of disruption by merge in GMPE, using merges selected by the calculation of the MML metric. It is an upper bound because the above formula assumes that merges occur with uniform

distribution across all nodes (i.e. random merges), while actually merge occurs within schema H with lower probability because such merges are likely to worsen the MML metric.

We want to examine the relationship between p_{before} and p_{after}. We assume that the total number of symbols of a specific schema H is much less than the total number of symbols of the grammar, i.e. $\sum_j^{POP} size(I_j) >> size(H) \cdot i$. This should usually hold, especially in the early stages of search, when the schemas should be small. Given this assumption, P_{after} can be simplified by removing the last term in its denominator:

$$\begin{aligned} p_{after}(H) &= \frac{size(H)}{\sum_j^{POP} size(I_j) - size(H) \cdot (i-1)} \\ &\approx \frac{size(H)}{\sum_j^{POP} size(I_j)} \end{aligned} \tag{7.5}$$

Comparing Equation 7.2 and 7.5, it is easy to see that $p_{before} > p_{after}$, which implies that the probability of disrupting a schema significantly decreases after unification. It is also interesting to see that the more copies a schema has, the greater the decrease. This suggests that the more frequently a schema occurs in the training examples, the more likely it is to be preserved in the model. This is also consistent with our intuition.

7.4 Exploration Enhancement

Exploration is an important issue in EDA-GP. We have discussed in Section 3.4.4 that EDAs usually heavily emphasise exploitation and de-emphasise exploration. To ensure a high probability of finding the optimal solution, population sizing has been heavily studied in conventional EDA, but it is not easy to derive a similar theory in EDA-GP due to its different approach to solving problems. Although it is difficult to formulate a population sizing theory for GMPE, it is still possible to borrow techniques from EC to encourage exploration, in particular techniques for

maintaining diversity.

In the original GMPE proposed in Chapter 6, this issue is not considered. In sGMPE, this issue is addressed to some extent by random merge. From the perspective of exploration, random merge serves a similar purpose to mutation in EC, in introducing a certain amount of randomness to encourage exploration. We anticipate that, with this diversity enhancement mechanism, sGMPE might even outperform GMPE, especially given the small population sizes used in both systems. The empirical studies reported in section 7.5 support this view.

7.5 Empirical Studies

The three experiments from Chapter 6 were repeated with sGMPE. We observe that sGMPE performs no worse than GMPE, and in most experiments it outperforms GMPE. This validates that the simplification made in sGMPE, for finding good merges, is appropriate, and we hypothesise that the performance improvement may largely derive from the enhanced diversity.

The three experiments are reported in the following three subsections. All the experimental settings are identical to those in the Chapter 6 and are therefore omitted.

For each of the three experiments, we performed three sets of runs, each consisting of fifty runs. Keeping all other parameters unchanged, these three sets of runs have different stopping thresholds p_n. We set p_n to be $30\%, 60\%, 90\%$ respectively, to study the impact on the behaviour of sGMPE.

7.5.1 Royal Tree Problem

The performance of sGMPE on Royal Tree Problem is listed in Table 7.1. For comparison, the success rate of GMPE is also listed in right most column of the table. Within 6×10^5 individual evaluations, 90% runs of sGMPE with $p_n = 60\%$

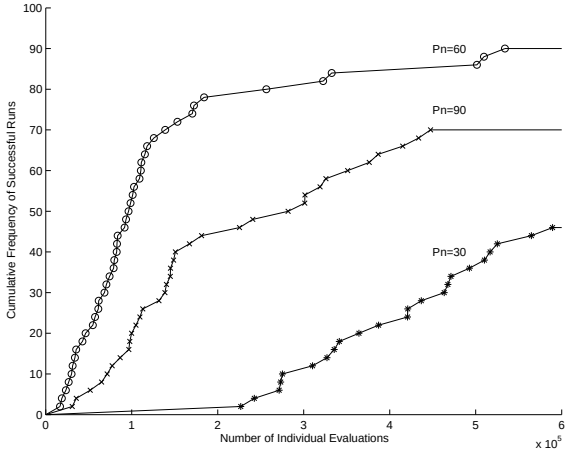


Figure 7.2: Cumulative frequency of successful runs of sGMPE on Royal Tree Problem.

succeed. This is comparable to the success rate of GMPE listed at the very right column. The runs with $p_n = 90\%$ do not perform as well while the runs with $p_n = 30\%$ are the worst.

p_n	30%	60%	90%	GMPE
Success Rate	46%	90%	70%	91%

Table 7.1: Success rate of different p_n of sGMPE on Royal Tree problem

The cumulative frequency of successful runs of sGMPE on this problem is plotted in Figure 7.2. As can be seen that runs with $p_n = 60\%$ constantly outperform the other two settings, i.e. $p_n = 90\%$ and $p_n = 30\%$

7.5.2 Max Problem

For the specified number of individual evaluations of 100,000, runs with $p_n = 30\%$ and $p_n = 60\%$ achieve a success rate of 100%, while runs with $p_n = 90\%$ have a success rate of 84%. These results are summarised in Table 7.2. For comparison, the success rate of GMPE is also listed in the table.

p_n	30%	60%	90%	GMPE
Success Rate	100%	100%	84%	89%

Table 7.2: Success rate of different p_n of sGMPE on Max problem

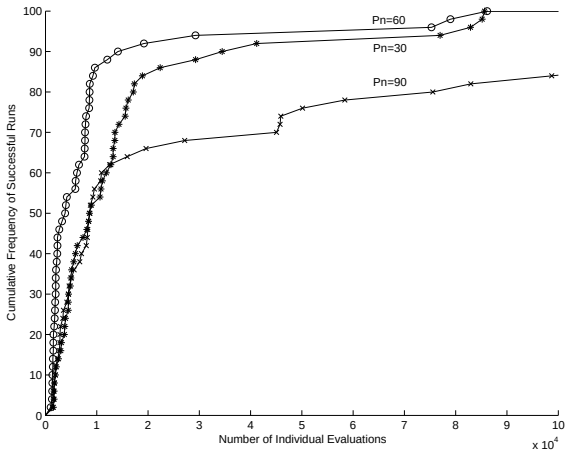


Figure 7.3: Cumulative frequency of successful runs of sGMPE on Max problem.

To have a better idea of the behaviours of these three sets of runs with different p_n, the cumulative frequency of successful runs are plotted in Figure 7.3. The x-axis is the total number of individual evaluations, and the y-axis is the percentage of successful runs. This figure has different scale from Figure 6.10. The x-axis in Figure 7.3 is only up to 1×10^5 individual evaluations because most of runs have succeeded by then. As can be seen, runs with $p_n = 60\%$ constantly outperform the others, while runs with $p_n = 30\%$ perform slightly worse although eventually managing to obtain the same success rate, i.e. 100%. Runs with $p_n = 90\%$ perform worst.

7.5.3 Simple Symbolic Regression Problem

The success rate of sGMPE on simple symbolic regression can be found in Table 7.3. Runs with $p_n = 60\%$ perform best with success rate 96%, while the other two are slightly worse, having the same success rate 86%. The last column is the success

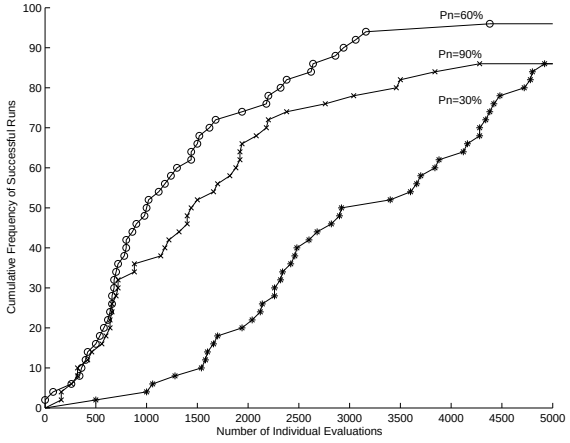


Figure 7.4: Cumulative frequency of successful runs of sGMPE on simple regression problem $x^4 + x^3 + x^2 + x$

rate of GMPE.

p_n	30%	60%	90%	GMPE
Success Rate	86%	96%	86%	68%

Table 7.3: Success rate of different p_n of sGMPE on symbolic regression problem $f(x) = x^4 + x^3 + x^2 + x$

The cumulative frequency of successful runs are plotted in Figure 7.4. It is clear that runs with $p_n = 60\%$ constantly outperforms the other two while runs $p_n = 90\%$ are better than runs with $p_n = 30\%$.

7.6 Conclusion

GMPE is a PRODIGY approach with solid theoretical support. However, it is very computationally expensive. To reduce the computational cost, a simplified version, sGMPE, is proposed in this chapter. From the theoretical and empirical analysis, we conclude that sGMPE is an appropriate approximation of the GMPE algorithm. Because the calculation of the MML metric is removed, the model obtained at

each generation might not be as optimal as the model from GMPE. However, the theoretical and empirical evidence suggests that this may not be as serious as it first appears. Furthermore, the random merge also introduces some random variation to the exploitation-intensive GMPE algorithm to encourage exploration. This explains the improved performance of sGMPE over GMPE.

We emphasise that this simplified version of GMPE, sGMPE, by no means undermines the importance of GMPE. The basic algorithm of sGMPE strictly follows GMPE. The grammar variation operator is still standard IGC merge. More importantly, although the calculation of the MML metric is removed from sGMPE, all the simplifications made in sGMPE are still under the guidance of this metric.

In the sGMPE algorithm, we expect runs with intermediate stopping thresholds, such as $p_n = 60\%$, to outperform runs with extreme values of this threshold, such as $p_n = 90\%$ and $p_n = 30\%$. Very high threshold values, such as $p_n = 90\%$, imply only a small number of random merges. Because the merges in the first phase do not generalise the grammar, insufficient random merges lead to an over-eager search, hindering performance. Furthermore, the threshold p_n is relative to the number of non-terminals of the primitive grammar. After the first phase of merging, this threshold may already have been closely approached. This further reduces the number of random merges happening. On the other hand, too many random merges make it very hard to accumulate search experience, because of over-generalisation. Thus, it is not surprising to see that sGMPE with $p_n = 30\%$ does not perform well either.

Chapter 8

Conclusion and Future Research

8.1 Conclusion

In this thesis, conventional EDA has been significantly extended in its handling of tree representations. This extension is important because tree representations, resembling GP tree individuals, provide a natural and expressive way to represent the solutions of a range of important problems.

This thesis presented a comprehensive survey of current research on EDA, with emphasis on EDA-GP, i.e. EDA with GP-style tree representation. The research reports on EDA-GP are scattered in a variety of conference proceedings, and we hope this thesis will help to bring the works of these researchers together, clarifying the relationships between different approaches and facilitating communications between those interested in the field.

Extending conventional EDA to tree representations is a non-trivial step because of the complexity of the tree structure. To accomplish it, this thesis proposes a framework, Program Distribution Estimation with Grammar Models (PRODIGY). The core of this framework is a Stochastic Context-free Grammar model (SCFG). We have argued that SCFG can model position-independent common structures, which

are believed important in GP. Within this framework, two approaches with different emphases have been proposed. The first is based on a constrained SCFG model, using a depth constraint to simplify the grammar learning. This approach trades off some level of effectiveness for computational efficiency. The second introduces an elegant grammar learning method from the field of Natural Language Processing (NLP). Both approaches were benchmarked on a range of problems, showing interesting characteristics. An MML metric for grammar learning was also derived in the thesis. Although it was derived for the purpose of grammar learning in the context of PRODIGY, it is generally applicable for other grammar learning applications. It is also relevant to probabilistic graph model learning.

We draw the following general conclusions from this research. EDA-GP is a promising research direction, and grammar models have some appealing properties as models for EDA-GP. Thus we envisage further studies under the PRODIGY framework in the near future. Grammar learning, also known as Grammar Inference, is a vital part of EDA-GP using grammar models. It has been studied for many decades in the field of Natural Language Processing (NLP). NLP thus provides a fertile resource of grammar learning methods which warrant further investigation for their suitability for EDA-GP. We see this work as just a start in the PRODIGY direction. Some future research issues will be discussed in the next section.

8.2 Future Research

The past three years have witnessed a surge of new research into EDA-GP, being part of the recent increasing interest in EDA as a whole. EDA-GP is both practically relevant and intellectually challenging. Our research provides some starting points rather than a complete solution to the problem. This section presents some future research issues, both research questions about the PRODIGY framework and alternative perspectives to explore this problem. Some of these discussions are also relevant to EDA-GP in general.

8.2.1 Grammar Learning Method

PRODIGY is a general framework for EDA-GP. Many kinds of grammar learning methods can be plugged into it. We investigated two possible implementations of PRODIGY, PEEL and GMPE, in this thesis. It is worth investigating what are the pros and cons of other grammar learning methods in the context of PRODIGY. Even within our implementation, hill-climbing is only one of the possible search methods, and a range of alternative search methods remain to be explored.

8.2.2 Knowledge Extraction, Reuse and Incorporation

In PRODIGY, and EDA in general, a well-formed model is used to encode the knowledge accumulated in the course of the search. Extraction and study of the knowledge embedded in a model is an important research direction.

The EDA model, being a probabilistic model, is more interpretable than a population. More importantly, even if it turns out to be too complex for human comprehension, it may still be possible to extract some form of knowledge that can be used by the EDA algorithm in other applications. This leads to one possible way of incorporating knowledge. If the knowledge can be extracted from other EDA models in similar applications, and represented in a standard format, such as symbolic logic, it is then perfectly possible to exploit this information. While this has not yet happened in EDA-GP, related fields, such as Bayesian Networks and Artificial Neural Networks, have explored the possibility of such knowledge extraction and reuse (d'Avila Garcez, Broda, and Gabbay 2001; Towell and Shavlik 1993; Haddawy 1994; Le, Bah, and Ungar 2004).

The second way to incorporate knowledge is to make use of prior human knowledge. For real world problems, domain experts usually have some level of background knowledge. The grammar model used in PRODIGY provides a mechanism to incorporate background knowledge. By incorporating domain dependent knowledge

into PRODIGY, we hope to achieve superior performance. In more general EDA, it has been shown in (Baluja 2002), that incorporating prior knowledge may improve EDA performance.

8.2.3 Parsimony Pressure and Noisy Data

PRODIGY has a built-in Occam's razor which causes small individuals to be preferred. It is well known in machine learning that less complicated models generalise better than more complicated ones. Hence, we expect PRODIGY to perform particularly well in learning from noisy data. This is very important for real world applications where training data are noisy and generalisation is critical. Preliminary research in this direction is promising.

8.2.4 Incremental Learning

This issue has been discussed in 3.4.2. It would be desirable to update the model in each generation rather than learn a new model from scratch. One possible way to do this may be to use the model of the previous generation as a prior, and then incorporate more information from the fitness distribution of the current generation into the model. This approach fits well into a Bayesian framework.

8.2.5 Making Use of Negative Examples

In PRODIGY, only the positive examples, i.e. high fitness individuals, are used for inferring the SCFG model. However, there is no obvious reason that the negative examples, i.e. low fitness individuals, cannot be used, although in the field of EDA, we are only aware of one work, LEM (Michalski 2000), which uses both.

The high fitness individuals may indicate promising search areas while the low fitness ones may tell us the less potential areas which we need to avoid. Use of both

examples may improve the efficiency of model inference and therefore is worth further investigation.

8.2.6 Developing Theory

PRODIGY and EDA eschew genetic operators and maintain a well-structured model. This provides an opportunity to develop a series of theories to characterise EDA. In conventional EC this has proven to be very difficult because of the highly complex behaviours of genetic operators and the dynamics of the population. In EDA, these are greatly simplified, and there has been limited progress in theoretical analysis, for example some EDA algorithms have been based on sound theory (Mühlenbein and Mahnig 1999) and there are interesting attempts at characterising conventional EDA (Pelikan 2002). So far, there have been no theoretical analyses of EDA-GP. For EDA-GP approaches using a fixed model, there seem no insuperable difficulties in extending analyses of conventional EDA. EDA-GP approaches incorporating model learning are clearly more complex to analyse because of the interaction between model learning and probability learning. The well-known equivalence between MML measures and probability provide some reason for hope that it might be possible to study these two components within a common framework.

Bibliography

Abbass, H. A., N. X. Hoai, and R. I. McKay (2002). AntTAG: A new method to compose computer programs using colonies of ants. In *The IEEE Congress on Evolutionary Computation*, pp. 1654–1659.

Allison, L. (2003). Multistate and multinomial distributions. http://www.csse.monash.edu.au/ lloyd/tildeMML/Discrete/Multistate.html.

Andrews, G. (1984). *The Theory of Partitions*. Cambridge: Cambridge University Press.

Angeline, P. J. (1994). Genetic programming and emergent intelligence. In K. E. Kinnear, Jr. (Ed.), *Advances in Genetic Programming*, Chapter 4, pp. 75–98. MIT Press.

Angeline, P. J. (1995). Two self adaptive crossover operations for genetic programming. In *Advances in Genetic Programming II*, pp. 89–110. MIT Press.

Angeline, P. J. and J. B. Pollack (1994). Coevolving high-level representations. In C. G. Langton (Ed.), *Artificial Life III*, Volume XVII, Santa Fe, New Mexico, pp. 55–71. Addison-Wesley.

Angluin, D. (1990). Negative results for equivalence queries. *Mach. Learn. 5*(2), 121–150.

Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA.

Baluja, S. (2002). Using a priori knowledge to create probabilistic models for optimization. *International Journal of Approximate Reasoning 31*(3), 193–220.

Baluja, S. and S. Davies (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proc. 1997 International Conference on Machine Learning*. Also Available as Tech Report: CMUCS -97-107.

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

Bengoetxea, E., P. Larrañaga, I. Bloch, A. Perchant, and C. Boeres (2002). Inexact graph matching by means of estimation of distribution algorithms. *Pattern Recognition 35*(12), 2867–2880.

Blanco, R., I. Inza, and P. Larrañaga (2003). Learning bayesian networks in the space of structures by estimation of distribution algorithms. *International Journal of Intelligent Systems 18*(2), 205 – 220.

Bleuler, S., M. Brack, L. Thiele, and E. Zitzler (2001, 27-30). Multiobjective genetic programming: Reducing bloat using SPEA2. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, pp. 536–543. IEEE Press.

Bockhorst, J. and M. Craven (2001). Refining the structure of a stochastic context-free grammar. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-2001)*.

Bonabeau, E., M. Dorigo, and T. Theraulaz (1999). *From Natural to Artificial Swarm Intelligence*. New York: Oxford University Press.

Boryczka, M. and Z. J. Czech (2002, 9-13 July). Solving approximation problems by ant colony programming. 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 (Eds.), *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, New York, pp. 133. Morgan Kaufmann Publishers.

Bosman, P. and D. Thierens (1999). An algorithmic framework for density estimation based evolutionary algorithms. Technical Report Technical Report UU-CS-1999-46, Utrecht University.

Bosman, P. A. N. and E. D. de Jong (2004, June). Grammar transformations in an eda for genetic programming. In *Special session: OBUPM - Optimization by Building and Using Probabilistic Models, GECCO*, Seattle, Washington, USA.

Buntine, W. (1990, February). *A Theory of Learning Classification Rules*. Ph. D. thesis, School of Computing Science in the University of Technology, Sydney.

Charniak, E. (1993). *Statistical Language Learning*. Cambridge, Massachusetts: MIT Press.

Chen, S. and J. Reif (1996, March). Efficient lossless compression of trees and graphs. In J. A. Storer and M. Cohn (Eds.), *Proceedings of IEEE Data Compression Conference*, Snowbird, UT USA, pp. 428.

Chen, S. F. (1995). Bayesian grammar induction for language modeling. In *Meeting of the Association for Computational Linguistics*, pp. 228–235.

Chen, S. F. (1996). *Buidling Probabilistic Models for Natural Language*. Ph. D. thesis, Harvard University.

Cramer, N. L. (1985, 24-26 July). A representation for the adaptive generation of simple sequential programs. In J. J. Grefenstette (Ed.), *Proceedings of an International Conference on Genetic Algorithms and the Applications*, Carnegie-Mellon University, Pittsburgh, PA, USA, pp. 183–187.

d'Avila Garcez, A. S., K. Broda, and D. M. Gabbay (2001). Symbolic knowledge extraction from trained neural networks: a sound approach. *Artif. Intell. 125*(1-2), 155–207.

de Bonet, J. S., C. L. Isbell, Jr., and P. Viola (1997). MIMIC: Finding optima by

estimating probability densities. In M. C. Mozer, M. I. Jordan, and T. Petsche (Eds.), *Advances in Neural Information Processing Systems*, Volume 9, pp. 424. The MIT Press.

D'haeseleer, P. (1994, 27-29). Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, Volume 1, Orlando, Florida, USA, pp. 256–261. IEEE Press.

Etxeberria, R. and P. Larrañaga (1999). Global optimization with bayesian networks. In *Second Symposium on Artifcial Intelligence(CIMAF-99)*, Cuba, pp. 332–339.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

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

Green, J., J. L. Whalley, and C. G. Johnson (2004, September). Automatic programming with ant colony optimization. In M. Withall and C. Hinde (Eds.), *Proceedings of the 2004 UK Workshop on Computational Intelligence*, pp. 70–77. Loughborough University.

Gruau, F. (1996). On using syntactic constraints with genetic programming. In P. J. Angeline and K. E. Kinnear, Jr. (Eds.), *Advances in Genetic Programming 2*, Chapter 19, pp. 377–394. Cambridge, MA, USA: MIT Press.

Grunwald, P. (1994). A minimum description length approach to grammar inference. In *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language*, Volume 1004 of *Lecture Notes in AI*, pp. 203–216. Berlin: Springer Verlag.

Haddawy, P. (1994). Generating bayesian networks from probability logic knowledge bases. In *Proceedings of the Tenth Conference on Uncertainty in Articial Intelligence*.

Harik, G. (1999). Linkage learning via probabilistic modeling in the ECGA. Technical Report IlliGAL Report No. 99010, University of Illinois at Urbana-

Champaign.

Harik, G. R. and D. E. Goldberg (1997). Learning linkage. In R. K. Belew and M. D. Vose (Eds.), *Foundations of Genetic Algorithms 4*, pp. 247–262. San Francisco, CA: Morgan Kaufmann.

Harik, G. R., F. G. Lobo, and D. E. Goldberg (1999, November). The compact genetic algorithm. *IEEE Transaction on Evolutionary Computation 3*(4), 287–297.

Hocaoglu, C. and A. C. Sanderson (1997). Multimodal function optimization using minimal representation size clustering and its application to planning multipaths. *Evolutionary Computation 5*(1), 81 – 104.

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

Iba, H. and H. de Garis (1996). Extending genetic programming with recombinative guidance. In *Advances in Genetic Programming II*, pp. 69–88. MIT Press.

Iba, H., H. de Garis, and T. Sato (1994). Genetic programming using a minimum description length principle. In K. E. Kinnear, Jr. (Ed.), *Advances in Genetic Programming*, pp. 265–284. MIT Press.

Jäske, H. (1996, August). One-step-ahead prediction of sunspots with genetic programming. In *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)*, Vaasa Finland.

Joshi, A. K., L. S. Levy, and M. Takahashi (1975). Tree adjunct grammars. *Journal of Computer and System Sciences 10*(1), 136–163,.

Keber, C. and M. G. Schuster (2002). Option valuation with generalized ant programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 74–81. Morgan Kaufmann Publishers Inc.

Keller, B. and R. Lutz (1997). Evolving stochastic context-free grammars from examples using a minimum description length principle. In *Proceedings of*

Workshop on Automata Induction Grammatical Inference and Language Acquisition, ICML-97, Nashville, Tennessee.

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

Langdon, W. B. and R. Poli (1997, 13-16 July). An analysis of the MAX problem in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford University, CA, USA, pp. 222–230. Morgan Kaufmann.

Langdon, W. B. and R. Poli (2002). *Foundations of Genetic Programming*. Springer-Verlag.

Lari, K. and S. J. Young (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language 4*, 35–56.

Lari, K. and S. J. Young (1991). Applications of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language 5*, 237–257.

Larrañaga, P. and J. A. Lozano (2001). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academis Publishers.

Le, P. P., A. Bah, and L. H. Ungar (2004). Using prior knowledge to improve genetic network reconstruction from microarray data. *Silico Biology 4*(27).

Manning, C. and H. Schütze (1999, May). *Foundations of Statistical Natural Language Processing*. Cambridge, MA:: MIT Press.

Michalski, R. S. (1983). A theory and methodology of inductive learning. *Artificial Intelligence 20*(2), 111–161.

Michalski, R. S. (2000). Learnable evolution model: Evolutionary processes guided by machine learning. *Machine Learning 38*, 9–40.

Montana, D. J. (1993, 7 May). Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street,

Cambridge, MA 02138, USA.

Müehlenbein, H. and G. Paaß(1996). From recombination of genes to the estimation of distributions i.binary parameters. In *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature, PPSN IV*, pp. 178–187. Springer.

Mühlenbein, H. and T. Mahnig (1999). The factorized distribution algorithm for additively decompressed functions. In *1999 Congress on Evolutionary Computation*, Piscataway, NJ, pp. 752–759. IEEE Service Center.

Nikolaev, N. Y. and H. Iba (2001, August). Regularization approach to inductive genetic programming. *IEEE Transactions on Evolutionary Computing 54*(4), 359–375.

O'Reilly, U.-M. and F. Oppacher (1995). The troubling aspects of a building block hypothesis for genetic programming. In L. D. Whitley and M. D. Vose (Eds.), *Foundations of Genetic Algorithms 3*, Estes Park, Colorado, USA, pp. 73–88. Morgan Kaufmann.

Osborne, M. (1999). DCG induction using MDL and parsed corpora. In J. Cussens (Ed.), *Proceedings of the 1st Workshop on Learning Language in Logic*, Bled, Slovenia, pp. 63–71.

Paul, T. K. and H. Iba (2004). Identification of informative genes for molecular classification using probabilistic model building genetic algorithm. In e. a. Kalyanmoy Deb (Ed.), *Proceedings of Genetic and Evolutionary Computation (GECCO 2004) (Lecture Notes in Computer Science)*, Volume 3102, Seattle, USA, pp. 414 – 425.

Pelikan, M. (2002). *Bayesian optimization algorithm: From single level to hierarchy*. Ph. D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL. Also IlliGAL Report No. 2002023.

Pelikan, M. and D. E. Goldberg (2000, 16-20). Genetic algorithms, clustering, and the breaking of symmetry. In H.-P. S. Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo (Ed.), *Parallel*

Problem Solving from Nature - PPSN VI 6th International Conference, Paris, France. Springer Verlag.

Pelikan, M., D. E. Goldberg, and E. Cantú-Paz (1999, 13-17). BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Volume I, Orlando, FL, pp. 525–532. Morgan Kaufmann Publishers, San Fransisco, CA.

Pelikan, M., D. E. Goldberg, and E. Cantú-Paz (2000). Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation 8*(3), 311–341. Also IlliGAL Report No. 98013.

Pelikan, M., D. E. Goldberg, and F. Lobo (1999, Sept). A survey of optimization by building and using probabilistic models. Technical Report IlliGAL Report No. 99018, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

Pelikan, M., D. E. Goldberg, J. Ocenasek, and S. Trebst (2003). Robust and scalable black-box optimization, hierarchy, and ising spin glasses. IlliGAL Report No. 2003019, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL.

Pelikan, M. and H. Mühlenbein (1999). The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. K. Chawdhry (Eds.), *Advances in Soft Computing - Engineering Design and Manufacturing*, London, pp. 521–535. Springer-Verlag.

Pereira, F. and Y. Schabes (1992). Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th conference on Association for Computational Linguistics*, pp. 128–135. Association for Computational Linguistics.

Prusinkiewicz, P. and A. Lindenmayer (1990). *The Algorithmic Beauty of Plants*. Springer.

Punch, W. F., D. Zongker, and E. D. Goodman (1996). The royal tree problem,

a benchmark for single and multiple population genetic programming. In P. J. Angeline and K. E. Kinnear, Jr. (Eds.), *Advances in Genetic Programming 2*, Chapter 15, pp. 299–316. Cambridge, MA, USA: MIT Press.

Ratle, A. and M. Sebag (2001, October 29-31). Avoiding the bloat with probabilistic grammar-guided genetic programming. In P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, and M. Schoenauer (Eds.), *Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001*, Volume 2310 of *LNCS*, Creusot, France, pp. 255–266. Springer Verlag.

Rissanen, J. (1989). *Stochastic Complexity in Statistical Inquiry*. Singapore: World Scientific Press.

Rojas, S. A. and P. J. Bentley (2004, 26 July). A grid-based ant colony system for automatic program synthesis. In M. Keijzer (Ed.), *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA.

Rosca, J. P. (1997a, 13-16 July). Analysis of complexity drift in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford University, CA, USA, pp. 286–294. Morgan Kaufmann.

Rosca, J. P. (1997b, 13-16). Analysis of complexity drift in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford University, CA, USA, pp. 286–294. Morgan Kaufmann.

Rosca, J. P. and D. H. Ballard (1994, February). Genetic programming with adaptive representations. Technical Report TR 489, University of Rochester, Computer Science Department, Rochester, NY, USA.

Ross, B. J. (2001). Logic-based genetic programming with definite clause translation grammars. *New Generation Computing 19*(4), 313–337.

Roux, O. and C. Fonlupt (2000). Ant programming: or how to ants for automatic

programming. In *Proceedings of the Second International Conference on Ant Algorithms (ANTS2000)*, Belgium.

Sagarna, R. and J. Lozano (2004). On the performance of estimation of distribution algorithms applied to software testing. *Applied Artificial Intelligence In press.*

Sakakibara, Y. (1990). Learning contextfree grammars from structural data in polynomial time. *Theoretical Computer Science 76*, 223–242.

Sakakibara, Y. (1992). Efficient learning of context-free grammars from positive structural examples. *Inf. Comput. 97*(1), 23–60.

Sakakibara, Y. (1997). Recent advances of grammatical inference. *Theor. Comput. Sci. 185*(1), 15–45.

Salustowicz, R. P. and J. Schmidhuber (1997). Probabilistic incremental program evolution. *Evolutionary Computation 5*(2), 123–141.

Sastry, K. and D. E. Goldberg (2003). Probabilistic model building and competent genetic programming. In R. L. Riolo and B. Worzel (Eds.), *Genetic Programming Theory and Practise*, Chapter 13, pp. 205–220. Kluwer.

Schmidhuber, J. (1987, 14 May). Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany.

Shan, Y., R. McKay, R. Baxter, H. Abbass, D. Essam, and H. Nguyen. (2004, June). Grammar model-based program evolution. In *Proceedings of The Congress on Evolutionary Computation*, Portland, USA. IEEE.

Shan, Y., R. I. McKay, H. A. Abbass, and D. Essam (2003, Dec). Program evolution with explicit learning: a new framework for program automatic synthesis. In *Proceedings of 2003 Congress on Evolutionary Computation*, Canberra, Australia. University College, University of New South Wales, Australia.

Spears, W. M., K. A. D. Jong, T. Bäck, D. B. Fogel, and H. de Garis (1993). An overview of evolutionary computation. In P. Brazdil (Ed.), *Machine Learning:*

ECML-93, European Conference on Machine Learning, Vienna, Austria, pp. 442–459. Springer.

Stolcke, A. (1994). *Bayesian Learning of Probabilistic Language Models*. Ph. D. thesis, University of California, Berkeley, CA.

Stutzle, T. and H. Hoos (1997, April). Max-min ant system and local search for the traveling salesman problem. In *Proceedings of IEEE International Conference on Evolutionary Computation*, Indianapolis, IN, US, pp. 309–314.

Tanev, I. (2004, June). Implications of incorporating learning probabilistic context-sensitive grammar in genetic programming on evolvability of adaptive locomotion gaits of snakebot. In *Proceedings of GECCO 2004*, eattle, Washington, USA.

Towell, G. G. and J. W. Shavlik (1993). Extracting refined rules from knowledge-based neural networks. *Mach. Learn. 13*(1), 71–101.

Wallace, C. S. and D. M. Boulton (1968). An information measure for classification. *The Computer Journal 11*(2), 185–194.

Wallace, C. S. and D. L. Dowe (1999). Minimum message length and kolmogorov complexity. *The Computer Journal 42*(4), 270–283.

Wallace, C. S. and P. R. Freeman (1987). Estimation and inference by compact coding. *Journal of the Royal Statistical Society. Series B (Methodological) 49*(3), 240–265.

Weigend, A. S., B. A. Huberman, and D. E. Rumelhart (1992). Predicting sunspots and exchange rates with connectionist networks. In *Nonlinear Modelling and Forecasting*, pp. 395–432. Addison-Wesley.

Whigham, P. (1995a, September). Inductive bias and genetic programming. In *Proceedings of First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 461–466. UK:IEE.

Whigham, P. A. (1995b, 9 July). Grammatically-based genetic programming. In J. P. Rosca (Ed.), *Proceedings of the Workshop on Genetic Programming:*

From Theory to Real-World Applications, Tahoe City, California, USA, pp. 33–41.

Whigham, P. A. (1995c, 29 November - 1 December). A schema theorem for context-free grammars. In *1995 IEEE Conference on Evolutionary Computation*, Volume 1, Perth, Australia, pp. 178–181. IEEE Press.

Wong, M. L. and K. S. Leung (1995, October). Genetic logic programming and applications. *IEEE Expert 10*(5), 68–76.

Yanai, K. and H. Iba (2003, Dec). Estimation of distribution programming based on bayesian network. In *Proceedings of Congress on Evolutionary Computation*, Canberra, Australia, pp. 1618–1625.

Zhang, B.-T. and H. Mühlenbein (1995). Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation 3*(1), 17–38.

Zlochin, M., M. Birattari, N. Meuleau, and M. Dorigo (2004). Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research 131*, 373–395.