JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von
Bogdan Burlacu

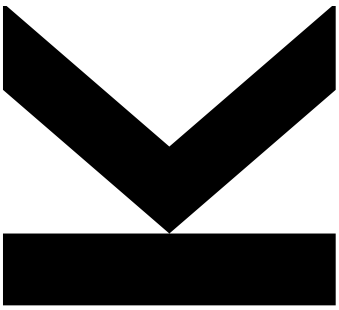
Angefertigt am
Institute for Formal Models and Verification

Erstbeurteiler
FH-Prof. PD DI Dr. Michael Affenzeller

Zweitbeurteiler
a.Univ.-Prof. Dr. Josef Küng

August 2017

Tracing of Evolutionary Search Trajectories in Complex Hypothesis Spaces



Dissertation

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

im Doktoratsstudium der

Technischen Wissenschaften

Acknowledgements

First and foremost, I would like to thank Prof. Michael Affenzeller for the opportunity to pursue a PhD within the Heuristics and Evolutionary Algorithms Laboratory (HEAL) at the University of Applied Sciences Upper Austria and for the continuous guidance and friendship. The work described in this thesis would not have been possible without the funding provided by the International PhD program in Informatics Hagenberg, offered as a specialization of the computer science PhD program at the Johannes Kepler University Linz (Austria).

I would also like to extend my thanks and gratitude to my colleagues from HEAL who supported me during this time with many insights, criticisms, discussions and development advice: Michael Kommenda, Gabriel Kronberger, Stephan Winkler, Andreas Beham, Stefan Vonolfen, and the whole Heuristiclab team.

Abstract

Understanding the internal functioning of evolutionary algorithms is an essential requirement for improving their performance and reliability. Increased computational resources available in current mainstream computers make it possible for new previously infeasible research directions to be explored. Therefore, a comprehensive theoretical analysis of their mechanisms and dynamics using modern tools becomes possible. Recent algorithmic achievements like offspring selection in combination with linear scaling have enabled genetic programming (GP) to achieve high quality results in system identification in less than 50 generations using populations of only several hundred individuals. Therefore, the active gene pool of evolutionary search remains manageable and may be subjected to new theoretical investigations closely related to genetic programming schema theories, building block hypotheses and bloat theories.

Genetic algorithms emulate emergent systems in which complex patterns are formed from an initially simple and random pool of elementary structures. In GP, complexity emerges under the influence of stabilizing selection which preserves the useful genetic variation created by recombination and mutation. The mapping between the structures used for solution representation and the ones used for the evaluation of fitness has a major influence on algorithm behavior. Population-wide effects concerning building blocks, genetic diversity and bloat can be conceptually seen as results of the complex interaction between phenotypic operators (selection) and genotypic operators (mutation and recombination). This coupling known as the "variation-selection loop" is the main engine for GP emergent behavior and constitutes the main topic of this research.

This thesis aims to provide a unified theoretical framework which can explain GP evolution. To this end, it explores the way in which the genotype-phenotype map, in relation with the evolutionary operators (selection, recombination, mutation) determines algorithmic behaviour. As the title suggests, the main contribution consists of a novel "tracing" framework that makes it possible to determine the exact patterns of building block and gene propagation through the generations and the way smaller elements are gradually assembled into more complex structures by the evolutionary algorithm.

Zusammenfassung

Um die Leistung und Zuverlässigkeit von evolutionären Algorithmen zu verbessern, ist es notwendig deren interne Funktionsweise zu verstehen. Die hohe Rechenleistung aktueller Mainstream-Computer erlaubt neue Forschungsrichtungen zu erkunden, welche früher, aufgrund fehlender Rechenleistung, nicht möglich waren. Für evolutionäre Algorithmen wird es dadurch möglich, deren interne Mechaniken und Dynamiken umfangreich zu analysieren. Aktuelle algorithmische Fortschritte, wie Nachkommensselektion (Offspring Selection) in Kombination mit linearer Skalierung, ermöglicht Genetischer Programmierung (GP) hochqualitative Ergebnisse in der Systemidentifikation zu erreichen, in weniger als 50 Generationen bei einer Populationsgröße von nur wenigen hunderten Individuen. Der dadurch überschaubare aktive Genpool der evolutionären Suche ermöglicht neue theoretische Untersuchungen zum GP Schema Theorem, zur Baustein Hypothese und zu Bloat-Theorien.

Genetische Algorithmen emulieren emergente Systeme in denen komplexe Muster geformt werden, basierend auf einer initialen, zufällig generiertem Menge an elementaren Strukturen. In GP entsteht die Komplexität durch den Einfluss der stabilisierenden Selektion, welche nützliche genetische Variation erhält die von Rekombination und Mutation erzeugt werden. Die Zuordnung zwischen Strukturen zur Lösungsrepräsentation (Genotyp) und Fitnessevaluierung (Phänotyp) beeinflusst das algorithmische Verhalten stark. Populationsweite Auswirkungen betreffend Bausteine, genetischer Diversität und Bloat entstehen durch das komplexe Zusammenwirken phänotypischen Operatoren (Selektion) und genotypischen Operatoren (Rekombination und Mutation). Dieser Mechanismus, bekannt als „Variation-Selektion Schleife", ist die treibende Kraft des emergente Verhalten von GP und bildet das Hauptforschungsthema dieser Arbeit.

Diese Arbeit zielt darauf ab, einen einheitlichen, theoretischen Rauem zu schaffen welcher Evolution in GP erklären kann. Dafür wird der Einfluss von auf das algorithmische Verhalten untersucht, basierend auf die Zuordnung von Genotyp und Phänotyp, unter Berücksichtigung der evolutionärer Operatoren (Selektion, Rekombination, Mutation). Wie der Titel der Arbeit bereits andeutet, besteht der wichtigste Beitrag aus einem neuartigen System zur Überwachung und Rückverfolgung von Genen über mehrere Generationen hinweg. Dies ermöglicht es das Verhalten von Bausteinen zu erforschen, sowie zu erkunden wie bei evolutionären Algorithmen aus kleinen Elementen nach und nach komplexere Strukturen gebildet werden.

Declaration

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäßentnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Dissertation ist mit dem elektronisch übermittelten Textdokument identisch.

Bogdan Burlacu, Linz, 2017

Contents

1 Introduction **5**

1.1 Thesis Statement . . . 5

1.1.1 Research Objectives . . . 6

1.1.2 Main Results and Contribution . . . 6

1.1.3 Thesis Structure and Organization . . . 7

2 Heuristics and Metaheuristics **9**

2.1 Optimization Problems . . . 9

2.1.1 Formal Statement . . . 9

2.1.2 Discrete and Combinatorial Optimization . . . 10

2.2 Heuristics . . . 11

2.2.1 Which Problems are Considered "Hard"? . . . 13

2.2.2 The Subset Sum Problem (Unidimensional Knapsack) . . . 14

2.2.3 Computational Class NP . . . 15

2.2.4 The Traveling Salesman Problem . . . 16

2.3 Metaheuristics . . . 19

2.3.1 Definition . . . 19

2.3.2 Randomized Algorithms and Probabilistic Methods . . . 20

2.3.3 Metaphors and Metaheuristics . . . 21

2.3.4 Taxonomy of Metaheuristics . . . 22

2.3.5 Trajectory-based Metaheuristics . . . 22

Local Search . . . 22

Simulated Annealing . . . 23

Tabu Search . . . 23

Scatter Search . . . 23

2.3.6 Population-based Metaheuristics . . . 23

Particle Swarm Optimization . . . 24

2.3.7 Constructive Metaheuristics . . . 24

Greedy Randomized Adaptive Search . . . 24

Ant Colony Optimization . . . 24

2.4 The No Free Lunch theorem . . . 25

2.4.1 Implications for Metaheuristics . . . 25

2.4.2 Summary . . . 29

3 Evolutionary Computation **30**
3.1 Introduction . . . 30
3.1.1 Evolution through Natural Selection . . . 30
3.1.2 A Short History of Population Genetics . . . 31
3.1.3 The Modern Synthesis . . . 33
3.1.4 Developmental and Molecular Genetics . . . 34
3.2 Evolutionary Search Methods . . . 35
3.2.1 Evolution Strategies . . . 36
3.2.2 Evolutionary Programming . . . 38
3.2.3 Differential Evolution . . . 39
3.2.4 Genetic Algorithms . . . 41
Recombination in Genetic Algorithms . . . 41
Selection in Genetic Algorithms . . . 43
Holland's Schema Theorem . . . 43
Criticism of the Schema Theorem . . . 45

4 Genetic Programming **47**
4.1 Solution Encoding . . . 47
4.1.1 Syntax Tree Encoding . . . 48
Closure and Sufficiency. Properties of Good Representations . . . 49
4.1.2 Other Encodings . . . 50
Grammar-based GP . . . 50
Graph-based GP . . . 51
Linear GP . . . 51
4.2 Genetic Programming Operators . . . 51
4.2.1 Tree Creation . . . 52
4.2.2 Selection . . . 53
Diversity and the Exploration-Exploitation Trade-off . . . 53
Adaptive Selection Pressure through Offspring Selection . . . 55
4.2.3 Crossover . . . 56
Crossover and Bloat . . . 58
Other Crossover Variants . . . 59
4.2.4 Mutation . . . 62
4.3 Genetic Programming Schema Theorems . . . 62
4.3.1 Koza's Schema Definition . . . 62
4.3.2 O'Reilly and Oppacher's Schema Definition . . . 63
4.3.3 Wigham's Schema Definition . . . 64
4.3.4 Rosca's Schema Definition . . . 64
4.3.5 Poli and Langdon's Schema Definition . . . 65
Exact Schema Theorem . . . 67
4.4 Advanced Concepts in Genetic Programming . . . 71

4.4.1 Genotype-Phenotype Maps . . . 71
4.4.2 Emergence, Evolvability and Robustness . . . 75

5 Tracing of Evolutionary Search Trajectories 81
5.1 Symbolic Regression in HeuristicLab . . . 82
5.2 Analysis of Evolutionary Dynamics . . . 84
5.2.1 Population Diversity . . . 84
Phenotypic Similarity . . . 84
Genotypic Similarity . . . 85
5.2.2 Genetic Operator Effectiveness . . . 86
5.2.3 Heredity, Inheritance and Genealogy Analysis . . . 87
Genealogy Graph: Recording Evolution . . . 87
Trace Graphs: Evolutionary Trajectory Analysis . . . 91
Trace-based Analysis Methods . . . 94
5.2.4 Schema-based Analysis Methods . . . 97
Tree Pattern Matching . . . 98
Schema Generation . . . 98
5.2.5 Schema-based GP Diversification Strategies . . . 100

6 Empirical Analysis 103
6.1 Standard GP . . . 104
6.1.1 Experiment Configuration . . . 104
6.1.2 Best and Average Solution Quality . . . 104
6.1.3 Selection Ratio and Parent Distribution . . . 105
6.1.4 Average Tree and Fragment Length . . . 106
6.1.5 Population Diversity . . . 111
6.1.6 Genetic Operator Effectiveness . . . 111
6.1.7 Schema Frequency Analysis . . . 112
6.1.8 Analysis of Solution Contribution Ratio . . . 115
6.2 Offspring Selection GP . . . 122
6.2.1 Experiment Configuration . . . 122
6.2.2 Best and Average Solution Qualities . . . 123
6.2.3 Selection Ratio and Parent Distribution . . . 124
6.2.4 Average Tree and Fragment Length . . . 125
6.2.5 Population Diversity . . . 128
6.2.6 Genetic Operator Effectiveness . . . 131
6.2.7 Schema Frequency Analysis . . . 136
6.2.8 Analysis of Solution Contribution Ratio . . . 138
6.3 Offspring Selection GP with Schemas . . . 142
6.3.1 OSGP-S Adaptive Replacement Ratio . . . 142
6.3.2 OSGP-S Fixed Replacement Ratio . . . 143

6.3.3 Comparison with OSGP 146
6.3.4 Overview of the Results 146

7 Final Remarks **148**
7.1 Overview 148
7.2 Main Contribution 150
7.3 Future Research Ideas 155

Appendix **156**

Glossary of Biological Terms **175**

Glossary of Computational Terms **180**

Bibliography **183**

List of Figures **200**

List of Tables **202**

1 Introduction

1.1 Thesis Statement

Genetic programming (GP) (Koza, 1992) has been around for more than two decades. Since its inception, it attracted many researchers, theoreticians and practitioners who were able to successfully apply it to many problem domains. The working idea of GP is very simple: evolve a population of computer programs using the principle of natural selection until they become reasonably suitable, or sufficiently well-adapted (according to a specified criteria or fitness measure) to solve a given problem.

From a computational standpoint, GP programs are usually represented by tree data structures which facilitate evaluation (trees can be easily evaluated in a recursive manner) and GP-specific operations such as crossover (subtree swap between trees) or mutation (random modification of a node or subtree). From a biological perspective, the evolved organisms' tree-like genotypes are mapped to phenotypes represented by their numerical evaluation. The most well-adapted phenotypes survive according to the laws of natural selection and their corresponding genes are passed on to the next generation.

Despite its straightforward nature, aspects of the evolutionary process and its dynamics remain still shrouded in mystery. In particular, the way genotype variation maps into phenotype variation, known as the *representation problem* poses considerable difficulty as it cannot be directly measured or quantified, although indirect frameworks based for example on the concept of *fitness landscape* (a mapping from a configuration space into the real numbers) can be used to examine its properties (Reidys and Stadler, 2002; Stadler and Stephens, 2003).

The genotype-phenotype map (GP-map) plays an essential role in the population's *evolvability*, defined by (Wagner and Altenberg, 1996b) as "the ability of random variations to sometimes produce improvement", or the "propensity to vary" in the sense that the genotype-phenotype map determines variability. Evolvability is directly connected to other important properties such as robustness against genotypic perturbations, redundancy or convergence rate. Intuitively, it describes a population's *potential* to evolve within a certain landscape, but not as a static property: on the contrary, the "evolution of evolvability" represents an observable, albeit hard to quantify, *emergent* property of GP systems. (Angeline, 1994) explains it as the emergent effect of indirect selection of genotypes based on their phenotypic properties (when the potential to adapt becomes a target of evolution). Similarly, (Altenberg, 1994a,b) shows that adaptation is an emergent property as

evolvability changes during the algorithm run through the proliferation of sub-expressions that have a higher chance of increasing fitness when added to programs. (Banzhaf, 2014; Hu and Banzhaf, 2010) argue that evolvability is selectable and environmental selection can improve the evolution of evolvability, accelerating evolution. They emphasize in this context the importance of incorporating new knowledge from areas such as molecular genetics, cell biology, or developmental biology towards the effort of understanding GP and other evolutionary algorithms. Further, they explain GP emergent behavior through the presence of top-down and bottom-up causal links between the selection mechanism and the variation producing operators.

1.1.1 Research Objectives

In this context, the main objective of this thesis is to describe GP evolutionary dynamics from a high-level perspective across the *genotype-phenotype map—fitness landscape—evolvability* axis. In more concrete terms, the goal of this research is to gain a better understanding of the evolutionary process, in particular to

- Provide a consistent and detailed treatment of GP as an emergent system centered around evolvability
- Analyze the exact effects of genetic operators and their consequences on the genotype-phenotype map, in terms of how they transform the randomly distributed initial genetic information into more and more powerful solution candidates
- Analyze the search trajectories of evolutionary algorithms in different hypothesis spaces for academic as well as real-world data mining applications.
- Investigate the evolvement of complete genealogical trees of solution candidates.

The insight obtained from the investigation of all the points above will, on the one hand, reveal aspects of schema-theories for tree-based GP, and, on the other hand, help design more powerful evolutionary algorithms and enable new application-oriented research projects.

Another important objective, implicitly assumed in the pursuement of the above-mentioned goals, is the development of a software framework that provides the necessary tools for our analysis. The developed procedures and algorithms described in this work are made public under an opensource license, as part of the HeuristicLab framework.

1.1.2 Main Results and Contribution

From the technical point of view, at the core of this approach lies a set of methods for saving the full genealogy and hereditary information of a GP run in the form of genealogy graphs. The core functionality is completed and enhanced by set of algorithms on genealogy

graphs allowing the user to investigate in detail the afore-mentioned aspects of evolutionary dynamics.

Therefore, the main contribution of this work is two-fold:

1) Firstly, it provides a comprehensive computational framework conveniently implemented as a HeuristicLab plugin, for the analysis of GP evolutionary dynamics.
2) Secondly, it brings new insight into the field, obtained through the application of new concepts and analysis methods on a selection of GP algorithms and test problems.

Technical Achievements

- ◇ Automatic, online construction of population genealogy graphs.
- ◇ Interactive visualization of lineages and genealogies.
- ◇ State-of-the-art tree matching algorithms (isomorphism and pattern matching).
- ◇ New structural and semantic diversity measures.
- ◇ Tracing of the genetic origins of individuals.
- ◇ Schema generation and matching.
- ◇ Prototype implementation of new algorithmic improvements.

Conceptual Achievements

- ◇ Integration with concepts from biology and population genetics.
- ◇ Unified treatment of evolutionary dynamics as an emergent phenomenon.
- ◇ Developmental approach using concepts from optimization theory (no free lunches) and complexity theory (properties of the genotype-phenotype maps).
- ◇ Analysis of genetic operator and search effectiveness.
- ◇ New schema-based self-tuning algorithm preserving the balance between exploration and exploitation.

1.1.3 Thesis Structure and Organization

Chapter 2 provides a theoretical introduction to optimization problems. Section 2.1 describes the basic optimization problem formulations in the continuous and discrete domains and the fundamental notions in the optimization vocabulary. Section 2.2 provides a theoretical introduction to algorithmic complexity and goes into more detail about the semantics of "algorithms" and "heuristics", with examples chosen to illustrate the main differences between exact and approximate solving methods. Section 2.3 defines metaheuristics from an algorithmic perspective and discusses their role in the field of optimization. It also

provides a taxonomy of existing metaheuristics based on their underlying metaphor or optimization paradigm. Section 2.4 discusses the theoretical properties of search algorithms in the context of optimization, and the implications of these properties on metaheuristics in general and on evolutionary algorithms in particular.

Chapter 3 is dedicated to modern evolutionary theory and to the topic of evolutionary computation. Section 3.1 introduces natural selection and the development of the modern evolutionary synthesis centered around population genetics. Then it discusses the developmental aspect of genetics and the importance of genes and genotype to phenotype mappings. Section 3.2 describes the idea of artificial evolution and the main evolutionary computation metaheuristics, with a focus on genetic algorithms and genetic algorithm schema theorems.

With the groundwork laid down by the previous chapters, Chapter 4 goes into depth on the topic of Genetic Programming (GP), describing in detail its particularities and algorithmic components like the encoding and operators. Section 4.1 goes through the most used encodings and discusses the design principles of suited representations as seen in the literature. Section 4.2 describes the main operators required by a GP system, covering existing state-of-the-art variants of each operator along with their advantages and disadvantages. Typical problems ensuing in GP systems like loss of diversity, premature convergence and the exploration-exploitation trade-off are also comprehensively treated. Section 4.3 discusses GP schema theorems and the theoretical insights they provide, while Section 4.4 discusses in a more general settings the importance of genotype-phenotype maps and the fundamental properties of evolving systems such as emergence, evolvability and robustness.

Chapter 5 introduces the analysis methods, methodology and algorithms representing the main contribution of this thesis. Section 5.1 describes the main features of symbolic regression in HeuristicLab. Section 5.2 presents the methodology for building genealogy graphs of GP populations and using them to investigate the various aspects of GP evolutionary dynamics, such as population diversity, effectiveness of genetic operators, inheritance patterns and schemas. Finally, a new algorithmic variant named "OSGP-S" (offspring selection genetic programming with schemas), inspired from the insight gained from our GP analysis is described in detail.

Chapter 6 presents a collection of empirical results obtained from testing different algorithmic configurations and test problems using the newly-introduced methodology and analysis methods. Sections 6.1 to 6.3 are dedicated each to one algorithmic configuration: standard GP (SGP), offspring selection GP (OSGP) and OSGP-S. The results are used to validate the proposed methodology and the new schema-based GP variant.

Finally, Chapter 7 provides an overview of this work and its significance for the field of Genetic Programming in Section 7.1, discusses our achievements and contribution in Section 7.2 and suggests future research ideas in Section 7.3.

2 Heuristics and Metaheuristics

2.1 Optimization Problems

We encounter optimization problems everywhere we turn. Whenever we try to minimize our expenses at the supermarket, find the fastest route from A to B, win at scrabble or chess, or in general make something better or more efficient (an action or decision, a design, a system), we are actually trying to solve an optimization problem.

Optimization problems which occur in most real-life scenarios involve finding solutions that have to satisfy one or more feasibility criteria. Mathematically speaking, the goal of such an optimization problem is to minimize or maximize an objective function[1], subject to domain constraints. In other words, the goal is to find the best solution out of the set of all feasible solutions. The set of all feasible solutions represents the *feasible region* of the problem, also known as a *search space* or *solution space*.

2.1.1 Formal Statement

In the most basic formulation, an optimization problem in the continuous domain can be defined as:

$$\underset{\mathbf{x}}{\text{minimize}} \quad f(\mathbf{x}) \tag{2.1}$$

$$\text{subject to} \quad g_i(\mathbf{x}) \leq 0, \qquad i = 1, \ldots, m \tag{2.2}$$

$$h_j(\mathbf{x}) = 0, \qquad j = 1, \ldots, p \tag{2.3}$$

where $f(\mathbf{x}) : \mathbb{A} \to \mathbb{R}$ is the objective function, $g_i(\mathbf{x}) \leq 0$ are inequality constraints and $h_i(\mathbf{x}) = 0$ are equality constraints[2]. Usually, $\mathbb{A}$ is a subset of the Euclidean space $\mathbb{R}^n$.

Most real-life optimization problems involve constraints. Constrained optimization methods differ from their unconstrained counterparts by the fact that they need only consider points in the feasible region. However, it may be useful in some cases to treat the problem as unconstrained and replace the constraints with penalty terms in the objective function (soft constraints).

[1] In different fields, the function f can have different names, such as *objective function*, *loss function*, *cost function*, or *fitness function*.

[2] A minimization problem can easily be turned into a maximization problem by simply negating the objective function.

When the objective function and the constraints are linear, we are dealing with a *linear programming problem* and otherwise when either the objective function or some of the constraints are non-linear we have a *nonlinear programming problem*.

We define a local minimum as a point $\mathbf{x}^* \in \mathbb{A}$ in the feasible region for which there exists a number $\delta \in \mathbb{R}, \delta > 0$ such that for all $\mathbf{x} \in \mathbb{A}$:

$$||\mathbf{x} - \mathbf{x}^*|| \leq \delta \implies f(\mathbf{x}^*) \leq f(\mathbf{x})$$

A global minimum will be a point $\mathbf{x}^*$ in the feasible space such that for all other $\mathbf{x}$, $f(\mathbf{x}^*) \leq f(\mathbf{x})$. Obviously, it is much more difficult to find the global optimum than it is to find local optima.

Thus, we have given so far a mathematical formulation for constrained optimization problems in the continuous domain. But how can these problems be solved with the help of computers?

Computers are inherently discrete systems; therefore, computational methods for continuous optimization problems require the discretization of the input variables. Discretization is necessary because continuous models and equations need to be made suitable for numerical evaluation. This is achieved using the so-called *iterates*: discrete values or steps generated by the algorithm using knowledge gained in previous iterations or information about the model at the current iterate. Among well-known continuous optimization paradigms we find *linear programming*, *gradient descent*, *convex programming* or *conic optimization* (Wright, 1999).

2.1.2 Discrete and Combinatorial Optimization

A discrete optimization problem can be defined as a tuple (E, c) where E is a finite or countably infinite set called the *ground set* and $c : E \to \mathbb{R}$ is the objective function that needs to be minimized or maximized. For combinatorial problems, we define the set of feasible solutions $\mathcal{F}$ as a subset of 2^E (the set of all subsets of E) and we set the objective to find a set $F^* \in \mathcal{F}$ such that

$$c(F^*) = \sum_{e \in F^*} c(e)$$

is minimized or maximized (Lee, 2004). We notice from the problem definition that, although the space of feasible solutions for these problems is finite in nature, explicit enumeration quickly becomes infeasible as n distinct objects can be arranged in $n!$ ways. Therefore, even for small problem dimensions, the number of steps needed by a search procedure to exhaustively examine the solution space grows exponentially. This phenomenon known as *combinatorial explosion* makes it very difficult to approach these problems using conventional methods or algorithms. In fact, the vast majority of practical combinatorial optimization problems can be described as "finding a needle in a haystack" and need to be approached cleverly using carefully designed approximation methods or heuristics.

In many relatively simple to formulate problems such as the minimum spanning tree problem, the traveling salesman problem (discussed in Section 2.2.4), the knapsack problem, the vehicle routing problem and others especially from the field of operational research, no efficient exact algorithms for finding the optimal solution are known. It becomes reasonable in such cases to employ approximate solving methods based on various "shortcuts" or heuristics.

2.2 Heuristics

The word *heuristic* originates from the Ancient Greek εὑρίσκω (heurískō) meaning "I find, discover", and is used to describe things or methods related to the process of learning, discovery or problem-solving. This definition already gives information about the scope and implications of heuristics: they involve exploration, (educated) guessing, learning from experience or through trial and error. We could imagine heuristics as "shortcuts", practical steps towards some practical objective. They are not guaranteed to be optimal or perfect, but sufficient according to some set of requirements.

Computational solving methods can be roughly divided into three categories:

1) Exact methods (algorithms) consist of an exact order of steps that will invariably lead to the correct or optimal solution in a finite amount of time. Algorithms are mathematically proven to be correct and to perform within certain complexity bounds.

2) Approximate methods or algorithms are mathematically proven to produce a solution within a certain distance from the optimal solution. Usually the range is given relative to the optimum by a so-called approximation factor no larger than $1 + \epsilon$, where ϵ usually depends on parameters. Their performance and complexity is again well understood and theoretically proven.

3) Heuristic methods: these methods essentially consist of hunches, guesses, rules of thumb of varying complexity and other techniques derived from experience or from empirical evidence. There are usually no guarantees about their optimality or computational complexity and they are hard to study from a theoretical standpoint. The reason why heuristics are so interesting is because they have distinguished themselves as very useful practical solving tools, producing good results (not necessarily optimal) in a short amount of time.

In computer science, heuristics are applied to hard problems which cannot be solved exactly in a reasonable amount of time. They are designed to be faster by searching not for the perfect solution but for a "good enough" approximation, where good enough describes a satisfactory compromise between speed or computational cost and optimality or completeness. In practice, this trade-off is often very advantageous, as the obtained

solution approximations may come very close to the optimal solution while keeping the computational costs manageable.

Historically speaking, the computer science community has been slow and reluctant to accept heuristics as an important field of research. Fred Glover, the inventor of scatter search and tabu search expressed this fact quite poetically in 1977:

> "Algorithms are conceived in analytic purity in the high citadels of academic research, heuristics are midwifed by expediency in the dark corners of the practitioner's lair." (Glover, 1977)

Glover also suggests that optimality ("ultimate convergence", as he calls it) is an idealistic goal:

> "Algorithms, after all, are merely fastidious heuristics in which epsilons and deltas abide by the dictates of mathematical etiquette. It may even be said that algorithms exhibit a somewhat compulsive aspect, being denied the freedom that would allow an occasional inconsistency or an exception to ultimate convergence. (Unfortunately, ultimate convergence sometimes acquires a religious significance; it seems not to happen in this world.)" (Glover, 1977)

Algorithms for which bounds and correctness were mathematically proven were considered to be a superior field compared to heuristics, even though heuristic methods had proven themselves more successful in practice, in those cases where the dimensions of the problem rendered exact methods impractical and infeasible. It will become clear that heuristics, as exponents of the human way of thinking, represent very powerful problem-solving tools. Intuitively, there are two fundamental aspects to solving a problem:

1) Knowing what qualifies as an acceptable solution (validation or verification step)
2) Knowing how (through which sequence of steps) a solution might be reached (solving step)

People, as opposed to computers, are experts in applying so-called "rules of thumb" of varying complexity, obtaining if not a satisfactory solution, then at least a very good starting point that can be subsequently improved upon. An essential instrument for this process is the *analogy*, described beautifully by mathematician G. Polya:

> "Analogy pervades all our thinking, our everyday speech and our trivial conclusions as well as artistic ways of expression and the highest scientific achievements. Analogy is used on very different levels. People often use vague, ambiguous, incomplete or incompletely clarified analogies, but analogy may reach the level of mathematical precision. All sorts of analogy may play a role in the discovery of the solution and so we should not neglect any sort." (Polya, 1971)

This definition is very important, since analogy and metaphor play an important role in the development of heuristic and metaheuristic methods.

Human intuition often works by forming an image of what the solution looks like before coming up with a way to reach it. The ability to rapidly verify whether a solution will work plays an important role in this process. In many cases the human approach is neither too exact or complete, nor does it exhaustively produce solutions; but it is very robust, and it works just well enough to solve the problem with minimal fuss.

Heuristics allow computers to employ the same paradigm in those cases where no exact, mathematically proven and computationally feasible algorithm is known.

2.2.1 Which Problems are Considered "Hard"?

The formalization of problem solving in computer science begins with the Turing machine, a hypothetical machine invented by mathematician Alan M. Turing (1912-1954) as a theoretical model of computation in the discrete domain. Turing machines can simulate the logic of any computer program and can be used to characterize the theoretical properties of algorithms.

A common way to characterize problem hardness is to estimate the resources taken by an algorithm to compute the solution as functions of the size of its input (the problem size). The limiting (asymptotic) behavior of these functions, usually expressed in big O notation (where "O" refers to the maximum order of the function in the worst case scenario, excluding coefficients and lower order terms), provides a good indication of algorithm complexity, describing the growth in resource usage relative to the growth in input size.

Notation	**Name**	**Example**
$O(1)$	constant	test if a binary number is even or odd
$O(\log n)$	logarithmic	binary search
$O(n)$	linear	finding an item in an unsorted list or array
$O(n \log n)$	loglinear or quasilinear	fast Fourier transform, heapsort, quicksort, merge sort
$O(n^2)$	quadratic	bubble sort, insertion sort direct convolution
$O(n^3)$	cubic	naive multiplication of two $n \times n$ matrices calculating partial correlation tree edit distance
$O(n^c)$	polynomial	maximum matching for bipartite graphs
$O(c^n), c > 1$	exponential	solving the traveling salesman problem using dynamic programming

Table 2.1: Algorithm complexity classes, in increasing complexity from top to bottom

Table 2.1 shows the most common complexity classes for algorithms, based on their behavior when the input size is changed. The most common analysis method employed in the study of algorithms is *time complexity analysis*[3], which gives a theoretical bound for the time needed by an algorithm to run (as a function of the length of its input).

In computational complexity theory, problems are considered tractable (feasibly computable) if they can be solved by a deterministic Turing machine (DTM) in polynomial time. That is, if the running time is bounded by a polynomial expression in the size of the input for the algorithm. Such problems belong to the computational class P. Problems for which a "quick" polynomial time solving algorithm is not known are said to be "hard" and are usually approached in practice using different strategies. In the following subsections some examples of computationally-hard combinatorial problems will be given. Though in no way exhaustive (as such efforts are beyond the scope of this work), the treatment of each example will outline the main reasons why heuristic methods are so suitable for combinatorial optimization problems.

2.2.2 The Subset Sum Problem (Unidimensional Knapsack)

The *subset sum problem* (SSP) poses the following question: given a set of integers $S = \{x_1, \ldots, x_n\}$ and a target value s, is there a non-empty subset of S whose sum is the closest to s without exceeding s?[4]

Several solving approaches are possible:

- The naïve approach, which computes the sum of each possible subset of S has a complexity of $O(2^n n)$ since there are 2^n subsets each containing at most n elements.
- While still exponential, the algorithm introduced by (Horowitz and Sahni, 1974), achieves a computational complexity of $O(2^{n/2})$ by arbitrarily splitting S in two, generating a sorted list of all possible sums for each split, and then merging the lists (the merging can be done in linear time).
- Dynamic programming achieves pseudo-polynomial time by taking advantage of the SSP's optimal substructure and overlapping subproblems properties. The resulting algorithm is polynomial in the size of the problem but exponential in the number of bits used to represent it and has a complexity of $O(n^r)$ where r is a constant which bounds all subset sums. For the case in which x_i is positive and bounded by a fixed constant C, the time complexity becomes linear $O(nC)$ (Pisinger, 1995).

[3] Time complexity is usually counted as the number of basic operations (ie., operations like addition, division etc. that take a fixed amount of time) performed by the algorithm, where the algorithm is modeled by a Turing machine.

[4] The more general Knapsack problem is: given a set of items, each with a mass w_i and a value v_i, we have to determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. We recognize an SSP instance when for each item $w_i = v_i$ (unidimensional case).

There is no known algorithm for the SSP that runs in less than $O(2^{n/2})$ in the general case, thus, for large enough n exact solvers become unfeasible. A good compromise in this situation would be to look for an approximate solution that is easier to compute and guarantee an accuracy bounded by a parameter ϵ. We mention a few approximation algorithms for the SSP:

- (Kellerer et al., 1997) solve the SSP in $O(\min\{n/\epsilon, n + 1/\epsilon^2 \log(1/\epsilon)\})$ time and $O(n + 1/\epsilon)$ space, with accuracy ϵ
- (Kellerer et al., 2003) improved upon the previous result and achieve $O(\min\{n \cdot 1/\epsilon, n + 1/\epsilon^2 \log(1/\epsilon)\}$ time and $O(n + 1/\epsilon)$ space complexity.
- (Williamson and Shmoys, 2011) give an approximate algorithm with runtime complexity of $O(n^3/\epsilon)$ and bounded by a polynomial in $1/\epsilon$ (where $\epsilon > 0$ is an error parameter).

In summary, the SSP is a difficult problem, as the combination of numbers that maximizes the sum is hard to find. But the reason it was included here as an example is to illustrate an important aspect that allows the characterization of hard problems in a more formal way. We notice that for this type of problem, however difficult it may be to come up with a suitable subset, any such candidate solution can be easily verified by simply computing the sum. This means that at least in theory, we could simply try *guessing the solution* since guessing costs nothing and *verification* costs nothing. Moreover, if we could keep guessing and improve our guesses based on experience, we could eventually achieve optimal or near optimal solutions.

This realization on the one hand, establishes the context for a new way to characterize hard problems and on the other hand, serves as a justification of why heuristics can be very useful in practice.

We continue with a formal definition of problem hardness.

2.2.3 Computational Class NP

Formally, problems for which no polynomial time solver is known, but for which a polynomial verifier exists are said to belong to computational class NP[5] and can be described by the following equivalent statements:

(a) They are verifiable by a deterministic Turing machine (DTM) in polynomial time

(b) They are solvable by a non-deterministic Turing machine (NTM) in polynomial time

[5]Whether or not a problem for which every solution can be verified in polynomial time is solvable in polynomial time remains to be decided. The $P = NP$ problem remains unsolved since its formulation in Stephen Cook's 1971 paper "The complexity of theorem proving procedures" (Cook, 1971). We say that a problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H.

A particular case of the non-deterministic Turing machine is the probabilistic Turing machine (PTM) which randomly chooses between the available state transitions at each point according to some probability distribution. When the transition probabilities are equal, the probabilistic Turing machine can also be described as a deterministic Turing machine with an added tape full of random bits called the *random tape*.

Probabilistic models of computation are useful for describing heuristics or strategies comprising of several heuristics which employ some form of "guessing" in order to come up with potential solutions. Cheap solution validation (polynomial *verifier*) makes it feasible to use different solving strategies in a two-step approach:

1) Generation of solution candidates by some heuristic
2) Validation of the generated candidates using a polynomial verifier

Many heuristics exploit known properties of the solution space (for example, by incorporating domain-specific knowledge provided by human experts) in order to produce solution candidates that can then be validated in polynomial time.

2.2.4 The Traveling Salesman Problem

The *traveling salesman problem* (TSP) is a landmark problem and one of the most intensively studied in the field of combinatorial optimization. It was proved to be NP-complete by (Karp, 1972). The problem definition is equivalent to finding the Hamiltonian cycle with the least weight in a complete weighted graph: given a list of cities and the distance between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? In its decision version, the problem formulation is: given a length L, is there any tour in the graph shorter than L?)

This problem has many practical applications in the fields of logistics, planning, microchip design or even genome sequencing (when the concept of city is appropriately represented by problem elements such as customers, soldering points or DNA fragments). As with the previous example, there exist several polynomial time approximate algorithms that are able to find "good" but non-optimal solutions to the TSP problem. The naïve brute-force approach for this problem would result in a runtime complexity of $O(n!)$, where n is the number of cities, rendering it highly impractical even for small tours (20 cities). The Held-Karp algorithm (Held and Karp, 1961), one of the earliest algorithms to employ dynamic programming, achieved a complexity of $O(n^2 2^n)$, still unfeasible for large instances.

In recent years various algorithmic solutions employing *branch-and-bound* or *cutting-plane* methods were able to improve the runtime complexity of TSP solvers. The best approximation algorithm is Arora's PTAS for the Euclidean TSP (Arora, 1998) based on a geometric partitioning of the plane. It guarantees a $(1 + 1/c)$-approximation for every $c > 1$ in $O(n(\log n)^{O(c)})$ time[6].

[6] Sanjeev Arora and Joseph S. B. Mitchell were awarded the Gödel Prize in 2010 for their concurrent

Given its complexity and importance, the TSP problem has become a standard benchmark problem for combinatorial optimization algorithms. Heuristic methods for solving the TSP problem can be classified into constructive and iterative methods. Constructive heuristics execute a sequence of operations until a valid tour is reached, while iterative heuristics start from a valid tour and try to improve it usually by local search. Constructive heuristics include:

- *Nearest-neighbour (greedy)* heuristics build a tour by adding an edge to the nearest unvisited city. For the Euclidean TSP, (Rosenkrantz et al., 1977) showed that the approximation ratio of the nearest-neighbour method is bounded above by a logarithmic function in the number of cities, so that $\mathrm{NN}(I)/\mathrm{OPT}(I) \leq 0.5(\lfloor log(n) \rfloor + 1)$, otherwise the runtime complexity is $O(n^2)$

- *Savings heuristics* as originally described by (Clarke and Wright, 1964) try to obtain the overall minimum cost by building a tour from edges ordered by increasing cost (skipping edges that would lead to an invalid tour). Similar to the above, this algorithm has a complexity of $O(n^2 log(n))$ with an approximation factor of $\lceil log(n) \rceil + 1$ when the triangle inequality is satisfied.

- The algorithm by (Christofides, 1976) guarantees an approximation factor of $3/2$ for Euclidean instances. The overall time complexity for Christofides algorithm is $O(n^3)$, although in practice it is competitive and often surpasses the other algorithms.

Iterative heuristics are difficult to analyze from a theoretical perspective, however they represent the most successful class of approximation algorithms. Furthermore, hybrid approaches that combine local search with a higher-level guidance mechanism such as metaheuristics have been shown to achieve near-optimal (and sometimes optimal) solutions (Aarts and Lenstra, 1997; Hoos and Sttzle, 2004; Toth and Vigo, 2001). This statement applies not only to the TSP but to other difficult combinatorial optimization problems. Among the most used methods for the TSP we find:

- The *2-Opt (pairwise exchange)* method by (Croes, 1958) is probably the most popular TSP solving method as it usually achieves very good results on "real world" Euclidean instances. It works by removing two edges from the cities graph and then reconnecting the relevant cities in an (locally) optimal way. According to (Johnson and McGeoch, 1995), the 2-Opt method requires a subquadratic number of improving steps to reach a near-optimal solution. Another study by (Englert et al., 2007) presents a family of problem instances on which 2-Opt can take an exponential number of steps. So far, the worst case running time on Euclidean instances is unknown.

- The *Lin-Kernighan* (L-K) algorithm (Lin and Kernighan, 1973) or *k-Opt* works by deleting k mutually disjoint edges and reassembling the remaining fragments into a tour leaving no disjoint subtours. The authors state that their method generates

discovery of a PTAS for the Euclidean TSP

optimum or near-optimum solutions for the TSP problem in $O(n^2)$ approximate runtime with above 95% confidence. From the general k-opt family, the 3-opt method developed initially by (Lin, 1965) as an extension to the work of Croes works very well in practice as well. Additionally, when the edges are not completely disjoint (two of them are adjacent to each other), it is possible for 3-opt to achieve considerable improvement over 2-opt by restricting the 3-changes to the subset of adjacent edges.

- ◇ Stochastic optimization methods were also successfully employed for solving the TSP. For example, (Kirkpatrick et al., 1983; Černý, 1985) were able to efficiently achieve near-optimal solutions with a simulated annealing algorithm inspired by the Monte Carlo method of sampling the configuration space of a thermodynamical system (Metropolis and Ulam, 1949; Metropolis et al., 1953), in which the total energy of the system was equivalent to the TPS tour length.

- ◇ Other metaheuristics for solving the TSP include genetic algorithms, tabu search (Glover, 1986) or ant colony optimization (Dorigo et al., 1996) (we will talk about these methods later).

The L-K algorithm in its many implementations was the preferred way of solving the TSP problem for over two decades. The common practice suggested by (Lin, 1965) was to start the search multiple times from randomly generated initial tours in order to increase the chances of finding the best tour; however it was shown by (Martin et al., 1992) that finding the global optimum by repeated random starts becomes unmanageable for large numbers of cities. It was instead suggested to extend the L-K neighbourhood by sampling locally optimal tours using a sampling biased towards shorter tours. By combining local opt with simulated annealing (slightly changing the current best tour and keeping it if better), the new algorithm by (Martin et al., 1992) known as *Chained Lin-Kernighan* was able to significantly outperform L-K. Currently the state-of-the-art implementation of chained L-K by (Applegate et al., 2001, 2003) part of the *Concorde TSP Solver*[7] is considered to be the fastest exact TSP solver.

Summing up, heuristics are approximate methods that are used to find good (near-optimal) solutions to NP optimization problems. These problems are characterized by huge solution spaces that cannot be feasibly explored in an exhaustive manner with our current knowledge of algorithms. We defined NP problems in a formal manner as either verifiable by a DTM in polynomial time or solvable by a NTM in polynomial time. It follows through the equivalence of definitions that heuristic solvers for any such problem can be either be deterministic or non-deterministic (ie., stochastic). For example, local search heuristics such as neighbourhood search can be either deterministic (nearest-neighbour) or stochastic (generate neighbours according to some probability distribution).

[7] http://www.math.uwaterloo.ca/tsp/concorde/index.html

2.3 Metaheuristics

2.3.1 Definition

We can define *metaheuristics*[8] as high-level strategies that guide a hierarchical collection of subordinate lower-level procedures or heuristics towards a given goal. A good overview of metaheuristics, including a number of definitions from earlier literature is given in (Blum and Roli, 2003):

> "A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions." (Osman and Laporte, 1996)

> "A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method." (Voss et al., 1999)

> "Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. But, the main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent, biased form." (Stützle, 1999)

All of the definitions above outline several important properties of metaheuristics:

- They provide high-level guidance for a number of lower-level problem-specific procedures that have to work together towards the goal

[8] metá + heurískó, meaning beyond, in an upper level (from Greek)

- ◇ They represent general, problem-independent optimization paradigms
- ◇ They provide near-optimal (approximate) solutions
- ◇ They use randomness in a beneficial way, to improve convergence and avoid getting stuck in local optima
- ◇ They provide a balance between *diversification* (the ability to "jump" in different regions of the search space) and *intensification* (the ability to improve locally within neighbourhoods in the search space).

From the computational complexity viewpoint, both heuristics and by extension, metaheuristics are assumed to be at most polynomial in runtime. Therefore a metaheuristic will always contain a polynomial verifier along with other deterministic or non-deterministic components.

A prime example of a metaheuristic that can efficiently explore a combinatorial search space was given in the TSP example. The chained Lin-Kernighan algorithm achieved its speed by combining aspects of heuristic search (k-opt) with aspects of stochastic optimization (simulated annealing). Metaheuristics that make use of non-deterministic or stochastic components are studied in the field of *Stochastic Optimization*. The idea is to inject randomness into the process in order to accelerate progress (convergence speed), make the method less sensitive to modeling errors and avoid getting stuck into local optima.

2.3.2 Randomized Algorithms and Probabilistic Methods

It is important to distinguish between stochastic methods such as Chained L-K that vary their output (with a small chance of producing incorrect output) and deterministic methods that incorporate a probabilistic element only in selecting their input, in order to reduce the expected running time or memory usage. Randomized algorithms can be split into two categories, "Las Vegas" and "Monte Carlo"[9]:

- ◇ Las Vegas algorithms always produce the correct answer in an amount of time that can vary randomly but will be bounded in expectation. For example, the `quicksort` algorithm is usually randomized by choosing a random pivot. The randomized version has the nice property of $O(n \log n)$ expected worst-case complexity (as opposed to $O(n^2)$ for the non-randomized version).
- ◇ Monte Carlo algorithms have deterministic runtime but produce non-deterministic results (there is a chance the output may be incorrect). The vast majority of metaheuristics fall into the category of Monte Carlo methods.

[9] The history of these two names is definitely interesting and deserves attention. For the Monte Carlo method, the reader is invited to study the papers of Metropolis (Metropolis and Ulam, 1949; Metropolis et al., 1953) or the fascinating historical account of (Eckhardt, 1987). The "Las Vegas algorithm" was coined by mathematician László Babai (Babai, 1979) to denote a stronger procedure where the correctness of the result can be checked.

By using Markov's inequality (sometimes also referred to as Chebyshev's inequality), a Las Vegas algorithm can be converted into a Monte Carlo algorithm through early termination. A succinct formulation of the difference between the two types of randomized algorithms is that Monte Carlo algorithms are always fast but only probably correct while Las Vegas algorithms are always correct but only probably fast.

When tackling hard optimization problems, one cannot hope to achieve the best of both worlds (in terms of speed *and* correctness). Instead, we are rather interested in achieving a high probability of success (the algorithm output is correct most of the time) within a short amount of runtime. From this point of view, metaheuristics represent a very good compromise between runtime and correctness.

In practice, for large problem instances metaheuristics have proven themselves superior to exact methods like branch-and-bound or dynamic programming. By incorporating some degree of randomness either at the level of the whole strategy or in the lower-level components, the problem space can be more efficiently sampled leading to better results and runtime performance. Of course, the fact that metaheuristics follow a search strategy and use randomness effectively and beneficially makes them vastly superior to random search.

2.3.3 Metaphors and Metaheuristics

Many different physical, biological or social phenomena have inspired the design of metaheuristics. Table 2.2 shows a list of metaphors along with the optimization methods that they inspired.

Metaphor	**Optimization method**
Intelligent behavior of colonial organisms (swarm intelligence) like bacteria, bees, ants, cells, etc.	Ant colony optimization, artificial bee colony, bacterial colony optimization, artificial immune system, etc.
Memory (remembering previous steps)	Tabu search, Scatter search
Annealing (thermodynamic process)	Simulated annealing
Natural evolution	Evolutionary programming, evolution strategies, genetic algorithms, genetic programming, differential evolution, etc.
Behaviour of various biological species of insects and animals	Shuffled frog leaping algorithm, cuckoo search, etc.

Table 2.2: Metaphors and metaheuristics

The general concepts behind metaheuristics (and their metaphors) include:

- Making random moves in the solution space

- ◇ Keeping a record of the past trajectory in order to avoid "walking in circles" (tabu search, scatter search)
- ◇ Accepting "worse" solutions with some probability, to avoid getting stuck in local optima (simulated annealing)
- ◇ Having "agents" explore different regions of the solution space, exchange information and perform a cooperative search (ant colony optimization, particle swarm optimization)
- ◇ Gradually generating and improving solution candidates by making them compete for survival (evolutionary optimization methods)

2.3.4 Taxonomy of Metaheuristics

It would be beyond the scope of this work to delve too deeply into the details of the various types of metaheuristics. However, a short description of the main types of metaheuristics will be given below. For a more in-depth treatment of metaheuristics, the reader is invited to take a look at the excellent (and free) book by (Luke, 2013) or the overview of (Blum and Roli, 2003).

Metaheuristics can be classified according to many different criteria, depending on whether they are nature-inspired or not, whether they are trajectory (single point search) or population-based, whether they use dynamic or static objective functions, single or various neighbourhood structures, or whether they incorporate some kind of memory or not.

2.3.5 Trajectory-based Metaheuristics

Trajectory methods owe their name to the fact that the search starts from a single initial state that is gradually improved upon, describing a trajectory in the state space. The fitness landscape for this type of methods is defined by the representation together with the neighbourhood structure from which new solutions are extracted. The manner in which new solutions are picked from the neighbourhood determines whether the method is greedy, probabilistic, or somewhere in between.

Local Search

The most basic trajectory-based methods are local search methods in which a *move* (a transition from one state to the next) is only performed if the new solution candidate (extracted from the neighbourhood) is better than the previous best. This kind of local search is also called iterative improvement due to the condition for accepting moves. The performance of local search methods strongly depends on the neighbourhood function and on the termination conditions (which can vary from reaching a local minimum to

exceeding a maximum number of iterations or CPU time). Local search methods include hill climbing and its variants, tabu search and simulated annealing.

Simulated Annealing

Simulated annealing (Kirkpatrick et al., 1983; Černý, 1985) resolves the problem of getting stuck in local optima by sometimes accepting "worse" moves, with decreasing probability dependant on the temperature T of the thermodynamic system and on the quality difference between the two solution candidates. The probability is usually computed using the Boltzmann distribution $\exp\left(\frac{f(s')-f(s)}{T}\right)$, where s represents the current solution and s' the new solution that was sampled from the neighbourhood.

Tabu Search

Tabu Search (TS) (Glover, 1986) is probably the most popular local search metaheuristic. The main idea behind TS is to avoid cycles (and implicitly, local optima) by forbidding or penalizing certain moves in the solution neighbourhood. This idea is realized by using a memory structure in which the most recently visited few solutions are stored (for example, those visited in the past n iterations). These solutions are said to be "tabu" and are penalized or altogether forbidden to be visited again. The memory structure is called a *tabu list* and its capacity is called the *tabu tenure*.

Scatter Search

Scatter Search (SS) (Glover, 1998) is an evolutionary method inspired from earlier ideas of combining decision rules and problem constraints for solving combinatorial and nonlinear optimization problems. SS consists of different components which work together to systematically combine solution vectors within (or around) a region of the solution space given by a set of reference points (good solutions obtained by prior problem solving efforts). A more in-depth discussion of SS and its components is available for example in (Martí et al., 2006).

2.3.6 Population-based Metaheuristics

Strictly speaking, population-based methods are those methods which produce new solutions by use of recombination operators such that parts of the old solutions are combined in new ways. This category is represented mostly by evolutionary algorithms modeled after the process of natural evolution, which will be discussed in detail in Chapter 3. In evolutionary methods, solution improvement is realized via the interplay between recombination operators (crossover and mutation) and the selection process which allows only the fittest individuals to survive and be part of the next iteration.

In a more relaxed classification, population-based methods could also include methods that employ different search techniques, such as particle swarm (PSO) (Kennedy and Eberhart, 1995) or ant colony optimization (ACO) (Dorigo et al., 1996) (see below).

The evolutionary-inspired population-based methods are discussed in Section 3.2.

Particle Swarm Optimization

Particle swarm optimization (PSO) (Kennedy and Eberhart, 1995) uses a population of particles that are initialized with a random velocity which gives them a trajectory in the solution space. At the end of each iteration, the fitness values are calculated for each particle. Then, each particle is accelerated towards fitter particles in the solution space (either taking the best known solution, or just the best known solution among a neighbourhood of the particle being moved).

2.3.7 Constructive Metaheuristics

In contrast with iterative improvement methods, constructive metaheuristics build solutions out of basic solution elements, one element at a time. As we have seen in the TSP problem example, constructive heuristics can implement some kind of greedy principle combined with a local improvement phase designed to improve the solution after the construction phase.

Greedy Randomized Adaptive Search

Greedy randomized adaptive search (GRASP) (Feo and Resende, 1989) could be described as a probabilistic greedy method in which solutions are constructed by randomly choosing at each step between a number of high-quality moves that are ranked by a greedy ranking method.

Ant Colony Optimization

The ant colony optimization (ACO) (Dorigo et al., 1996) method exchanges information between ants (agents) by having each ant deposit a trail of pheromones on its walked path. The amount of deposited pheromones depends on the path length and shorter paths are favored by depositing more pheromones. The probability that other ants would choose to walk the same path is then weighted according to the pheromone concentration. In the end, the optimal path within the graph or lattice will be the one favored by the ants with the highest pheromone concentration. Due to its working principle, ACO can be used for solving computational problems which can be reduced to finding good paths in graphs.

2.4 The No Free Lunch theorem

We discuss here aspects of the No Free Lunch theorems and their implications on metaheuristics and in particular evolutionary algorithms. This discussion is important because it provides a necessary context for the more detailed discussion about evolutionary search landscapes, neutrality, and genotype-phenotype mappings in Chapter 4.

The *no free lunch theorem* (NFL) states that all algorithms that search for an extremum of a cost function f perform exactly the same when averaged over the set F of all such functions. It was developed as a "framework for investigating search" that can describe problems from an information-theoretical perspective and help design more powerful algorithms that match the particular features of particular problems.

Although NFL concepts were previously known in the computer science community, (Wolpert and Macready, 1995) were the first to provide a formal mathematical proof, highlighting the need for exploiting problem-specific knowledge to achieve better than random performance. In their original phrasing, the NFL theorem states that "there are no «free lunches» for effective optimization; any algorithm performs only as well as the knowledge concerning the cost function put into the cost algorithm". The original paper includes a proof of the NFL theorem for deterministic, non-retracing algorithms but the authors explicitly state that their results also apply to stochastic and retracing algorithms. This last point especially concerning metaheuristics such as evolutionary algorithms was a point of open debate, as we will see in what follows.

In the context of optimization, the same authors (Wolpert and Macready, 1997) discuss the utility of search algorithms based on what can be deduced *a priori* from their mathematical properties. In this approach, algorithm performance is assessed in terms of the number of distinct function evaluations required to find a certain solution using an oracle-based[10] model of computation, and the results are given in a probabilistic formulation.

2.4.1 Implications for Metaheuristics

The implication of the NFL theorem is that, since some black-box algorithms will outperform others on some problems and will be outperformed on other problems, there is no reason to assume that one search method is better than the others, or even that it will be better on average than random search. In the particular example of evolutionary algorithms, this means that we cannot assume natural selection to be an effective search strategy and furthermore, nothing can be stated about the usefulness of evolutionary heuristics. The take-home message is that any blackbox algorithm has the same average performance as random search, and that problem-specific knowledge has to be incorporated in order to be better than random search. As we will see, this is not entirely correct.

It will not come as a surprise that the conclusion by Wolpert and Macready on the

[10] An "oracle" is a computational construct that can decide certain problems in a single operation

usefulness of guided random search methods did not sit well with other proponents of metaheuristics, and sparked some controversy in the computer science community. Some of the arguments and further developments of the NFL theorem are summarized below:

- (Droste et al., 1998) argued that the scenario on which the NFL is based does not reflect a realistic optimization scenario. They showed that in restricted black box optimization scenarios where it would actually make sense to apply evolutionary algorithms, there exist correlations between fitness values of different points in the solution space. The way these correlations or dependencies are exploited by the various search algorithms can explain why one algorithm may outperform another, invalidating the NFL theorem. The authors provide empirical proof that evolutionary algorithms perform just as well as the other specialized algorithms in concrete optimization scenarios. They conclude that the NFL theorem can only be applied to unrestricted black box optimization scenarios that do not occur in practice. Although specific techniques are superior to general ones in some scenarios, a "small free lunch" or "free appetizer" is still possible.

- (Schumacher et al., 2001) prove that the NFL theorem holds for a set of functions if and only if that set of functions is *closed under permutation* (c.u.p.), meaning that for every $f \in F$, $f : \mathcal{X} \to \mathcal{Y}$, every permutation of f is also in F, where the permutation σf of f is the function $\sigma f : \mathcal{X} \to \mathcal{Y}$ defined by $\sigma f(x) = f\left(\sigma^{-1}(x)\right)$. This sharpening of the NFL has powerful implications as it ultimately leads to the conclusion that in most cases NFL theorems are not applicable (see below).

- (Igel and Toussaint, 2003b) prove that the actual fraction of permutation-invariant subsets of the set of all possible functions is "negligibly small", then contend that "classes of objective functions resulting from important classes of real-world problems are likely not to be closed under permutation". This means that for important classes of real-world problems, the sharpened NFL scenario is not satisfied. The authors notice that the fact a problem class is not c.u.p. does not lead to a "free lunch", but ensures the possibility of a "free appetizer".

- (Igel and Toussaint, 2003a) use the sharpened NFL theorem to determine the average number of fitness evaluations necessary to reach a desirable solution in the context of evolutionary search. They find that this number depends on the size of the genotype space and the number of genotypes mapped to optimal solutions, but is not greatly affected by neutrality (ie., increase of genotype space). We briefly sketch the formal argument:

  - Let $\mathcal{G}$ be a finite search space with cardinality $n = |\mathcal{G}|$
  - Let $F_m \subset \mathcal{F}^{\mathcal{G}}$ be the set of all functions where m elements in $\mathcal{G}$ are mapped to optimal solutions. It can be shown that $F_m \subset \mathcal{F}^{\mathcal{G}}$ is c.u.p.

- It results from the NFL theorem (Schumacher et al., 2001) that the time to find an optimum averaged over all $f \in F_m$ is the same for all algorithms
- Then, the expected number of evaluations $E\{T_{n,m}\}$ averaged over $F_m \subseteq \mathcal{F}^{\mathcal{G}}$ is equal to

$$E\{T_{n,m}\} = \frac{n+1}{m+1}$$

- The average number of evaluations $E\{T_{k\cdot n,k\cdot m}\}$ needed to find a desirable solution increases with increasing *neutrality factor* k and has the limit

$$\lim_{k\to\infty} E\{T_{k\cdot n,k\cdot m}\} = \frac{n}{m}$$

- For $k \geq 1$ it holds

$$E\{T_{(k+1)\cdot n,(k+1)\cdot m}\} - E\{T_{k\cdot n,k\cdot m}\} = \frac{n-m}{(1+km)(1+m+km)} > 0$$

- Therefore, in the considered NFL scenario enlarging the genotype space by adding redundancy without a bias does not considerably increase the average number of iterations needed to find a desirable solution if initially m is large enough. In the worst case when initially just one element encodes a desirable solution ($m = 1$), the deterioration of the average search performance is bounded by a factor of two.

◇ (Corne and Knowles, 2003) show that free lunches can arise in multiobjective optimization and that the NFL does not generally apply. The reason is given by the fact that multiobjective optimizers usually combine a *generator* (the 'algorithm' in the NFL sense) and an *archiver* which filters the sample generated by the algorithm in a way that undermines the NFL assumptions. Thus, the conclusion that some multiobjective approaches are better than others.

◇ (Whitley and Watson, 2005) point out that, since the set of all possible functions is uncountably infinite and the set of all possible programs is only countably infinite, the set of all possible cost functions represents a small subset of the set of all possible functions. Therefore, the space of all possible discrete function is largely composed of incompressible functions[11] (ie., functions that cannot be described by any algorithm), making the NFL theorem less relevant.

[11] Here the term *compression* refers to the concept from algorithmic information theory that describes the *Kolmogorov complexity* of a string s. A string is incompressible if it cannot be compressed, ie. encoded using fewer bits than its original representation, because it contains too few repeating sequences. Russian mathematician Kolmogorov proved that some strings are incompressible using any algorithm. For more information, see (Grunwald and Vitányi, 2004)

Then, they argue that search is often not blind, in the sense that generic methods can exploit application-specific operators and representations while still maintaining their generality by being able to solve many different instances of the same problem.

Finally, they prove by contradiction that NFL does not hold for NP problems since otherwise such a result would imply $P = NP$:

> "if No Free Lunch holds for any NP-complete problem, then it immediately follows that no algorithm is better than random enumeration on the entire class of NP-complete problems (because of the existence of a polynomial-time transformation between any two NP-complete problems). However, this would also prove that $P \neq NP$, since it would prove that no algorithm could solve all instances of an NP-complete problem in polynomial time".

- (Wolpert and Macready, 2005) show that there are free lunches in coevolutionary contexts when self-play is involved (two "players" working together to produce a "champion") as the objective function depends on the moves of both players, thus invalidating the NFL hypothesis.
- (Auger and Teytaud, 2010) investigate the extension of NFL theorems to countably infinite and uncountably infinite domains and show that NFL does not hold for continuous domains.

With regard to genetic algorithms, convergence analysis proved that the canonical GA can converge to the global optimum:

- (Hartl, 1990) derived a global convergence proof for a class of genetic algorithms that incorporate certain aspects of simulated annealing (each generation, some randomly selected individuals can survive regardless of their fitness).
- (Rudolph, 1994b) showed that the canonical genetic algorithm with mutation, crossover and proportional selection will converge to the global optimum only when elitism is used to always maintain the best solution in the population.
- (Greenhalgh and Marshall, 2000) also showed (using Markov chain theory) that global convergence can be guaranteed with a prespecified level of confidence, giving also a lower bound of the necessary number of generations.
- (Woodward and Neil, 2003) demonstrate that there is a free lunch if a non-uniform many-to-one mapping exists between the space of programs (genotypes) and the space of functionality (phenotypes).
- (Poli et al., 2009) show that for symbolic regression problems the set of fitness functions $\mathcal{F}$ is c.u.p. if and only if a particular set of geometric constraints are satisfied. They identify situations where a free lunch does indeed exist for genetic programming.

2.4.2 Summary

In conclusion, the NFL theorems may appear to have limited practical significance but are in fact very helpful in giving a more accurate picture of what can be realistically expected from optimization algorithms, especially in areas where a trade-off between generality and specificity needs to be achieved.

They have powerful implications especially in the case of evolutionary algorithms, where it is shown that neutrality given by the non-injectiveness of the genotype-phenotype map is a necessary ingredient of self-adaptation and can improve performance (Igel and Toussaint, 2003a).

Using the sharpened NFL, (Igel and Toussaint, 2003a) showed that redundancy of the genotype space does not greatly increase the average number of iterations needed to find a solution if the initial population is large enough. Furthermore, the neutrality property implies that the genotype does not encode just the phenotype but also information on further explorations. This last aspect is particularly important (as we will see in Section 4.4) as it shows that neutrality is directly correlated to the algorithm's ability to evolve good solutions.

3 Evolutionary Computation

3.1 Introduction

3.1.1 Evolution through Natural Selection

> "I have called this principle, by which each slight variation, if useful, is preserved, by the term Natural Selection." (Charles Darwin, *The Origin of Species*, 1859)

Modern evolutionary theory as we know it today began in the 19th century with the works of Charles Darwin, who theorized that all species had developed from common ancestors and had become different from one another under the effects of a process he called *natural selection*, and Gregor Mendel, considered the father of genetics, who discovered the rules of heredity.

The idea that useful random changes might be preserved was traced by Darwin to ancient times, when Aristotle considered the question whether different traits appeared accidentally and survived due to their usefulness:

> "So what hinders the different parts (of the body) from having this merely accidental relation in nature? As the teeth, for example, grow by necessity, the front ones sharp, adapted for dividing, and the grinders flat, and serviceable for masticating the food; since they were not made for the sake of this, but it was the result of accident. And in like manner as to other parts in which there appears to exist an adaptation to an end. Wheresoever, therefore, all things together (that is all the parts of one whole) happened like as if they were made for the sake of something, these were preserved, having been appropriately constituted by an internal spontaneity; and whatsoever things were not thus constituted, perished and still perish." (Aristotle, *Physicae Auscultationes*, lib.2, cap.8, s.2)

In nature, the concept of usefulness counts towards the one single goal of every living organism: to survive long enough to reproduce. Under Darwin's theory, useful changes that improve an organism's ability to survive and to produce offspring are preserved and transferred to the next generation of organisms. A central assumption in this theory was of course the idea that these useful changes were heritable and could be transferred from

parents to offspring. But in 1859, the year Darwin put forward his theory of evolution by natural selection, there existed no clear knowledge about how these changes or traits might actually be inherited. Darwin himself came up only years later in 1868 with a hypothetical theory called *pangenesis* that speculated the existence of *gemmules* or "particles of inheritance", responsible for the inheritance of certain physical characteristics.

Unfortunately, Darwin was not aware of Gregor Mendel's inheritance experiment on pea plants: Mendel provided an explanation for the underlying mechanism behind heredity which later became known as Mendel's Laws of Inheritance. In Mendel's vision, inheritance worked through the transfer of discrete inherited units that determined an organism's visible traits. Mendel also showed that recessive traits could be hidden for several generations. Although Mendel did not use the term *gene*, he saw the distinction between what we now call the genotype and phenotype. Mendel's work published in 1865 went largely unnoticed until the very beginning of the 20th century, when it was rediscovered by botanists Hugo de Vries, Carl Correns and Erich von Tschermak.

The term gene was coined in 1909 by Danish botanist Wilhelm Johannsen who used it to describe the basic units of heredity, while the word *genetics* originates from the Greek γενετικόσ (genetikos), derived from γένεσισ (origin) — the origins of the traits passed down from parents to offspring. Johannsen was also the first to make a clear distinction between genotype and phenotype, proposing in his 1911 paper titled *"The Genotype Conception of Heredity"* the idea that genotypes determine phenotypes under the influence of the environment. The genotype-phenotype distinction later became the basis for the field of developmental biology which studies the mechanisms through which genotypes determine phenotypes.

3.1.2 A Short History of Population Genetics

In the beginning of the 20th century, Lamarckian and orthogenic theories were used by the majority of biologists to explain the mechanisms of evolution which had lead to the complexity of natural life. These outdated and inaccurate theories were eventually invalidated by new discoveries in the field of genetics most prominently by geneticists Theodosius Dobzhansky, Sewall Wright, Sir Ronald Aylmer Fisher and John Burdon Sanderson Haldane.

Theodosius Dobzhansky, an Ukrainian-born American scientist, played an important role in the shaping of the new theory, defining evolution as "a change in the frequency of an allele within a gene pool". In his work he supported the idea that natural selection takes place through genetic mutations. He showed that populations in the wild had large amounts of genetic diversity and clear differences between subpopulations (as opposed to what was generally assumed at the time). Dobzhansky was also familiar with the theoretical research done by the population geneticists such as Wright or Fisher and used mathematical models to explain how mutation rates vary with different population sizes. His more accessible treatment of the mathematics of population genetics, put into perspective in the context of

macro-evolutionary patterns observed in nature have made his 1937 book *"Genetics and the Origin of Species"* a popular, highly influential and far-reaching work.

Sewall Wright was an American geneticist considered to be one of the founders of population genetics, along with Ronald Fisher and J. B. S. Haldane. Wright's major contribution was the mathematical theory of genetic drift (known also as the Sewall-Wright effect) and the concept of effective population size, roughly defined as the number of individuals in a population who contribute offspring to the next generation. Wright also came up with the idea of adaptive landscapes, expressing the mean fitness of a population as a function of allele frequencies at one or more loci. Wright and Fisher held conflicting views of evolution: according to Wright, evolution resulted from gene interactions and the interaction of drift with selection; according to Fisher, evolution was mass selection acting on relative fitnesses in large populations. The longstanding argument between Wright and Fisher was considered to be a "central, fundamental and very influential" for the development of modern evolutionary theory (Skipper, 2002).

Sir Ronald Fisher was a British statistician, mathematician and biologist who established himself as one of the founders of the modern synthesis through his major contributions in the fields of statistics, quantitative genetics and evolutionary theory. His close friend E. B. Ford describes his brilliant achievements:

> "Fisher was far ahead of his contemporaries, so far, indeed, that when his epoch-making book, *Statistical Methods for Research Workers*, was published in 1925, it did not receive one favorable review. At the time of his death in 1962, it was in its 14th edition, with reprints, and had been translated into six languages."

In his 1930 book, *The Genetical Theory of Natural Selection*, Fisher explained evolution as natural selection acting on the variation produced by combinations of discrete genes, changing the allele frequencies in a population. Fisher was the first to provide an explanation (known as "Fisher's principle") why the sex ratio in most sexually reproducing species is approximately 1:1. He showed that the probability that a mutation increases an organism's fitness is inversely proportional to its magnitude, and proved that more variation increases the chances of survival, therefore larger populations are more likely to survive. He also founded the field of quantitative genetics and contributed mathematical tools and methods for calculating gene frequencies, estimating genetic linkage and explaining Mendelian inheritance within the continuous, gradual context of evolution. Together with geneticist E. B. Ford, Fisher showed that natural selection was a much stronger force than had been assumed at the time and explained the presence of polymorphism in nature due to selection.

J. B. S. Haldane was another British scientist (later naturalised Indian) with important contributions to statistics, biometry and population genetics. Haldane demonstrated genetic linkage in mammals in his 1915 paper *"Reduplication in mice"*. Along with Sewall Wright and Ronald Fisher, Haldane was one of the three major contributors to the mathematical

theory of population genetics, using maximum likelihood methods for the estimation of human linkage maps and developing mathematical models for allele frequency changes at a single gene locus. In his ten-paper series entitled *"A Mathematical Theory of Natural and Artificial Selection"* and later in his book *"The Causes of Evolution"*, Haldane explained the mathematics behind natural selection and Mendelian inheritance in terms of changes in gene frequencies and the interaction between selection, mutation and migration.

By the early 1930s, the works and efforts of the leading population geneticists have resulted in a consistent mathematical framework which integrated natural selection with Mendelian genetics. Thus the new discipline of Population Genetics was born, becoming the core of the modern evolutionary synthesis. Scientists had managed to identify the main factors that influence evolution and explain their mechanisms within a mathematical framework, although the relative importance of these factors remained subject to debate.

An understated aspect of the new framework was that it did not represent merely a confirmation and formalization of previous Darwinian and Mendelian hypotheses. It represented a paradigm shift in its own right, shifting focus from individual organisms and speciation at the level of individuals to processes happening at the level of genes and populations. The change in perspective did not happen without opposition. In particular, reconciliation between the continuous microevolution determined by the accumulation of small genetic changes and the macroevolutionary leaps observed by paleontologists in the fossil records was still not fully achieved.

3.1.3 The Modern Synthesis

> "...a study of the effects of genes during development is as essential for an understanding of evolution as are the study of mutation and that of selection."
> (Julian Huxley, *Evolution: The Modern Synthesis*, 1942)

The so-called *modern synthesis* was a mid-century effort by the leading biologists of the time to create a unifying view of evolution, merging ideas from different fields of biology such as genetics, botany, ecology, morphology, paleontology and systematics. This synergistic approach was necessary in order to bridge the gap between the theoretical work of mathematicians and statisticians and the experimental observations of geneticists, naturalists and paleontologists.

The name was coined by evolutionary biologist Sir Julian Sorell Huxley (brother of famous novelist and philosopher Aldous Huxley) in his book entitled *"Evolution: The Modern Synthesis"* (Huxley, 1942). Intimately familiar with the works of his predecessors, Huxley explained with great clarity the ideas of the time that evolution happens through changes in gene frequency due to genetic drift, gene flow and – most importantly – natural selection.

Built on the foundation of population genetics, the modern synthesis was further extended most notably by biologists Ernst Mayr, Gaylord Simpson and Ledyard Stebbins.

Mayr invented the concept of a biological species and developed models to explain the occurrence of speciation. Simpson — a paleontologist, used fossil evidence to validate the modern synthesis. He supported the claim that small, continuous, gradual change described as microevolution could be extrapolated to explain the macroevolution observed by paleontologists. In his work *"Tempo and Mode in Evolution"*, he showed that evolution can take place at varying rates – very fast, average and very slow – and that differing rates yield different patterns of evolution. Stebbins was a botanist whose most important work *"Variation and Evolution in Plants"* helped integrate botanical science into the modern evolutionary synthesis. Stebbin's work explained evolutionary mechanisms in plants and provided the conceptual framework for the field of plant evolutionary biology.

By the end of the 1950s, the modern synthesis had become the fundamental framework of evolution, containing several important ideas:

- Evolution is gradual, through the accumulation of small genetic changes
- Change is determined by natural selection acting on genetic variation
- Other mechanisms are also present, in particular gene flow and genetic drift
- Events identified in the fossil record are consistent with evolution through natural selection
- There is a clear distinction between genotypes and phenotypes
- The environment affects phenotypes but not genotypes

With the modern synthesis, the scientists of the time had made the best of the knowledge they had available. However, in the 1950s the DNA, as well as many of the biological mechanisms present inside the cell and inside the gene had not yet been discovered, and biologists had yet to understand the role of genes in development.

3.1.4 Developmental and Molecular Genetics

In current times, the field of genetics deals predominantly with the functioning and behavior of genes. We know now that a gene is a DNA sequence that codes for a known cellular function or process. Physically speaking, a gene is made up of molecules called *nucleotides* that are named according to the four nitrogenous bases they contain: cytosine (C), guanine (G), adenine (A) and thymine (T). Therefore, a concrete gene is given by a string of nucleotides such as "CTGGAG". Mutations occur at gene level when nucleotides are inserted, changed or deleted from the DNA sequence.

Under environmental pressure, mutations can either have an immediate effect or they can accumulate over time, leading to evolutionary change. Adaptation is considered to be the consequence of genetic change, manifesting itself at the level of the phenotype through the acquisition of traits that enhance the fitness and survival of individuals. Environmental factors can also turn genes on and off and affect how cells read genes, therefore affecting the

phenotypic traits of the organism. An organisms' phenotype and behavior are influenced by genetic as well as environmental factors[1].

In its early stages, evolutionary theory concerned itself solely with the mechanisms of gene expression and the correlation between genes and phenotypes, with no special emphasis on the developmental aspect and the causal relationships within the evolutionary process. However, at the beginning of the 21st century, despite modern tools and techniques such as whole genome sequencing, it is still not possible to establish a clear relationship between genotypic and phenotypic variation.

Understanding the mapping of genotypes to phenotypes represents a core aspect of modern biological research. Despite a historical systematic bias towards linear causation schemes in biology, the post-genomic era marks an increasing focus towards a systems approach trying to clarify the generative properties of genetic variation and how phenotypic variation is generated given a genetic background (Rodríguez-Mega et al., 2015).

3.2 Evolutionary Search Methods

The idea of artificial evolution appeared quite early in the field of computer science. We find it in the work of Alan Turing, where it was used to outline a teaching process for machines (Turing, 1950), or in the work of John von Neumann on self-reproducing automata (Neumann, 1966)[2].

Metaheuristics that implement various aspects of biological evolution belong to the category of Evolutionary Computation (EC). These algorithms iteratively improve a population of solution candidates called *chromosomes* or *individuals* through the repeated action of recombination[3] and fitness-based selection.

Algorithm 1 shows a high-level algorithmic outline of the evolutionary process. Evolutionary methods with the general structure given in Algorithm 1 differentiate themselves from one another by the way solution candidates are encoded (the chromosome representation) and by the concrete implementations of the recombination and selection operators. The recombination step is usually realized by multiple recombination operators that can be applied with different probabilities.

The most popular EC metaheuristics include evolutionary programming (Fogel and Fogel, 1996), evolution strategies (Rechenberg, 1971; Schwefel, 1974), genetic algorithms

[1] The common example illustrating the environment's importance is two seeds of genetically identical corn, one placed in a temperate climate and one placed in an arid climate. The one in the arid climate only grows to half the height of the one in the temperate climate, due to lack of water and nutrients in its environment.

[2] For a more in-depth and comprehensive account of evolutionary algorithms we turn the reader to (Back et al., 1997).

[3] We describe recombination as the action of producing new genetic variation from the existing variation, ie., creating new solution candidates from the old ones.

```
initialization: random starting point or collection of points in the solution space;
fitness evaluation: assign fitness values to the initial solutions;
while stopping criteria not satisfied do
    parent selection: select parent individuals for recombination based on fitness;
    recombination: generate new child individuals from the selected parents;
    fitness evaluation: calculate objective values for each solution;
    offspring selection: fill mating pool with individuals based on their fitness;
return: current solution or best solution from a population of solutions;
```

Algorithm 1: General workflow of evolutionary methods

(Holland, 1975), differential evolution (Storn and Price, 1997) and genetic programming (Koza, 1992).

3.2.1 Evolution Strategies

Evolution Strategies (ES) were originally developed in the late 60s from a set of rules for driving a physical system towards an optimal state (Beyer and Schwefel, 2002). The rules in their original formulation were the following:

1) Change all variables at each step, mostly slightly and at random
2) If the new set of variables does not diminish the goodness of the device, keep it, otherwise return to the old status

Recognizing the efficiency of these rules in optimizing noisy and multimodal processes, as well as their similarity with the process of evolution (where the first rule models mutation and the second rule models natural selection), Rechenberg and Schwefel coined the name *evolution strategy* and shifted their focus on the new method in their respective dissertations (Rechenberg, 1971) and (Schwefel, 1974).

Initially, ESs worked with a "population" of just two individuals, one parent individual and one descendant obtained via mutation, thus being named (1 + 1)-ES. Mutation was realized by applying a vector of normally distributed random numbers from $\mathcal{N}(0, \sigma^t)$, such that a new solution was given by:

$$x'(t) = x(t) + \mathcal{N}(0, \sigma^t)$$

The values of $\sigma^t \in \mathbb{R}^n$ determine the so-called "mutation strength" and can either remain constant over time or be dynamically adjusted using Rechenberg's *1/5 success rule*, developed analytically by calculating the convergence rates (Rechenberg, 1973)

$$\sigma^{t+n} = \begin{cases} c_d \cdot \sigma^t & \text{if } p_s^t < 1/5 \\ c_i \cdot \sigma^t & \text{if } p_s^t > 1/5 \\ \sigma^t & \text{, if } p_s^t = 1/5 \end{cases}$$

where p_s^t is the frequency of successful mutations and step sizes $c_d, c_i \in \mathbb{R}$ are control parameters of the evolution strategy. The 1/5 rule can be stated as follows:

> The ratio of successful mutations to all mutations should be 1/5. If it is greater than 1/5, increase the standard deviation, if it is smaller, decrease the standard deviation.

(Droste et al., 2002) show that the (1+1)-ES can optimize linear functions in $O(n \ln n)$ expected time with mutation rates of size $\Theta(1/n)$ but will need exponential time for polynomials of degree 2 and unimodal functions, where n is the length of the parameter vector.

Rechenberg was also the one to come up with the idea of having a population of $\mu > 1$ individuals, and this version was named $(\mu + 1)$-ES by Schwefel. The $(\mu + 1)$-ES also benefits from a recombination operator that generates an offspring from two randomly selected parents from the population. If the offspring is fit enough it will replace one of the parents, thus keeping the population size constant. (Rechenberg, 1971) showed that crossover can significantly speed up the evolution. In the case of ESs, one disadvantage of having a recombination operator and a population of individuals is that self-adaptation of step sizes is no longer possible since offspring with reduced mutation variances are always preferred (Bäck et al., 1991).

The intrinsic tendency of the $(\mu + 1)$-ES to reduce mutation strength led Schwefel to introduce two new versions of ESs (Schwefel, 1981):

- The $(\mu + \lambda)$-ES, in which $\lambda \geq 1$ descendants are created at every generation and the worst λ out of all the $\mu + \lambda$ individuals are discarded.
- The (μ, λ)-ES, in which the selection takes place among the λ offspring only and the parents are completely replaced by the offspring. Obviously in this strategy $\lambda > \mu$.

When ρ parents are involved in the creation of one offspring, the ES is described using the notation $(\mu/\rho^{+}_{,}\lambda)$-ES.

Another motivation for the $(\mu+\lambda)$-ES and (μ, λ)-ES was the possibility to simultaneously adapt the individual components of the mutation vector σ^t by adding it as part of the individual's genotype. The mutation operator was adapted to handle this change (Schwefel, 1995):

$$
\begin{aligned}
a'^t_i t &= r(P^t) \\
a''^t_i &= m(a'^t_i) = (x''^t, \sigma''^t) \\
\sigma''^t &= \sigma'^t \cdot e^{\mathcal{N}(0,\Delta\sigma)} \\
x''^t &= x'^t + \mathcal{N}(0, \sigma''^t)
\end{aligned}
$$

In the above equation, r represents the recombination operator and m represents the mutation operator which in the first step adjusts the σ^t vector according to the normal

distribution $\mathcal{N}(0, \Delta\sigma)$ (where $\Delta\sigma$ is the mutation step size meta-parameter). Then mutation is applied on the part of the individual which encodes the problem with the self-tuned mutation strength. The idea behind the use of the exponential for the calculation of σ''^t is to permit a wider variation range for the mutation strength, under the implicit assumption that good individuals will pass on their "good" settings for the mutation strength to their offspring via recombination, causing the search to take larger steps towards the optimum.

In a further analysis of the two member ES using a sphere model of the objective function, Schwefel also derived a more precise formulation of the 1/5 success rule:

> After every n mutations, check how many successes have occurred over the preceding $10n$ mutations. If this number is less than $2n$, multiply the step lengths by the factor 0.85; divide them by 0.85 if more than $2n$ successes occurred.

Another significant improvement comes from (Hansen and Ostermeier, 2001) who introduced a new step-size control method for ES in which they use step history information (accumulated in a so-called evolution path) to automatically tune the covariance matrix of the multivariate distribution from which new candidate solutions are sampled, such that the algorithm is more likely to perform mutation steps in the right direction with an adaptive step size. The new approach, entitled Covariant Matrix Adaptation Evolution Strategy (CMA-ES) is a state-of-the-art evolutionary algorithm for unconstrained or bounded constraint optimization problems.

For a more in-depth look at evolution strategies, the reader may also consult (Beyer and Schwefel, 2002) and (Hansen et al., 2013).

3.2.2 Evolutionary Programming

Evolutionary Programming (EP) represents a similar optimization approach to ES, developed independently by (Fogel, 1964) initially for the optimization of finite state machines, and performed according to the following steps:

1) *Inititalization*: The initial population of μ machines $\mathbf{P}_i = (N_i, S_i, \mathbf{L}_i, \mathbf{O}_i, \lambda_i), \forall i \in \{1, 2, ..., \mu\}$, where N_i represents the number of states in the machines, selected at random uniformly from $\{1, 2, ..., N_{max}\}$, S_i represents the randomly selected start state, $\mathbf{L}_i$ the number of links and $\mathbf{O}_i$ the output symbols.

2) *Fitness evaluation*: A score describing how well the FSM was able to solve the problem at hand (prediction or control).

3) *Mutation*: In the original formulation, mutation consisted of FSM modification operations like adding a state, deleting a state, reassigning a start state or reassigning a link or an output symbol. The number of mutation operators M_i to be applied

was determined by sampling a Poisson random variable with mean parameter λ_i^i obtained from the parent's mean parameter λ_i:

$$\lambda_i^{'} = \lambda_i + 0.5\mathcal{N}(0, 1)$$

4) *Fitness evaluation*

5) *Selection*: For population sizes smaller than 10, the better half of the population according to fitness was chosen as parents for the next generation. For larger population sizes, tournament selection was used.

EP was refined and adapted to the continuous optimization domain by (Fogel, 1991) and shares a number of features with ES, such as the real-valued representation, normally distributed random mutations and self-adaptation of the strategy parameters. A scaling parameter α was introduced for the self-adaptation of strategy parameters.

$$x_i' = x_i + \mathcal{N}(0, \sigma_i)$$
$$\sigma_i' = \sigma_i + \alpha \cdot \sigma_i \cdot \mathcal{N}(0, 1)$$

If any value σ_i becomes nonpositive, it is reset to a small value ε (Fogel, 1991). In the experiments of (Fogel et al., 1995) α was set to a value of 0.01.

One notable difference though between ES and EP is the absence of a recombination operator in EP. Furthermore, EP implements a softer kind of probabilistic selection which makes it a slightly weaker optimization method than ES, according to (Bäck et al., 1993).

Furthermore, ES use the mutated strategy parameters for modification of object variables, while EP first modifies object variables and mutates strategy parameters in the second step, leading to a delayed effect of strategy parameter changes (Bäck, 1996).

3.2.3 Differential Evolution

Differential Evolution (DE) (Storn and Price, 1997) is a parallel direct search method that uses a population of n D-dimensional parameter vectors $x_{i,G}$, where $i = 1, 2, ..., n$. DE differs from other methods in the way new solution candidates are generated, by adding a weighted difference vector between two population members to a third member.

Different DE strategies are usually denoted by $DE/x/y/z$ where x represents the way of producing mutants, y represents the number of difference vectors involved in crossover and z represents the crossover type. For example, a $DE/rand/1/bin$ will choose a random vector to be mutated, use one difference vector and a binomial crossover scheme.

The mutation operator in DE has received a lot of attention leading to the development of several mutation variants, described below. Several mutation schemes are possible:

1) **DE/rand/1**

$$v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G})$$

2) **DE/best/1**

$$v_{i,G+1} = x_{best,G} + F \cdot (x_{r_1,G} - x_{r_2,G})$$

3) **DE/current-to-best/1**
This mutation scheme incorporates an extra variable λ to control "greediness"

$$v_{i,G+1} = x_{i,G} + \lambda \cdot (x_{best,G} - x_{i,G}) + F \cdot (x_{r_2,G} - x_{r_3,G})$$

4) **DE/rand/2**

$$v_{i,G+1} = x_{i,G} + F \cdot (x_{r_1,G} - x_{r_2,G}) + F \cdot (x_{r_3,G} - x_{r_4,G})$$

5) **DE/best/2**

$$v_{i,G+1} = x_{best,G} + F \cdot (x_{r_1,G} - x_{r_2,G}) + F \cdot (x_{r_3,G} - x_{r_4,G})$$

In the above, $r_1, r_2, r_3, r_4 \neq i$ represent random integers chosen from the interval $[0, n-1]$, G represents the generation number and F is a real constant factor which controls the amplification of the differential variation.

Crossover (parameter mixing) was also introduced as a way to increase diversity (and make larger steps in the solution space)

$$u_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G})$$

DE uses two different crossover operators:

1) **Binomial crossover** This type of crossover selects each component to be used in the offspring from the mutated vector according to independent Bernoulli (binomial) trials. The probability that a component is mutated is $p_m = CR(1-1/n)+1/n$ where n is the number of components and CR is the crossover rate.

2) **Exponential crossover** For this crossover, CR controls not only the probability for mutation but also the number of components to be mutated. We denote the bound for the number of mutated components with L and give the probability distribution for the case where we have h mutated components:

$$P(L = h) = \begin{cases} (1 - CR)CR^{h-1} & \text{if } 1 \leq h < n \\ (CR)^{n-1} & \text{if } h = n \end{cases}$$

As far as selection is concerned, newly generated individuals are accepted into the new population based on the greedy criterion: the new trial vector replaces the target vector if it yields a better cost function value.

3.2.4 Genetic Algorithms

Genetic Algorithms (GA) were introduced by (Holland, 1975) in their book "Adaptation in Natural and Artificial Systems". Although initially developed as a tool for the study of a more general theory of adaptive systems, GAs have found many applications in real-world optimization problems from many technical and engineering fields. Especially suited for combinatorial optimization problems, GAs use a binary encoding and make explicit use of genetic operators called crossover and mutation for the generation of new solutions. In analogy with nature, the bit positions in the binary chromosome are called *loci*, the variable at a given *locus* is a *gene* and its value it an *allele*.

As a theoretical justification for the algorithm's optimization capability, Holland developed an equation called the "Schema Theorem" which demonstrates how the evolutionary search is guided by the selection mechanism towards "promising regions of the search space". Roughly speaking, the Schema Theorem describes the GAs ability to explore hyperplanes in the solution space and gradually close in to feasible regions where the optimum might be found. It does so by factoring in the disruptive effects of crossover and mutation on the distribution of certain patterns (as we will see bellow) in the bit string population of individuals, under a proportional selection scheme.

Recombination in Genetic Algorithms

In a GA, recombination takes place through crossover and mutation. Crossover takes two parent bit string individuals and exchanges parts of their chromosomes in order to create new child individuals. Mutation creates new individuals by making random changes in a parent bit string. The main exploration tool is the crossover operator, through which a new child individual is created from two parent individuals, in a manner similar to the sexual reproduction from biology. Crossover returns a new bit string obtained from the combination of two parent bit string individuals.

The three most common crossover operators used by GAs are the one-point, two-point and uniform crossover.

1) **One-point crossover** The one-point crossover takes two bit strings $\vec{u}$ and $\vec{v}$ of length l and picks a random integer c in the range $[1, l]$. Then, two new child individuals $\vec{u'}$ and $\vec{v'}$ are produced by swapping between $\vec{u}$ and $\vec{v}$ the bit values from positions 0 to c.

2) **Two-point crossover** Like its name suggests, the two-point crossover takes two bit strings $\vec{u}$ and $\vec{v}$ of length l and randomly chooses two numbers c, d in the range $[1, l]$. Then, it swaps the bit values between $\vec{u}$ and $\vec{v}$ on the portion defined by the two indices c and d.

3) **Uniform crossover** The uniform crossover operates according to a probability p of swapping bit values between the two parents. It iterates over the two parent bit

strings and if p is greater or equal than a random number generated uniformly on the interval [0, 1], then it swaps the bit values. In this context, p is also called a "mixing ratio" as it determines how many genes come from the first parent and how many from the second parent.

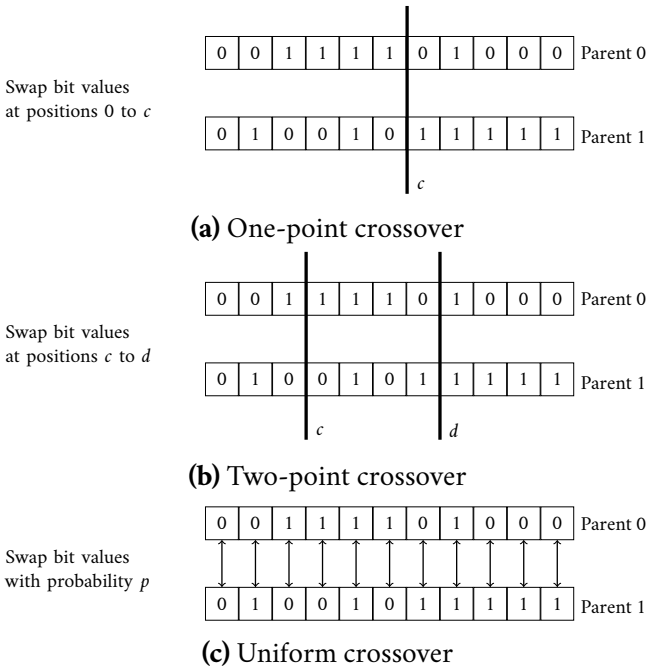


Figure 3.1: Crossover operators

Holland's introduction of crossover as the main GA recombination operator (as opposed to mutation which was the main operator used by evolution strategies) was based on the idea that two good parents will have a good chance of producing an even better offspring. However, crossover alone cannot guarantee progress as new solutions will always be inside the region of the solution space defined by the hypercube bounded by the initial binary vectors. If the global optimum is outside of this region, the algorithm will not be able to find it. Therefore, the algorithm will be sensitive to population size and random initialization.

To minimize the risk of premature convergence and improve the performance of the genetic search, mutation needs to be introduced in the population as a diversity bringer. Here, the concept of diversity has the same meaning as in nature: new genetic material introduced by mutation will help the algorithm explore previously inaccessible regions of the problem space. The mutation operator changes one or multiple bit values in the individual according to a predefined probability p.

Selection in Genetic Algorithms

The selection operator is responsible for deciding which child individuals make it into the next generation based on their relative fitness. Many studies of GA selection are available in the literature (Blickle and Thiele, 1995; Bäck, 1994; Goldberg and Deb, 1991; Miller and Goldberg, 1995). We present the two main types of selection:

Fitness-proportionate Selection In fitness-proportionate selection, each individual is assigned a relative selection probability based on its relative fitness compared to the cumulated fitness of the population. An individual i from a population of size n will have a chance of being according to its fitness f_i:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

Examples of fitness-proportionate selection are roulette-wheel selection (Golberg, 1989; Holland, 1975) and stochastic universal sampling (Baker, 1987; Grefenstette and Baker, 1989).

Ordinal Selection Ordinal selection methods fill the recombination pool with individuals selected from groups of the population. Tournament selection (Miller et al., 1995), picks s individuals from the population at random and puts them together in a group where they compete against each other for a place in the recombination pool. Truncation selection, used in the Breeder GA (Mühlenbein and Schlierkamp-Voosen, 1993) tries to fill the recombination pool using only the best $T\%$ individuals in the population.

Holland's Schema Theorem

Holland's schema theorem provides a theoretical explanation for the GA's ability to produce increasingly good solutions by factoring the effects of crossover, mutation and fitness-proportionate selection into an equation which explains the increase in average quality in the population of binary vectors.

A *schema* (pl. *schemata*) H is a string of symbols from the alphabet $\{0, 1, *\}$, where '$*$' represents a wildcard symbol that can be matched by either a 0 or a 1 ($* \in \{0, 1\}$). Schemata can be seen as templates which describe sets of strings on the alphabet $\{0, 1\}$. For example, the schema 1*0* describes the set of all strings which have a 1 at position 1 and a 0 at position 3. Obviously, this set will contain four strings, 1000, 1001, 1100 and 1101. A schema H can be described by the following properties:

- The *order* $O(H)$ represents the number of fixed positions inside the schema
- The *defining length* $\mathcal{L}(H)$ represents the distance between the first and last fixed positions

- ◇ The *average fitness* $f(H)$ is given by the average fitness of all the strings matching the schema

Using these properties, the Schema Theorem states that short, low-order schemata with above average fitness increase exponentially in successive generations. This schemata were also called *building blocks* as they were believed to have a big contribution to the optimization results. The reasoning is as follows: the children of fit individuals will likely have a selective advantage unless disrupted by crossover or mutation. The probability of disruption can be described within the schema itself as the probability that one of the bits in the fixed positions is perturbed. Therefore, the lower the order (less fixed bit positions that can be disrupted), the lower the probability. Similarly, the shorter the defining length, the less chance that a schema is split by crossover.

The disruption probability in the case of crossover depends on the number of positions N where crossover can take place, and the schema defining length $\mathcal{L}(H)$:

$$\Pr\{D_c\} = p_c \frac{\mathcal{L}(H)}{N-1}$$

In the case of mutation, the probability for disruption depends on the schema order and can be expressed as

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

For $p_m \ll 1$, the schema survival probability for mutation may be approximated as $1 - p_m^{O(H)}$. Using the equations above and ignoring the small cross-product terms, the total probability of disruption can be expressed as

$$p = \frac{\mathcal{L}(H)}{N-1} p_c + O(H) p_m$$

where p_c, p_m are the probabilities of crossover and mutation. The expected frequency of a schema H is described by the following inequality:

$$E\left(m(H, t+1)\right) \geq \frac{m(H,t) f(H,t)}{\overline{f}(t)} (1 - p)$$

where N is the number of bits (length of the representation), p_c and p_m are the probabilities of crossover and mutation, $m(H, t)$ is the number of strings belonging to schema H at generation t, $\overline{f}(t)$ is the average fitness at generation t and $f(H, t)$ is the average fitness of schema H at generation t.

Holland defines the GA *intrinsic parallelism* in terms of schemata by observing that the sampling of a new chromosome yields information about the sampling averages of each of the 2^l schemata of which it is an instance. This information can be useful under the assumption that correlations exist between the various instances of a schema. This particular assumption represents the main argument for the effectiveness of the genetic search.

In practice, the schema theorem may fail to account for sampling artifacts in finite GA populations, especially in the context of multimodal solution spaces where selection may steer the search in the wrong direction. Furthermore, a number of dissenting views regarding Holland's schema theorems have been expressed by other researchers dealing with the mathematical and statistical analysis of GAs and their convergence.

Criticism of the Schema Theorem

Grefenstette (Grefenstette and Baker, 1989) claims that the introduction of schemata only creates artificial confusion. They criticize the mainstream interpretation of the schema theorem describing it as a tautology:

> "The estimate of the fitness of a schema is equal to the exact fitness in very simple applications only. Therefore interpretations using the exact fitness cannot be applied in connection with the schema theorem. But if the estimated fitness is used in the interpretation, then the schema theorem is almost a tautology, only describing proportional selection."

Moreover, Grefenstette objects to the fact that the schema theorem estimates only the disruption probability. Their alternate explanation of the GA search strategy is by analogy with scatter search, with selection and crossover leading to a concentration of individuals into promising areas.

Radcliffe (Radcliffe, 1991) showed in the context of real-valued problems that gathering information about the performance of any subset of the chromosomes provides no information about the performance of the remaining structures – even when the schema theorem is obeyed. Therefore, the search could not be effective except by chance because the schemata would not relate chromosomes with correlated performance.

Vose (Voset and Liepinsl, 1991) conjenctured, using computer simulations of GAs modeled as dynamical systems in a high-dimensional Euclidean space, that GA populations often alternate between generations of relative stability and periods of sudden rapid evolution. Vose explains this phenomenon known in biology as punctuated equilibria in terms of properties of the selection ("focusing operator") and recombination ("diffusing operator") operators. One of their conclusions is that "the emergence of growth of a string having greater fitness typically requires events less probable than does focusing a population towards a prevalent high-fitness string whose dominance is not interfered-with by finite population effects". In other words, the diffusion property of the recombination operator under selection explains the search better than the schema theorem.

Rudolph (Rudolph, 1994a) also stated that schema theorems have limited explanatory power. He showed that although the schema theorem may give some clues about the dynamics of the search, it does not imply convergence to the global optimum (nor does it imply non-convergence).

Altenberg (Altenberg, 1995) analyzed GAs using Price's theorem[4] and found that the schema theorem has no implications for how well a GA is performing. The main point of criticism is that the schema theorem does not account for the intuitive idea that offspring with above-average fitness can be produced by parents belonging to schemata of above-average fitness. In other words, he found that an increase in the frequency of schemata of above-average fitness says nothing about GA performance. Expressing the schema theorem in terms of Price's theorem, Altenberg shows that GA performance depends on the algorithm's ability to increase the upper tail of the fitness distribution of the population.

Mühlenbein (Mühlenbein, 1997) describes GAs as parallel random search algorithms with centralized control represented by the selection schedule. He then questions the utility of the schema theorem as it only accounts for proportional selection and it regards the recombination operators (crossover and mutation) as disruptions of the population. Furthermore, he claims that the schema theorem cannot be used to explain the GA search strategy as it does not include the gene frequencies of the actual population, and is only limited to proportional selection.

[4] Price's theorem is a mathematical identity used for investigating an evolutionary process in terms of how a trait or gene changes in frequency over time. It was introduced by mathematician Richard H. Price in 1970.

4 Genetic Programming

Chapter 3 provided a synthesis of the most important concepts in the process of evolution and the main evolutionary computation methods. Now that we are finished with the prerequisites, we are ready to move forward to a detailed discussion of genetic programming – our main tool for the study of artificial evolutionary dynamics.

Genetic programming (GP) (Koza, 1992) represents, in a manner of speaking, the "perfect playground" for the experimental investigation of many biological ideas such as inheritance, diversity, evolvability and dynamics, in a simpler (but not necessarily less complex) setting where the user can control various aspects of the evolutionary process and study their effects.

In contrast to other methods such as genetic algorithms, evolutionary programming or evolution strategies, GP does not encode its solutions directly as a binary vector, real vector, etc. but represents its individuals as computer programs which are then executed in order to solve the problem. A computer program is nothing more than a set of instructions that tell the computer what to do – in our case, how to solve a given problem.

Figure Fig. 4.1 shows the general flowchart of GP evolution.

4.1 Solution Encoding

Since there are more than one way to represent or *encode* (as chromosomes encode genetic information) a computer program, GP can be seen as a sort of meta-evolutionary optimization model in which several predefined components (the recombination component, the selection component, the initialization component) interact with the encoding and with one another according to the rules of evolution, to produce solutions (or technically speaking, to *get computers* to solve problems). In other words, GP is essentially a domain-independent optimization method.

Since the development of the "standard" GP with syntax trees by (Koza, 1992), many other GP variants appeared, differing from one another by the concrete implementation of their encodings and associated operators. Additionally, GP encodings can be classified as direct and indirect, based on whether the chromosome encoding is the same as the solution representation. For example, an indirect encoding might use an integer vector as a chromosome and a grammar with appropriate rules for translating the chromosome encoding to a more meaningful solution representation (ie., a tree or a graph) that can be evaluated by an interpreter on the problem domain.

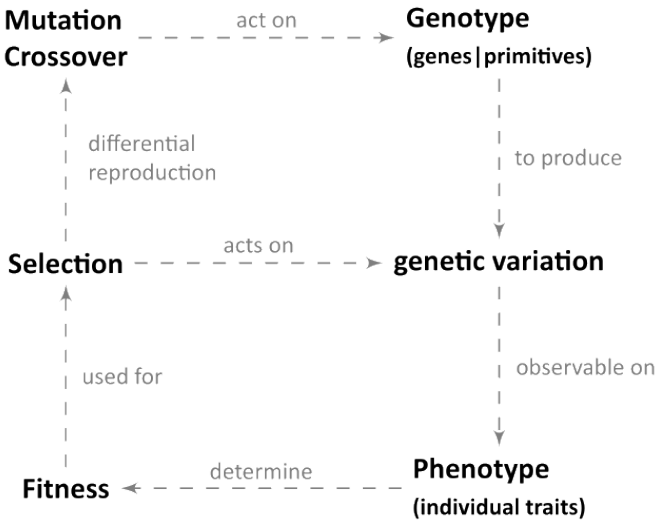


Figure 4.1: Flowchart of the GP evolutionary process

4.1.1 Syntax Tree Encoding

Since GP's invention by (Koza, 1992), syntax trees have remained the most popular chromosome encoding, and "standard GP" the most popular GP variant. In a GP syntax tree, internal nodes represent functions such as mathematical or logic operators and leaf nodes represent terminals such as variables or constants. For example, a chromosome encoding instructions for the Santa Fe Trail problem could encode in its tree structure a combination of conditional and movement instructions such as `(IF (FOOD-AHEAD MOVE-FORWARD TURN-LEFT))`. A chromosome encoding a mathematical formula could contain combinations of function symbols and terminals such as `(+ (× 2 a) b)`, evaluated by the interpreter as $2a + b$. In the given examples we used Koza's original S-Expression syntax typical of LISP programs, in which the first element represents an operator or function name (`IF`, `+`, `×`) and the following elements represent arguments (`a`, `b`, `2`, `MOVE-FORWARD`, `TURN-LEFT`). S-Expressions are equivalent to syntax trees as the one shown in Fig. 4.2. Direct tree representations in which the tree data structure resides in memory have the advantage of flexibility as they can be traversed, parsed or interpreted in many different ways (preorder, postorder, breadth, using a stack, using a queue, etc).

The chromosome representation is a very important detail in the implementation of any evolutionary search strategy. In GP, the elements of the alphabet which defines chromosome encoding are directly related to biological concepts such as robustness and redundancy; these concepts have a big influence on the evolutionary process.

Closure and Sufficiency. Properties of Good Representations

In GP, the set of all functions and terminals is called the *primitive set*. An essential requirement for programs encoded by the GP individuals is that any combination of instructions must result in a valid program, or any combination of subexpressions must result in a valid expression. In other words, the primitive set must satisfy the *axiom of closure*.

Another requirement for the primitive set is *sufficiency*, in the sense that the algorithm must be able to evolve solutions using the elements in the primitive set. For example, the set of boolean functions $\{\wedge, \vee, \neg\}$ is sufficient for any problem in the boolean domain.

According to (Goldberg, 1989) good representations should respect the following two principles:

1) *The Principle of Meaningful Building Blocks*
   The user should select a representation such that short, low-order schemata are relevant to the underlying problem and relative unrelated to schemata over other positions.

2) *The Principle of Minimal Alphabets*
   The user should select the smallest alphabet that permits a natural expression of the problem.

Although the two principles were originally stated for GA representations, it is a reasonable assumption that they apply to GP representations as well[1].

A more powerful generalization of the above principles is given by (Radcliffe, 1991) when discussing the design principles of good representations. Radcliffe uses the Latin term *formae* to emphasize the links between schemata and equivalence classes. Therefore in their formulation formae are induced by equivalence relations. The *precision* of an equivalence relation is defined as the number of formae it induces. We reproduce here the first two of Radcliffe's design principles and their explanations from (Radcliffe, 1991).

1) *Minimal redundancy*
   The representation should have minimal redundancy, and the redundancy should be capable of being expressed in terms of the equivalence relations used. This would allow the algorithm to "fold out" the redundancy (instead of treating redundant solutions as unrelated).

2) *Correlation with formae*
   Some of the equivalence relations, including some of low precision, must relate chromosomes with correlated performance. This ensures that information can be gathered about the performance of a forma by sampling its instances. Such information is used [by the algorithm] to guide the search.

[1] Assuming 'meaningful building blocks' and schemata can be defined for GP

Linear Encoding	Tree Encoding
Grammatical Evolution (GE) (Ryan et al., 1998)	CFG-GP (Whigham et al., 1995)
Cartesian GP (CGP) (Miller, 1999)	LOGENPRO (Wong and Leung, 1997)
Gene Expression Programming (GEP) (Ferreira and Gepsoft, 2008)	GGGP (Geyer-Schulz, 1996)

Table 4.1: Linear and tree-based grammar guided GP systems

We will refer to these principles later on, during our analysis of evolutionary dynamics and building blocks.

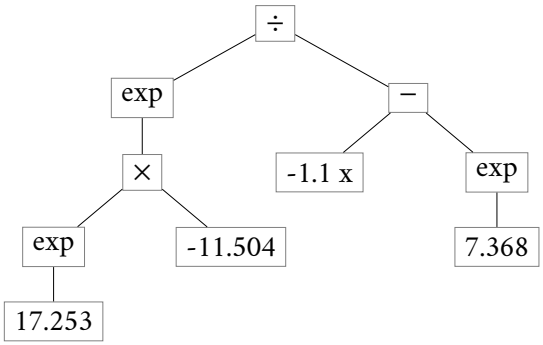


Figure 4.2: Example symbolic expression tree encoding a mathematical formula

4.1.2 Other Encodings

Grammar-based GP

In formal languages, grammars are defined as finite sets of production rules for strings. Conversely, grammars can also be used within this formalism to recognize and validate certain language constructs such as computer programs.

In the context of GP, grammars were introduced primarily to overcome the closure requirements of the primitive set. Grammar-based GP uses grammars to map chromosomes to computer programs, therefore separating the evolutionary search domain from the representation space. Grammars can also be used to immediately invalidate non-viable individuals (those individuals whose chromosomes do not map to valid computer programs).

Grammar-based genotype encoding can be either linear or tree-based. Linear encodings use bit strings or integer vectors while tree encodings use derivation trees. The main variants are presented in Table 4.1.

Tree representations in grammar-based GP are considered more powerful but linear representations have the advantage of a vast background of theory and practice from other evolutionary algorithms with a fixed-length representation such as GAs. For further information we turn the reader to (Mckay et al., 2010).

Graph-based GP

Graph theory defines trees as a simple, undirected, connected acyclic graphs. Additional edges in a tree could mean, from a GP perspective, the ability to evolve different kinds of structures such as neural networks or finite-state automata, or to achieve more compact representations by exploiting modularity within the structure of the computer programs. However, graph-based representations come at the cost of a more complicated logic required by the GP operators such as crossover and mutation.

Examples of graph-based GP include Parallel Distributed Genetic Programming (PDGP) (Poli, 1996), Cartesian GP (CGP) (Miller, 1999). CGP is both grammar-based and graph-based as the linear representation is expanded into a graph before evaluation, using rules which map the integer tuples from the linear representation to graph nodes placed on a 2-dimensional lattice. Special care must be taken at the level of linear GP operators that the created offspring can be expanded into valid graph representations.

Linear GP

In linear GP the chromosomes are represented by fixed or variable-length instruction sequences. The idea behind linear encodings is to move the representation closer to the machine level, using linear structures and registers for the evaluation of individual fitnesses. The simplicity of the encoding gives up some flexibility in return for greater speeds, usually orders of magnitude faster as individuals do not have to be evaluated by a tree interpreter or parsed by a grammar. Notable examples of GP systems using a linear encoding are Linear GP (Holmes and Barclay, 1996), Stack-based GP (Spector and Stoffel, 1996) and Machine-code GP (Nordin et al., 1999).

4.2 Genetic Programming Operators

Together with the representation, the choice of genetic operators is an essential, critical aspect of GP performance. In this section, we will present the main operators required by a GP system in order to evolve a population of syntax trees. For an easier notation, from now on we will refer to 'syntax trees' simply as trees and we will use the following terminology to refer to its properties:

- The *root node* represents the topmost node of a tree.

- ◇ The *node depth* represents the number of edges on the longest path from the node to a leaf. Thus, the tree depth is given by the depth of the root node.
- ◇ The *tree length* represents the total number of nodes (including the root node) contained in the tree.
- ◇ The minimum and maximum *node arity* represent the minimum and the maximum number of children that the node is allowed to have.

Generally speaking, a GP system needs to include the following components:

1) A tree initializer, used in the preliminary phase to initialize the population.
2) A tree evaluator, used to interpret the syntax trees and assign fitness values.
3) A selection operator. By default GP uses the same selection operators described in Section 3.2.4. Additionally, GP can employ specially-tailored selection operators which use supplementary criteria in choosing which individuals participate in reproduction. Special selection criteria can lead to dramatically different algorithmic dynamics.
4) Crossover and mutation operators. These operators modify the tree structure by making changes to leafs, nodes or subtrees. A detailed description is given below.

4.2.1 Tree Creation

Originally Koza (Koza, 1992) suggested three methods for tree creation: grow, full, and ramped hald-and-half. The methods generate tree individuals by recursively adding subtrees to a randomly-chosen root node until the specified tree depth or length limits are reached. The nodes to be added are picked at random from the available primitive set, with a specified probability of choosing between functions or terminals. The difference is that while the grow method can pick any node as a child and can generate trees of any shape and size, the full creation method is restricted to complete trees where leafs are only allowed to be placed on the last level. The ramped half-and-half method simply initializes some of the trees (half) with the grow method and the others with the full method.

Koza's tree creation methods provided a reasonably-good initialization for the genetic search. It was generally assumed that good diversity in the initialization phase could be obtained by using a large enough population size, to avoid sampling artifacts and to ensure a more or less uniform distribution of tree shapes and sizes. However, the methods were still sensitive to the depth and length control parameters and could not guarantee a target length for all created individuals. In this context, researchers tried to improve tree creation in order to provide a better start for the search procedure.

Attempts were made to ensure that the initial individuals are uniformly sampled from the set of available trees according to the maximum tree depth and tree length parameters. For example (Bohm and Geyer-Schulz, 1996) suggested a tree generation algorithm based

on previous work by (Alonso and Schott, 1995) on the random generation of combinatorial objects. Their creation method uses precomputed information to ensure uniform sampling from the set of all possible trees of a given size.

(Langdon, 1999) observed that GP search spaces are partitioned by the ridge in the number of program versus their size and depth and proposed a ramped uniform random initialization to straddle the ridge. Their tree initialization algorithm also derived from (Alonso and Schott, 1995) generates a uniform range of tree sizes. According to Langdon, the ramped uniform method represents an improvement over Koza's ramped half-and-half as it produces trees with shapes near the ridge in the search space.

Luke (Luke, 2000) identified several weaknesses in the tree creation methods introduced by Koza: no control over the probabilities that certain function symbols are included (the methods sample the available functions uniformly), no control over the generated tree shape (in the case of the grow method) and no possibility to create trees with a fixed or average tree length or depth. He proposed two new probabilistic tree creation methods named `PTC1` and `PTC2` which allow the user to specify an expected tree size and a probability distribution for the desired function nodes.

In a follow-up paper Luke and Panait (Luke and Panait, 2001) found that `PTC1` and `PTC2` do not offer any significant advantage in terms of solution quality as uniformity does not have a big influence on improving fitness. However, the possibility to hand-tune function probabilities and expected tree lengths during initialization is likely to have a positive impact on algorithm dynamics and on the sizes of the resulting solutions and their interpretability.

4.2.2 Selection

Fitness-based selection used by default in GP was already discussed in Section 3.2.4. We discuss here some of the more subtle effects of the selection scheme. As selection only acts on phenotypes, it does not protect the population against the deleterious effects of crossover and mutation. Depending on the relative fitness differences between individuals, genotypes with high adaptive potential affected by some deleterious change may be discarded altogether by selection in the early stages of evolution. This leads to the loss of potentially useful genetic variation, decreasing the population's adaptive potential.

Depending on how they sample the distribution of fitness values, different selection schemes will exert different levels of selection pressure on the population and will have different effects on algorithm dynamics and convergence.

Diversity and the Exploration-Exploitation Trade-off

In the context of genetic search, exploration refers to the algorithm's ability to probe different areas of the search space, and exploitation refers to the algorithm's ability to focus on a single area and locally improve solution quality.

It is assumed that more diverse individuals in the population would correspond to more spread out points in the solution space. Selection is a force that acts against genetic variation by only preserving useful[2] variation and throwing away the rest. Therefore, selection gradually changes the distribution of points in the solution space towards local optima. It is shown in (Xie, 2009) that loss of population diversity in GP is entirely due to the not-sampled individuals.

The antagonistic relationship between the exploration and exploitation aspects of genetic search has been observed quite early. (Goldberg and Deb, 1991) analyze the available selection schemes and their influence on algorithm convergence. Each selection scheme is analyzed in terms of its *growth ratio* – the growth of an individual with a specified rank in a population. They suggest a number of guidelines for maintaining a balance between exploration and exploitation:

- Use slow growth ratios to prevent premature convergence
- Use higher growth ratio followed by mutation
- Permit localized differential mutation rates
- Preserve useful diversity through niching, dominance and diploidy

Choosing the right selection scheme has proven to be a delicate matter, especially as a selection scheme's effectiveness depends on the population size and the structure of the fitness landscape.

Population diversity is generally seen by the GP community as a major factor in algorithm performance. In many approaches, researchers have tried to obtain a more fine-tuned control over the exploration-exploitation aspect of the search through the incorporation of secondary mechanisms in the selection scheme meant to preserve and improve population diversity. A summary of the various diversity measures and diversification strategies is given below:

- (Altenberg, 1994b) introduced soft brood selection to "shield reproduction from the cost of producing deleterious offspring" and to "shift the evolution of representations away from a conservative strategy towards an exploratory strategy". Soft brood selection works by producing a "brood" of offspring from two parents and then holding a tournament between the brood members. The winner is returned as the offspring contributed by the two parents.
- (Rosca, 1995) used an analogy with statistical physics in which population diversity and average fitness are seen as state variables corresponding to the entropy and energy of a physical system. Diversity is taken at the semantic level and defined using the Boltzmann entropy function in which the probabilities p_i of a particle finding

[2] Usefulness in the context of GP is entirely dependant on fitness, therefore the rate of diversity decrease in a population varies with selection pressure.

itself in state i are interpreted as probabilities for an individual in the population to belong to a certain fitness group. Rosca suggests that the correlation between the two state variables can be used as a steering mechanism for the algorithm.

- (O'Reilly, 1997) measured the average tree edit distances between the best individuals in the population and the remaining ones and between crossover children and their parents. They use these measures to characterize in a quantitative manner the exploration and exploitation properties of the search.
- (Matsui, 1999) introduced a correlative tournament selection method which tries to mate individuals based on their correlation. Basically, one parent is chosen using a regular selection scheme and the other as the one with the best correlation from a set of randomly chosen candidates. The idea of this selection method is that similar parents will produce better offspring.
- (Wiese and Goodwin, 1998) suggested a "keep best" selection scheme where after crossover is applied, the worst offspring is replaced by the best parent. The idea is to ensure that good previous genetic material is being preserved.
- (Ekárt and Németh, 2000) used a variant of the edit distance where the cost of edit operations is weighted according to the node level. While exponential in complexity for unordered trees, the distance metric allowed the authors to employ distance-based niching techniques in order to improve algorithm performance.
- (Burke et al., 2003) proposed a selection scheme guided by genealogy information and named lineage selection. They define a genetic lineage as the genealogy formed by an individual and all its descendants. The purpose of lineage selection is to "redirect selection pressure from the *fit* to the *fit and diverse*". Lineage selection works by running tournaments between individuals belonging to distinct genetic lineages.
- (Yan and Clack, 2006) grouped individuals based on their fitness in intervals called 'fitness segments', then used a behavioral similarity measure based on an individual's 'behavioral history' (a vector of past evaluations) to delete individuals that are too similar (according to the Spearman correlation coefficient) with the average history of their containing fitness segment. In addition, they incorporated the behavioral history into the fitness calculation formula.

Adaptive Selection Pressure through Offspring Selection

Realizing the importance of preserving good genetic material in the population, (Affenzeller and Wagner, 2003; Affenzeller et al., 2009) introduced a selection scheme similar to soft brood selection (Altenberg, 1994b), but extended in such a way as to:

1) Preserve population diversity and use it in more efficiently

2) Maintain the balance between exploration and exploitation by providing an adaptive control of selection pressure

3) Support self-organization and emergence of building blocks by minimizing the destructive effects of the recombination operators

Offspring selection represents an extension of regular selection methods where newly produced offspring are conditionally accepted into the new population. The additional selection criteria are verified in a post-reproduction selection step where the offspring are evaluated and compared to their parents. Normally, this means that a specified inequality equation between the offspring and parents fitness value needs to be satisfied. For example, different offspring selection criteria might accept new offspring only if they fitness is better than the best parent, the worst parent, or any intermediate value.

The work-flow of the offspring selection procedure is described in pseudocode in Algorithm 2. The control parameters for the procedure are described below:

- The comparison factor c controls the fitness threshold for the offspring selection criteria and is dynamically adjusted during the run between $[0, 1]$. A value of 0 means that the offspring has to be better than the worst parent while a value of 1 means that the offspring has to be better than the best parent. The online scaling of the comparison factor between 0 and 1 effectively implements a search strategy where the algorithm is more focused on exploration in the beginning, gradually becoming more and more directed towards the end. Similar to simulated annealing, the scaling of parameter c can take place according to different rules: linear, exponential, etc. Usually, c is kept fixed during the evolutionary run.

- The success ratio s specifies what ratio of the population should be filled with offspring that outperform their parents (according to the offspring selection criteria). If $s < 1$, then the remaining ratio of the population is filled with offspring randomly chosen from the rejected pool of offspring that did not pass the offspring selection criteria.

- The maximum selection pressure value p_{max} defines the maximum number of offspring that may be considered for the next generation, as a multiple of the actual population size.

The offspring selection mechanism can be used to detect premature convergence when the algorithm is no longer able to produce a sufficient number of successful offspring (according to the population size and success ratio s) after $p_{max} \cdot PopulationSize$ candidates have been generated.

4.2.3 Crossover

The crossover operation plays an important role in GP as the main tool for exploring large search spaces. GP crossover is usually performed by swapping randomly chosen

Input:

- success ratio r
- comparison factor c
- upper limit for selection pressure p_{max} (expressed as a multiple of the actual population size)

Output: a new generation of child individuals

$POP_i \leftarrow$ population at current generation i;
$POP_{i+1} \leftarrow$ future population to be created at generation $i + 1$;
$POOL \leftarrow$ pool of offspring that do not satisfy offspring selection criteria;
while $|POP_{i+1}| < r \cdot |POP_i|$ **do**
    select parents using default method (tournament, proportional, etc.);
    generate a child by applying crossover and/or mutation on the parent(s);
    $f_c \leftarrow$ fitness value of the child individual;
    $f_{p_1} \leftarrow$ fitness value of first parent;
    $f_{p_2} \leftarrow$ fitness value of second parent;
    if $f_c > \left(\min(f_{p_1}, f_{p_2}) + c \cdot |f_{p_1} - f_{p_2}|\right)$ **then**
        add child to POP_{i+1};
    else
        add child to $POOL$;
    $p_{active} \leftarrow \frac{|POP_{i+1}|+|POOL|}{|POP_i|}$;
    if $p_{max} < p_{active}$ **then**
        maximum selection pressure reached – stop the algorithm;
`// At this moment` $r \cdot |POP_i|$ `successful offspring will have been created`
while $|POP_{i+1}| < |POP_i|$ **do**
    insert a random child from $POOL$ into POP_{i+1};

Algorithm 2: Pseudo-code description of the offspring selection scheme

subtrees between two parent tree individuals. As the main variation-producing operator in GP populations, crossover has been thoroughly investigated and many different variants and improvements can be found in the literature.

In its basic form (Fig. 4.3), crossover takes two parent individuals and generates a child by replacing a subtree from the first parent with a subtree from the second parent. We call the receiving parent that contributes the rest of its genotype the *root* parent and the donating parent the *non-root* parent. Throughout this work, we use the *fragment* (as in genotypic fragment) to denote a subtree that was swapped from the non-root into the root parent.

Considering that a significant portion of all the nodes in the population are terminals, it makes sense to bias the crossover towards function nodes; otherwise, crossover might

simply replace one terminal with another, effectively behaving like point mutation. To avoid this problem, in practice the crossover operator is biased towards function nodes with a probability of 90%, as originally suggested in (Koza, 1992).

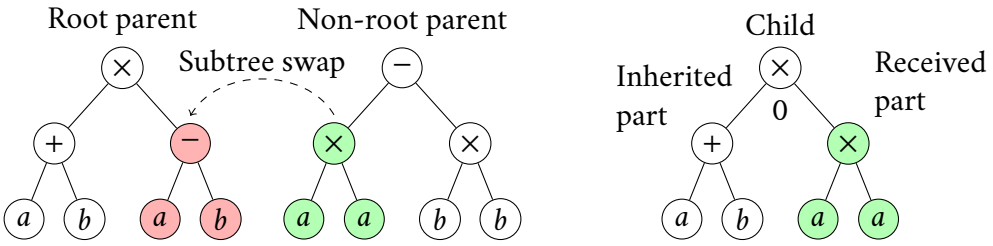


Figure 4.3: Example crossover

The interplay between selection and crossover influences the way existing genotypic variation within the population is used and maintained.

Crossover and Bloat

It was observed quite early during GP trials that the average program size increases with the number of generations and this increase is not always accompanied by an increase in fitness. This phenomenon, known as *bloat* has been found to affect the search negatively by promoting large, overfit solutions and slowing down the algorithm as large programs are more computationally expensive to evolve.

In practical terms, bloated programs contain regions of code which have no influence on program output. Such regions with no useful function are called *introns*. A further distinction can be made between code fragments that represent inviable code (for example, a conditional branch that will never be hit) and code fragments that owe their neutral (or even detrimental) effect on fitness to inappropriate function arguments like weighted variables or constant values. In the second case, a bad or neutral code fragment can become active after being improved through crossover or mutation.

Explanations for Bloat Early theorems regarded introns (inviable code in particular) as the main reason for bloat proposing different explanations for their occurrence:

- *Protective effects of introns* (Mcphee and Miller, 1995; Soule and Foster, 1998) suggest that introns proliferate due to their protective role as a buffer against the disruptive effects of genetic operators. Therefore, bloated individuals have a higher chance of maintaining their fitness after selection. Additionally, as suggested by McPhee, replication accuracy can act as a force towards more accurate solutions.
- *Removal bias* (Langdon et al., 1999; Soule and Foster, 1998) Since these regions are more likely to be found towards tree extremities, the probability of crossover

affecting fitness is inversely proportional with branch size. Therefore, fitness neutral operations in regions of inviable code have a tendency to replace smaller subtrees with average-sized ones.

- ◇ *Fitness-based selection* (Langdon and Poli, 1998; Langdon et al., 1999) showed that bloat is inherent to variable-length representations under fitness-proportional selection when the genetic operators can alter the sizes of programs. They argue that variable length allows many more larger representations to exist for the same fitness level, leading to an increase in average program size as a side-effect of selection.
- ◇ *Crossover bias* (Poli et al., 2007a,b) isolate the effects of crossover by using crossover on a flat fitness landscape, and find that while crossover does not affect the average program size, it changes the tree size distribution in the population to a Lagrangian distribution of the second kind. This means that small programs will be present in the population with a much higher frequency. They then argue that since small programs are less likely to have good fitness, larger ones will be more likely to survive under selection. Therefore, the crossover bias towards a particular distribution of tree sizes ultimately translates into an increase in average program size under selection. Intuitively, this may be explained by the fact that it is easier to achieve progress with larger programs than with simpler (smaller) programs.

Bloat Control Methods The most obvious method for controlling bloat is to impose static depth and length limits on the tree individuals. Another method for controlling bloat in GP programs is the *parsimony pressure* method suggested by (Koza, 1992). The idea is simple: to penalize program fitness based on their size according to the formula:

$$f_p(x) = f(x) - c \cdot \ell(x)$$

where $\ell(x)$ returns the program size, and c is a penalty factor. In practice, it is not easy to find an appropriate value for the penalty factor c. (Poli and Mcphee, 2008) used a more general form for the formula:

$$f_p(x) = f(x) - g\left(\ell(x), t\right)$$

where g is a function of program size and the number of generations t. They applied Price's theorem to the GP size evolution equation (Poli and McPhee, 2003b) and showed that a covariant relationship exists between g and f. Taking the absence of growth in the expectation is equivalent with $Cov(\ell, g) = Cov(\ell, f)$. Since the method deals with expectations, a sufficiently large population size is needed.

Other Crossover Variants

Ideally, the goal of crossover is to produce useful genetic variation leading to adaptation under the effects of selection. However, empirical evidence has led to criticism of the

crossover operator for being highly destructive (Nordin et al., 1996). Additionally (Angeline, 1997) showed crossover to be no better on average than mutation, although Angeline's criticism has been disproved by Koza in a later edition of their book (Koza, 1999).

In order to reduce bloat, preserve population diversity and increase evolvability, crossover operators can include additional heuristics for selecting more appropriate subtrees to be swapped (for example mechanisms for ensuring the exchange of homologous structures).

Many authors investigated crossover in terms of how it affects population diversity and algorithm dynamics. As a consequence, multiple variants aimed at improving diversity and GP search dynamics were proposed. Some of them are listed below.

- *One-point crossover* by (Poli and Langdon, 1997) is basically a restricted version of the standard crossover that only swaps subtrees with the same size and shape. The main advantage of this form of crossover is that it allows (through its simplicity) the calculation of crossover disruption rates on a model of GP schemata[3]. In their experiments the authors confirmed that one-point crossover works very well in combination with one-point mutation and outperformed the standard crossover when a variable mutation rate is used.

- *Size-fair crossover* (Langdon, 1999). The size-fair crossover choses a random crossover point in the root parent and replaces it with a subtree of approximately the same size from the non-root parent. This means that a terminal will always be replaced by another terminal, while a function node will always be replaced by another function node. The main idea of the size-fair crossover is to reduce bloat by controlling program size during the recombination phase.

- *Homologous crossover* (Langdon, 1999) The homologous crossover works in a similar manner to the size-fair crossover, with the exception that the second subtree from the non-root parent is deterministically chosen as the most similar to the subtree to be replaced, with respect to size, position and shape. In their experiments, 16% of all children produced by the homologous crossover were identical with the root parent, compared to 7% for the size-fair and 5% for the standard crossover.

- *Uniform crossover* (Poli and Langdon, 1998b) This crossover acts on the common region between the two parent individuals. This involves the additional effort of aligning the crossover points and identifying the compatible subtree pairs. In their analysis, Poli and Langdon showed that standard crossover behaves as a local search operator, producing offspring which inherit most of their code from one parent most of the time. Similarly, the size-fair crossover becomes more local as the search converges. The uniform crossover, while still losing its global search properties, is able to perform a less biased search since any node can be transferred from parent to child with the same probability.

[3] Schema theorems for GP will be discussed in detail in Section 4.3

- *Semantically-driven crossover* (Beadle and Johnson, 2008) The authors try to enhance behavioral diversity by not allowing crossover child programs that are functionally equivalent to their parents. Functional equivalence is calculated by bringing programs to a canonical form where all redundant and unreachable arguments are removed. Programs that reduce to the same canonical form are deemed functionally equivalent. The semantically-driven crossover was shown to outperform standard crossover and reduce the average depth of programs by approximately one-third.

- *Probabilistic functional crossover* (Bongard, 2010) In an initial phase, this operator calculates behavioral distances between a randomly selected crossover point in the root parent and the nodes of the non-root parent parent. Then, the behavioral distances are normalized and turned into selection probabilities, and the crossover operator makes a weighted choice of the second subtree. The point behind this approach is to increase the probability of beneficial crossover events.

- *Semantic similarity crossover* (Uy et al., 2010) follows a similar idea and defines subtree sampling semantics as the collection of its evaluated values over a sequence of points in the dataset. A sampling semantics distance between two nodes is calculated as the Manhattan norm of the point wise differences between the sampling semantics. Two nodes are similar if their sampling semantics distance falls within a predefined interval. The crossover operator picks a crossover point in the root parent then swaps it with the first node in the non-root parent that satisfies the semantic similarity condition.

- *Context-aware crossover* (Majeed and Ryan, 2006) The context-aware crossover randomly picks a subtree from the non-root parent and then produces a set of candidate offspring corresponding to each possible position where the subtree can be swapped into the root parent. The best offspring in terms of performance is returned as the result of crossover. Due to its exhaustive nature, this crossover greatly increases the number of evaluations performed by the algorithm.

The crossover operators described above all have one thing in common: their authors showed that each of them outperforms standard crossover in terms of search performance and bloat reduction. In all cases, the perceived benefits come at the cost of significantly increased computational complexity of the crossover operator due to the need to evaluate multiple child candidates, to calculate distance metrics or to perform tree alignment operations. Therefore, it could be argued that the claim of increased performance is not an entirely fair claim. Furthermore, as the dynamics of the artificial evolutionary process in GP are not fully known we cannot say for sure (in the spirit of the no free lunch theorem) which methods are better to direct the search. In this context, a too-specialized crossover may limit our ability to focus on the evolutionary system as a whole and investigate all of its aspects.

4.2.4 Mutation

Mutation in genetic programming is employed to inject new genotypic variation into the population. It works best as a complement to crossover and acts mostly as a local search operator. Similar to crossover, different variations exist:

- *Point mutation* randomly changes a single node in the individual. It ensures that the mutated tree is valid by only exchanging compatible function nodes with the same arity.
- *Subtree mutation* changes an entire subtree in the individual. The replacement subtree can be randomly generated or can be created to be approximately the same size (a size-fair mutation). Another version of this operator called shrink mutation replaces a function subtree with a randomly generated terminal.
- *Constants mutation* only affects constants of a randomly selected subtree. Constant mutation operators can also use local optimization methods

4.3 Genetic Programming Schema Theorems

With the development of GP many efforts by Koza and others were made towards a working "GP schema theorem" that could explain its dynamics in a manner similar to Holland's schema theorem described in Section 3.2.4.

However, the difficulties posed by the highly-polymorphic, variable-length tree encoding caused the theoretical advancements in the field of GP schema theorems to take place at a very slow rate. Empirical investigations in this direction were especially difficult to set up due to the vast number of subtree combinations that had to be analyzed.

At the most basic level, tree-based schema representations have to overcome all the algorithmic challenges brought by dealing with labeled, unordered trees. Additionally, encoding-specific particularities concerning the properties of the primitive set (for example, whether some functions are transitive or not) have to be considered.

Once a suitable schema definition, along with tools for calculating its basic properties in the population (such as frequency, average fitness, defining length, etc.) has been obtained, the next step is to investigate the effects of genetic operators (crossover, mutation, selection) on the distribution of schemata in the population.

In the last three decades, several schema definitions and methodologies have been investigated. Previous work in this area is summarized below.

4.3.1 Koza's Schema Definition

(Koza, 1992) defined schemas rather loosely as the set of subtrees which included in their structure a predefined set of complete trees. For example, in Koza's definition the schema

$H = \{(+\ 1\ x), (\times\ x\ y)\}$ represents all the programs in which both expressions $(+\ 1\ x)$ and $(\times\ x\ y)$ occur at least once. One problem with this schema definition is that it does not include positional information regarding the placement of the expressions within the tree, making it impossible to calculate the schema order or defining length.

4.3.2 O'Reilly and Oppacher's Schema Definition

(O'Reilly and Oppacher, 1992) extended Koza's work and defined schemas as trees that contained wildcard nodes identified by "don't care" symbols denoted by '#'. Since wildcard nodes can be matched by any other subtree (including leafs), they made an additional distinction between subtrees (valid S-expressions) and *fragments*, defined as more general expressions isomorphic to subtrees but not required to be complete (within a fragment, a function node which would normally require arguments can have no children). Under this definition, schema order was calculated as the number of non-# nodes. The schema defining length was calculated as the sum between the *fixed defining length* (the number of edges within each subtree or fragment) and the *variable defining length* (the number of edges separating the groups of subtrees or fragments):

$$\mathcal{L}(H) = \mathcal{L}_{fixed}(H) + \mathcal{L}_{var}(h, H)$$

O'Reilly's schema theorem only considers the effects of crossover. When the crossover point selection probabilities vary between internal and leaf nodes, the probability of disruption for program h sampling schema H is given by

$$p_d(h) = \frac{L_b v(H) + (1 - L_b)\left(\mathcal{L}(H) - v(H)\right)}{Size(h)}$$

where $v(H)$ represents the number of leaf crossover points, and L_b represents the selection probability for leaf crossover points. The *compactness* of a program h is defined as the opposite of disruption such that: $C(h) = 1 - p_d(h)$.

The average probability that a schema H will be disrupted is given by

$$\overline{p}_d(H) = \frac{\sum_h^{i(H,t)} p_d(h)}{i(H,t)}$$

Finally, with the same notation from the GA schema theorem, the equation for the GP schema theorem has the expression:

$$E\left[m(H, t+1)\right] \geq m(H,t)\frac{f(H)}{\overline{f}}\left(1 - p_c \cdot \max\{P_d(H,h,t)\}\right)$$

where p_c is the crossover probability, P_{d_c} is the probability of disruption by crossover of schema H, $\overline{f}$ is the average fitness of the population and $f(H)$ is the observed fitness of the schema.

Disruption is defined as a probability $Pr(E)$ of the event E that $\widetilde{D}_t$ is less than a constant β. A schema is disruption prone if $Pr(E) < \alpha$, with α a constant. Compactness is defined as $1 - Pr(E)$ so for $Pr(E) > \alpha$ the schema is compact.

O'Reilly and Oppacher give their result the following interpretation: "Compact schemas with above average observed performance (GP Building Blocks) will be sampled at exponentially increasing rates".

4.3.3 Wigham's Schema Definition

(Whigham, 1995) investigated schemas in the context of grammar-based GP. In their CFG-GP (Whigham et al., 1995) individuals are derivation trees in which internal nodes represent rewrite rules and leaf nodes represent functions and terminals used in the generated programs. Genetic operators in CFG-GP manipulate the derivation trees and always produce valid offspring derivation trees.

Whigham defined a schema as a partial derivation tree, and gave equations for the probabilities of disruption under crossover and mutation, $P_{d_c}(H, h, t)$ and $P_{d_m}(H, h, t)$. Since the disruption probabilities depend on the size of the matching trees, Whigham took the average probabilities $\overline{P}_{d_c}$ and $\overline{P}_{d_m}$ (for crossover and mutation).

Using these equations they arrived to the following expression:

$$E\left[m(H, t+1)\right] \geq m(H, t)\frac{f(H, t)}{\overline{f}_t}\left(P_{d_m}(H, t) \cdot P_{d_c}(H, t)\right)$$

where

$$P_{d_c}(H, t) = \left(1 - p_c\overline{P}_{d_c}(H, t)\right)$$
$$P_{d_m}(H, t) = \left(1 - p_m\overline{P}_{d_m}(H, t)\right)$$

with crossover probability p_c and mutation probability p_m.

4.3.4 Rosca's Schema Definition

(Rosca, 1997) was the first to introduce a schema representation as rooted trees, in which wildcard symbols have a well-defined position in the tree relative to the root node. Rosca's reasoning for fixing the top-level structure of the tree-schema had been, in their own words, that "the evolved structures appear to drift towards large and slow forms on average". In Rosca's schema theorem the expression of the disruption probability is given by:

$$P_d(H, t) = \sum_{h \in H \cap Pop(t)} \frac{O(H)f(h)}{N(h) \cdot \sum_{h \in H \cap Pop(t)} f(h)}$$

where $N(h)$ is the size of a program h matching the schema H. The schema theorem equation becomes:

$$E\left[m(H,t+1)\right] \geq m(H,t)\frac{f(H,t)}{\bar{f}_t}\left(1-(p_m+p_c)\cdot P_d(H,t)\right)$$

4.3.5 Poli and Langdon's Schema Definition

(Poli and Langdon, 1998c) defined schemas as rooted trees, with nodes from the primitive set $\mathcal{F} \cup \mathcal{T} \cup \{=\}$, where the wildcard symbol '=' represents a single node (function or terminal). The main difference in using this schema representation is that, as opposed to the previous schema definitions which allowed the set of matching trees to vary in shape and size, in this case the shape and size of the matching trees will be fixed (the same size as the schema itself).

The goal of making schemas independent of the matching trees' shape and size was to make the effects of genetic operators easier to calculate. The schema order, length and defining are defined in the following way:

- The order $O(H)$ of a schema H is given by the number of non-= symbols.
- The schema length $N(H)$ of a schema H is given by the total number of nodes in the schema.
- The schema defining length $\mathcal{L}(H)$ is given by the number of links in the minimum tree fragment including all the non-= symbols within a schema H.

In their paper, Poli and Langdon considered all the cases in which a schema H can be disrupted by crossover and mutation. The calculations apply to GP runs using fitness-proportionate selection, one-point crossover, and point mutation:

- Point-crossover works by first identifying the common region between the two parents and then swapping nodes from the same position in the common region between the two parents, at a randomly chosen cut-point
- Point mutation works by changing a single node in the tree, and only with a node of the same type (functions with functions and terminals with terminals)

The main results of Poli and Langdon's schema theorem are given below, skipping the full-length demonstration which can be found in (Poli and Langdon, 1998c). The theorem is derived from calculating the probabilities of the events that affect schema sample count from one generation to the next:

1) Disruption by crossover or mutation happens when the events $D_c(H)$ (disruption by crossover) and $D_m(H)$ (disruption by mutation) take place
2) The event $D_c(H)$ can take place in two mutually-exclusive situations. We consider tree h instantiating schema H, and the zero-th order schema $G(H)$ (the hyperspace

associated with H):

a) h is crossed with a program $\hat{h} \notin G(H)$ (that has a different structure from H). We denote this disruption event $D_{c_1}(H) = \{D_c(H), \hat{h} \notin G(H)\}$

b) h is crossed with a program $\hat{h} \in G(H)$ (that has the same structure as H). We denote this disruption event $D_{c_2}(H) = \{D_c(H) | \hat{h} \in G(H)\}$.

The two mutually-exclusive events correspond to two distinct probabilities:

$$\Pr\{D_{c_1}(H)\} = \Pr\{D_c(H) | \hat{h} \notin G(H)\} \Pr\{\hat{h} \notin G(H)\}$$
$$\Pr\{D_{c_2}(H)\} = \Pr\{D_c(H) | \hat{h} \in G(H)\} \Pr\{\hat{h} \in G(H)\}$$

Since the two probabilities will need to be added together to obtain the expression for the crossover probability of disruption, it is useful to rewrite some terms:

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

Additionally, for the sake of brevity the notation $p_{\text{diff}}(t)$ was used to denote the term $\Pr\{D_c(H) | \hat{h} \notin G(H)\}$.

3) In the case of mutation, the disruption probability is considerably easier to calculate. For mutation probability p_m, the probability that all defining nodes of schema H survive mutation unaltered depends on the schema order $O(H)$:

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

When $p_m \ll 1$, the equation can be simplified using the first terms of its Taylor expansion:

$$\Pr\{D_m(H)\} \approx p_m O(H)$$

4) The expression for Poli and Langdon's schema theorem after including all the probabilities described above becomes:

$$E\left[m(H, t+1)\right] = M \Pr\{h \in H\} \left(1 - \Pr\{D_m(H)\}\right)\left(1 - p_c \Pr\{D_c(H)\}\right)$$

Performing all the substitutions in the terms and considering the upper bound for the possibility of disruption by crossover, Poli and Langdon's schema theorem can be rewritten as a lower bound for the expected number of individuals sampling a

schema H:

$$E\left[m(H,t+1)\right] \geq m(H,t)\frac{f(H,t)}{\overline{f}(t)} \cdot (1-p_m)^{O(H)} \cdot \left\{1 - p_c\left[p_{\text{diff}}(t)\left(1 - \frac{m\left(G(H),t\right) f\left(G(H),t\right)}{M\overline{f}(t)}\right) + \frac{\mathcal{L}(H)}{N(H)-1}\frac{m\left(G(H),t\right) f\left(G(H),t\right) - m\left(H,t\right) f\left(H,t\right)}{M}\right]\right\}$$

The notation in the above expression is consistent to the notation used for the expression of the GA schema theorem: M represents the number of individuals. $m(H,t)$ represents the number of strings matching schema H at generation t and $f(H,t)$ represents the average fitness of schema H at generation t.

In (Poli and Langdon, 1998c), Poli and Langdon introduced new versions of the crossover and mutation operators, namely the one-point crossover and point-mutation in order to derive a schema theorem which gives a lower bound for the expected number of schema instances at each generation. Their theoretical investigations produced the following conclusions:

1) Because of crossover, the probability of schema disruption is very big at the beginning of a run. In the beginning, crossover heavily counteracts the effects of selection.

2) Schemata with above average fitness and short defining length (with respect to their total size) tend to survive more frequently, if their shape is of above average fitness ($f(G(H),t) > \overline{f}(t)$) and is shared by an above average number of programs ($m(G(H),t) > 1$))

3) One-point crossover outperforms standard crossover

Exact Schema Theorem

In subsequent work, (Poli, 2000; Poli and Langdon, 1998a) identified the main weaknesses in their schema theorem, namely the presence of the expectation operator and a rather loose lower bound for the expected number of schemas in the next generation. They work around these limitations by extending the results of Stephens and Waelbroeck, who developed an exact schema theorem for GAs with fixed-length string representation (Stephens and Waelbroeck, 1999, 1998).

The constructive effects of crossover

Reformulating the results of Stephens and Waelbroeck, Poli arrives to the following form

of the schema evolution equation for a GA with a fixed-length string representation:

$$\alpha(H,t) = (1 - p_{xo})p(H,t) + \frac{p_{xo}}{N-1}\sum_{i=1}^{N-1} p\left(L(H,i),t\right) p\left(R(H,i),t\right)$$

where:

1) N is the length of the representation and p_{xo} is the probability of crossover.
2) $p(H,t)$ is the selection probability of schema H at generation t
3) $L(H,i)$ represents the schema obtained from H by replacing all elements from position $i+1$ to position N with $*$ (don't care)
4) $R(H,i)$ represents the schema obtained from H by replacing all elements from position 1 to position i with $*$ (don't care)

The equation tells us that the total transmission probability $\alpha(H,t)$ of schema H depends on the probabilities that lower-order schemas are present in the population (the terms $p\left(L(H,i),t\right)$ and $p\left(R(H,i),t\right)$ in the equation). In more general terms as a function of schema selection and creation probabilities, $\alpha(H,t)$ can be rewritten as:

$$\alpha(H,t) = p_s(H,t)p(H,t) + p_c(H,t)(1 - p(H,t))$$

where

- $p_s(H,t)$ represents the probability that schema H survives crossover (offspring of individuals sampling H will still sample H)
- $p(H,t)$ is the schema selection probability
- $p_c(H,t)$ is the schema creation probability

Taking the schema sampling event (after crossover and selection) as a Bernoulli trial, the number of times a schema H is sampled at step $t+1$ can be defined as a stochastic variable:

$$\Pr\{m(H,t+1) = k\} = \binom{M}{k}\alpha(H,t)^k(1 - \alpha(H,t))^{M-k}$$

where M is the number of individuals in the population. When $k = 0$, the probability of schema extinction becomes:

$$\Pr\{m(H,t+1) = 0\} = (1 - \alpha(H,t))^M$$

The expectation and variance are given by:

$$E[m(H,t+1)] = M\alpha(H,t)$$
$$Var[m(H,t+1)] = M\alpha(H,t)(1 - \alpha(H,t))$$

Using the Chebychev inequality with the two equations above leads to a general schema theorem without the expectation operator:

$$\Pr\{m(H,t+1) > M\alpha(H,t) - k\sqrt{M\alpha(H,t)(1-\alpha(H,t))}\} \geq 1 - \frac{1}{k^2}$$

Hyperschema

In order to be able to define a schema transmission probability $\alpha(H,t)$ for GP in the same way Stephens and Waelbroeck did for GA, Poli introduced a more general definition of a schema called a *hyperschema* which included a second type of a wildcard symbol. Poli's hyperschema definition, given below, is the one which will be used in the remainder of this work. By using the word schema, we will automatically refer to Poli's hyperschema.

Definition (**Hyperschema**). A hyperschema is defined as a rooted tree were nodes could be any of the symbols $\mathcal{F} \cup \mathcal{T} \cup \{=, \#\}$. The '=' symbol can match a single node (function or terminal) while the '#' symbol can match any valid subtree.

Using this hyperschema definition, the total transmission probability can be expressed in a very similar manner as before:

$$\alpha(H,t) = (1-p_{xo})p(H,t) + \frac{p_{xo}}{N(H)-1}\sum_{i=1}^{N-1} p\left(L(H,i),t\right)p\left(R(H,i),t\right)$$

where:

1) $N(H)$ is the number of nodes in the schema H
2) $L(H,i)$ is the hyperschema obtained by replacing with = all the nodes on the path between crossover point i and the root node and with # all the nodes connected to the nodes replaced with =.
3) $U(H,i)$ is the hyperschema obtained by replacing with a # node the subtree below crossover point i

Finally, after all the prerequisites the exact schema theorem can be formulated:

Theorem (**Exact GP Schema Theorem**). The total transmission probability for a fixed-size-and-shape GP schema H under one-point crossover and no mutation is

$$\alpha(H,t) = (1-p_{xo})p(H,t) + p_{xo}\cdot\sum_{h_1,h_2\in Pop(t)}\frac{p(h_1,t)p(h_2,t)}{NC(h_1,h_2)}\cdot\sum_{i\in(h_1,h_2)}\delta\left(h_1\in L(H,i)\right)\delta\left(h_2\in U(H,i)\right)$$

where

1) $NC(h_1,h_2)$ is the number of nodes in the common region between h_1 and h_2
2) $C(h_1,h_2)$ is the set of indices of common crossover points.

3) $\delta(x)$ is the Kronecker-Delta function (1 if x is true and 0 otherwise)

When the individuals have the same shape and size as schema H (belonging to the zeroth-order schema $G(H)$), a lower bound for the total transmission probability can be obtained:

$$\alpha(H,t) \geq (1-p_{xo})p(H,t) + \frac{p_{xo}}{N(H)-1}\sum_{i=1}^{N(H)-1}\sum_{h_1\in G(H)} p(h_1,t)\delta(h_1 \in L(H,i))\sum_{h_2\in G(H)} p(h_2,t)\delta(h_2 \in U(H,i))$$

where a few simplifications were made since $h_1, h_2 \in G(H)$, therefore $NC(h_1,h_2) = N(H)$, therefore $C(h_1,h_2) = \{1,2,\ldots,N(H)-1\}$.

From the equation above, the schema theorem with schema creation correction can be obtained:

Theorem (GP Schema Theorem with Schema Creation Correction). The lower bound for the total transmission probability for a fixed-size-and-shape GP schema H under one-point crossover and no mutation is

$$\alpha(H,t) \geq (1-p_{xo})p(H,t) + \frac{p_{xo}}{N(H)-1}\sum_{i=1}^{N(H)-1} p\left(L(H,i)\cap G(H),t\right) p\left(p(U(H,i)\cap G(H),t\right)$$

The equation becomes an equality when all the programs in the population sample $G(H)$.

Subsequent research in (Poli, 2001a,b) extended the exact schema theorem from (Poli, 2000) based on microscopic properties (of single individuals) and produced a macroscopic version where only the selection probability of lower or same-order schemata appear in the formula.

Theorem (Macroscopic Exact GP Schema Theorem). The total transmission probability for a fixed-size-and-shape GP schema H under one-point crossover and no mutation is

$$\alpha(H,t) = (1-p_{xo})p(H,t) + p_{xo}\sum_j\sum_k \frac{1}{NC(G_j,G_k)}\sum_{i\in C(G_j,G_k)} p\left(L(H,i)\cap G_j,t\right) p\left(U(H,i)\cap G_k,t\right)$$

Using a cartesian node reference system, (Poli and McPhee, 2003a,b) extended the exact GP schema theorem to mutation and other crossover operators.

4.4 Advanced Concepts in Genetic Programming

4.4.1 Genotype-Phenotype Maps

Since the Modern Synthesis and the integration of Mendel's work with modern evolutionary theory, the "gene as a blueprint" metaphor has become the generally accepted explanation for phenotypic development. The relationship between genotypes and phenotypes was believed to be one between two relatively regular and symmetric spaces, where genes as distinct entities determined different phenotypic traits.

Attempts to model this relationship can be found as early as the inception years of the Modern Synthesis. Most notably, Sewall Wright's *fitness landscapes* (Wright, 1932, 1967) proved to be useful mathematical instruments for visualizing the fitness (reproductive success) of different gene combinations. Wright's fitness landscapes were represented by three-dimensional plots where the allele frequencies at two loci or two phenotypic traits were shown on the x and y axes, while the fitness value of the given combination (as a function of genotype or phenotype frequencies) was shown on the z axis (Fig. 4.4). In general, a fitness landscape can be seen as a multidimensional space where genes represent $n - 1$ dimensions and fitness represents the last dimension.

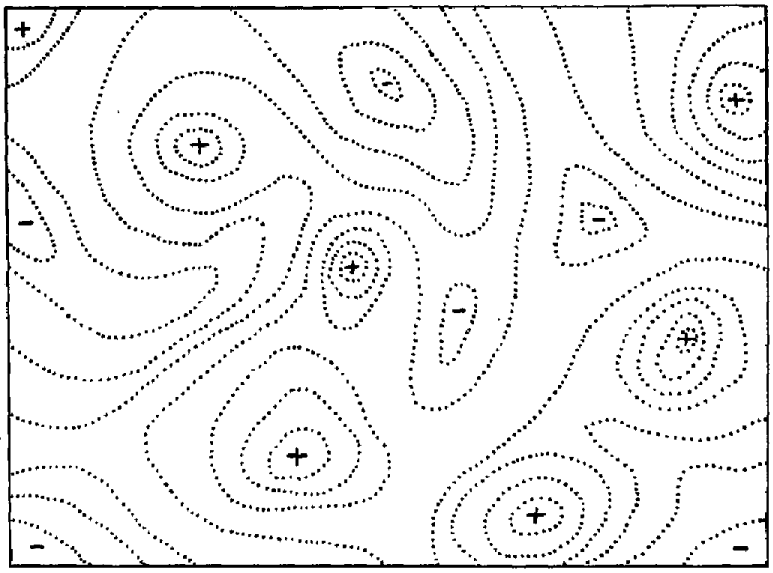


Figure 4.4: Sewall Wright's fitness landscape as originally shown in (Wright, 1932). Contours on the map connect points with equal fitness.

Fitness landscapes make it easier to conceptualize the genotype-phenotype relationship in terms of movement of populations across fitness landscapes. An important aspect investigated by Wright was the existence of mechanisms through which a population of genotypes can escape local peaks in the fitness landscape.

> "In a rugged field of this character, selection will easily carry the species to the nearest peak, but there may be innumerable other peaks which are higher but which are separated by "valleys". The problem of evolution as I see it is that of a mechanism by which the species may continually find its own way from lower to higher peaks in such a field. In order that this may occur, there must be some trial and error mechanism on a grand scale by which the species may explore the region surrounding the small portion of the field which it occupies. To evolve, the species must not be under strict control of natural selection. Is there such a trial and error mechanism?" (Wright, 1932)

In Wright's vision, populations could cross valleys in the fitness landscape through genetic drift, natural selection and competition (accompanied by migration and interbreeding) with other populations inhabiting neighbouring peaks. Wright believed that adaptive evolution was enabled by a *shifting balance* between these evolutionary forces.

The capacity to overcome local optima represents a desirable property not only of biological populations but also of optimization algorithms. Thus, the concept was readily adopted by other areas where optimization could be seen by analogy as crossing a rugged fitness landscape, full of hills and valleys. Fitness landscape analysis (FLA) deals with the properties of algorithms and problem spaces which enable such low-fitness or neutral areas to be crossed in the search for the global optimum. In optimization, FLA can offer a significant advantage by allowing the practitioner to pick the most appropriate optimization algorithm and settings according to the specific problem instances and the particularities of it's search space.

Using the analogy of evolution as a hill-climbing process, the question naturally arises: what enables a population of genotypes to cross valleys or flat areas in the fitness landscape? In order to keep evolving in the face of a changing environment, the population must be able to accumulate mutations at the genotypic level while maintaining a certain level of stability in the phenotype and producing appropriate phenotypic responses to environmental changes.

The advent of molecular genetics has marked a paradigm shift from bean-bag genetics and additive gene action to complex interactions within gene networks and epistatic effects bearing subtle consequences to phenotypic development.

A later model of (Kauffman and Levin, 1987) known as the "NK model" introduced a fitness landscape with tunable ruggedness using the two parameters: N – the number of components and K – the degree of interaction between components. The "NK model" defines a fitness function over binary strings of length N:

$$F(S) = \sum_i f(S_i)$$

where $f(S_i)$ represents the individual contribution for each position (or locus) i and is

dependent on the value of K other loci:

$$f(S_i) = f(S_i, S_1^i, ..., S_K^i)$$

The advantage of the NK model is that it accounts for the epistatic effects in the genotype-phenotype map. In order to account for neutral variation which was observed in biology, the NK model was extended to include degeneracy for certain components so that their contribution is either zero or bound to a limited range of contribution values (Geard et al., 2002).

Modern developments in molecular genetics and developmental biology have shown that genes do not encode information directly but are part of larger gene networks and are therefore context-dependent. This discovery marked an important deviation from the idea that genes were independent entities which determined distinct phenotypic traits.

In this context, systems biology has become an increasingly important field of research. A systems-level approach is necessary to leverage the complexity of genotype-phenotype relations (Davila-Velderrain and Alvarez-Buylla, 2014). Although selection acts on phenotypes, phenotypic variation is the product of both genetic and epigenetic processes. This lead to the idea of the genotype-phenotype (G⟶P) map (Alberch, 1991): a mathematical structure whose properties can be used to investigate the patterns of phenotypic evolution in an evolving system. In Alberch's view, the mapping from genotype to phenotype is determined by non-linear interactions at the molecular, cellular and tissue levels. The interactive nature of developmental processes implies the existence of a cyclical/feedback mechanism for gene expression – in other words, gene expression itself is under epigenetic control (Alberch, 1991).

Alberch defined the G⟶P mapping as a function of a given set of developmental parameters which (at least some) could be mathematically described. He derived his model from previous work on pattern formation models such that morphological diversity is generated by regulation (perturbation) in parameter values or initial conditions.

Fig. 4.5 illustrates how many combinations of parameter values can result in the same phenotype as there is no one-to-one correlation between genetically or environmentally mediated changes in parameter values and phenotypic transformation.

According to (Balari and Lorenzo González, 2008), Alberch anticipated the application of contemporary complexity theory to organic development. He contended that changes in the underlying parameter space are the main source of evolutionary processes and that developmental systems possess the properties of complex dynamic systems.

(Pigliucci, 2010) explain the properties of the G⟶P map in their discussion of Alberch's concept:

1) The G⟶P map is complex: the same phenotype may be obtained from different combinations of genetic informational sources

2) The area in parameter space where a particular phenotype exists gives an indication to how stable it is likely to be against environmental and genetic perturbation

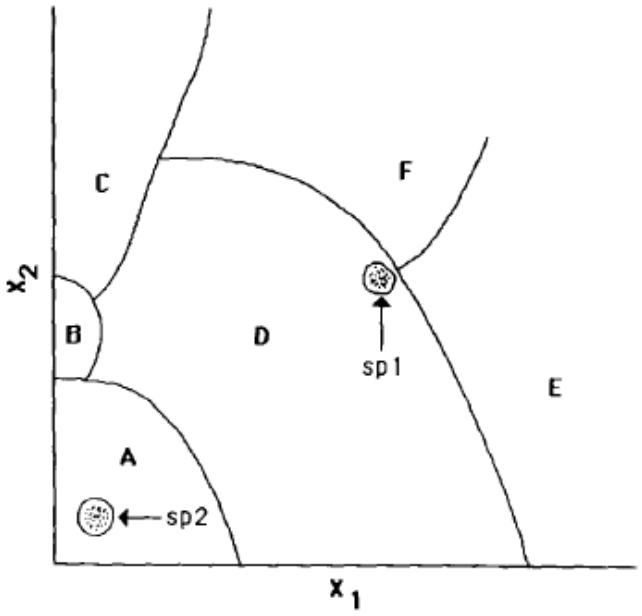


Figure 4.5: Hypothetical parameter space composed of six phenotypes (A, B, C, D, E, F) determined by the developmental interactions of two parameters x_1 and x_2. Image reproduced from (Alberch, 1991).

3) The parameter space is marked by transformational boundaries: areas where a small change will cause a transition from one phenotypic state to another

4) The phenotypic stability of a given population will depend on which area of the parameter space it occupies, and in particular whether it is close to a transformational boundary or not

The properties of the G→P map determine genetic operator behavior and topological properties of the genotype-phenotype space (Stadler and Stephens, 2003). When multiple genotypes can be mapped to the same phenotype and hence to the same fitness, we can speak of an additional degree of freedom at the level of selection. In the presence of genotypic degeneracy, the system evolves along preferred neutral network directions.

Patterns of phenotypic evolution such as punctuation, irreversibility and modularity are also determined by the properties of the G→P map (Stadler and Stadler, 2006). Stadler builds on the ideas from (Fontana and Schuster, 1998) who defined the notion of "nearness" between phenotypes as the probability of one phenotype being accessible from another through changes in the genotype. Phenotypic neighbourhoods defined this way are induced by the G→P map and cannot be described simply by comparing morphological features.

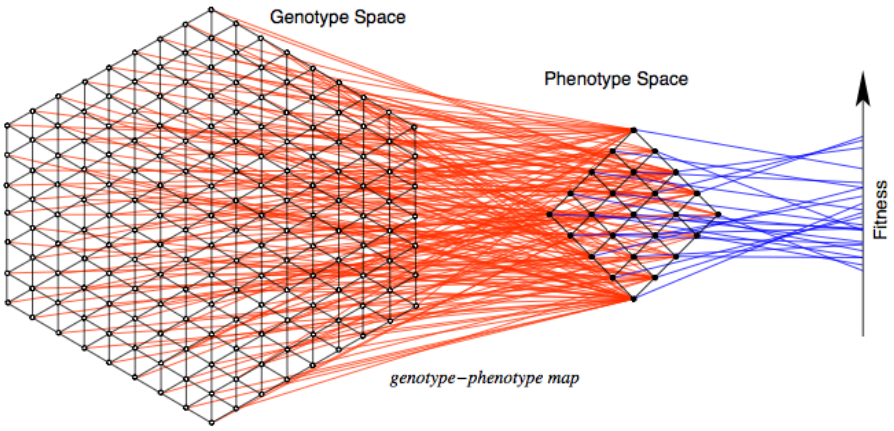


Figure 4.6: Genotype-phenotype map. Source: (Stadler and Stephens, 2003)

In genetic programming, the non-injectiveness of the G⟶P map results from the closure property of the primitive set. Phenomena such as bloat further contribute to the many-to-one structure of the G⟶P map. It is therefore important to consider GP evolutionary dynamics in the context of non-injective many-to-one G⟶P mappings.

From a theoretical standpoint, understanding the relationship between genotypes and phenotypes in GP boils down to understanding the behavior of its operators: selection, crossover and mutation. While the theory has seen substantial progress in the last decade with important results about crossover, selection, and the existence of GP schemas, it remains still unclear how the overall interplay between operators at the genotype and phenotype level affects evolutionary dynamics.

In this context, the empirical investigation of GP aspects such as the evolution of diversity can bring new insight on the properties of the G⟶P map. For example, (Winkler et al., 2016) use structural and semantic similarity measures to investigate the relationship between genotype and phenotype similarity for standard GP as well as two adaptive GP variants, namely OSGP (Affenzeller and Wagner, 2003) and ALPS (Age Layered Population Structure) GP (Hornby, 2006). They find significant differences between the dynamics and G⟶P mappings of different algorithmic variants. This result confirms the intuition that GP operators (selection in particular) have a big influence on the search behavior of the algorithm, in terms of its ability to perform an efficient exploration of the solution space.

4.4.2 Emergence, Evolvability and Robustness

"Emergence" is not a term that can be easily explained; it is rather an elusive concept. It is a favoured term of holistic thinkers that share the Aristotelic view of the "whole", that is "something over and above its parts, and not just the sum of them all". According to (Corning, 2002), the term "emergent" was coined by psychologist G. H. Lewes (Lewes, 1879), as a way to explain what the post-Darwinian scientists of the time considered a veritable

puzzle: the human mind. Lewis argued that emergent phenomena such as evolution can produce "qualitative novelties" that cannot be expressed in simple quantitative terms, these novelties being "emergents" rather than resultants:

> "[...] with emergents, instead of adding measurable motion to measurable motion, or things of one kind to other individuals of their kind, there is a cooperation of things of unlike kinds... The emergent is unlike its components in so far as these are incommensurable, and it cannot be reduced to their sum or their difference".

Economist Jeffrey Goldstein defined emergence as "the arising of novel and coherent structures, patterns and properties during the process of self-organization in complex systems". An important idea behind the concept of emergence is that "emergents", macro-entities that are formed from the micro-level components, exhibit properties and characteristics that take explanatory precedence over the parts of which the whole is made up (the idea of the "whole before its parts"). Emergent phenomena can appear in many kinds of physical or simulated systems, but they share certain interrelated, common properties (Goldstein, 1999):

1) *Radical novelty*: The emergents have features that are not previously observed in the complex systems under observation.

2) *Coherence or correlation*: Emergents appear as integrated wholes that tend to maintain some sense of identity over time.

3) *Global or macro level*: Since coherence represents a correlation that spans separate components, the locus of emergent phenomena occurs at a global or macro level. Observing emergents is observing their behavior at macro level.

4) *Dynamical*: Emergent phenomena arise as a complex system evolves over time. Emergence is associated with the arising of new attractors in dynamical systems.

5) *Ostensive*: Emergents are recognized by showing themselves (they are ostensibly recognized).

As it is already apparent, many of the properties of emergent systems can be readily identified in evolutionary systems as synergistic effects of the selection-variation loop: self-organization, modularity, repetitive patterns and the emergence of new, better structures. Although it may be argued that treating evolution as an emergent process is merely a philosophical distinction, the change in perspective has powerful implications; it brings into focus the necessity to investigate the fundamental mechanisms that allow a system to evolve and the existence of phenomena that depend on the global properties of the system as a whole rather than on any of its individual components.

Emergent behavior in biological systems is intimately connected with the concept of *evolvability*, and the "evolution of evolvability" (Pigliucci, 2008). Indeed, "natural selection

must work not on the individual mutation, but also on the very *mechanisms* that generate genetic variation – as does on all biological functions. [...] our genomes, and those of other life forms, have evolved mechanisms that *create* different kinds of mutations in their DNA, even to the point that there are genomic 'interchangeable parts'" (Caporale, 2002).

In GP, two of the most important phenomena, namely bloat and the emergence of building blocks cannot be explained by any of the genetic operators alone. They are rather emergent effects that occur not through any specific action but through an interplay of genotypic and phenotypic effects.

One of the first accounts of GP as an emergent process (Angeline, 1994) defines GP as a weak evolutionary method that employs empirical credit assignment on the solution representations to bias the search towards representations that are better suited for solving the problem. Empirical credit assignment allows the dynamics of a system to implicitly determine credit and blame; it is *holistic*, in the sense that credit is assigned to individuals and not their representational components (Spector et al., 1999).

In Angeline's view the fact that the algorithm did not rely directly on the representation but on its more abstract phenotypic properties implied the existence of another mechanism by which the representation is evolved in parallel to its phenotype. He defined evolvability as "the ability of a population to produce variants fitter than any existing" and theorized the existence of an emergent selection phenomenon in genetic programming coined as the "evolution of evolvability" (Angeline, 1994). Angeline further notes that it would be improper to speak of "schemas" in GP, as weak evolutionary methods have no direct correspondence between structures in the genotype and the features of the phenotype.

In a similar perspective, (Altenberg, 1994a) argued that adaptation is an emergent property achieved by selection on heritable variation: "because of the indeterminate size, structure and algorithmic properties of the tree-structured representation, both the genetics and the representation can evolve as an emergent property of the dynamics. Thus, GP has an intermediate combination of emergent and specified features". Altenberg further investigated the relationship between the "variational properties" of a program's representation and the way changes in its structure map to changes in its performance. He showed that evolvability changes during the algorithm run through the differential proliferation of subexpressions within programs.

(Wagner and Altenberg, 1996a) refined the concept of evolvability adding an alternative definition as the *propensity to vary*. Their definition made the important distinction between the existing variation in a population (a concrete, quantifiable measure of the existing differences) and the *variability* which represents a population's potential to produce adaptive variation under the effects of genetic operators. Wagner and Altenberg showed that evolvability as a property of a population's variability depends critically on the structure of the G→P map. Their notion of evolvability seems to be the inspiration for all ulterior definitions.

(Ebner et al., 2001) find that redundancy of the G→P map favors evolvability by increas-

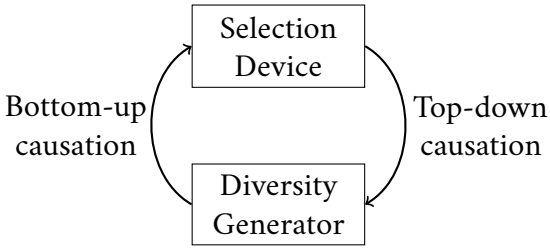


Figure 4.7: Variation–selection emergent loop in GP. Source: (Banzhaf, 2014)

ing accessibility between phenotypes, and also allows individuals to maintain higher fitness values as the majority of mutations will be neutral.

(Hu and Banzhaf, 2008) defined evolvability as "the capability of a system to generate adaptive phenotypic variation and to transmit it via an evolutionary process". They showed that many properties of an evolutionary system can be linked to evolvability and that improving a system's potential to evolve can have powerful implications for artificial evolutionary systems especially towards the goal of open-ended evolution. Furthermore, they argue that modularity, a property of complex systems materialized as a loose horizontal coupling between system entities at the same level, can be seen in the context of evolutionary systems as a mechanism to promote evolvability.

More recently, (Banzhaf, 2014) explained how "the variation-selection loop" is the driving force for GP emergent phenomena (Figure 4.7). They outlined the correspondence between emergent properties and GP elements:

- Variation in the genes (low level entities) of the artificial chromosomes provides bottom-up causation: genotypes are mapped to phenotypes which in turn are mapped to fitness values
- Selection is the mechanism for top-down (downward) causation, and fitness is the property selected for, which determines which combinations of genes survive.

Commenting on Banzhaf's essay, (Altenberg, 2014) acknowledged the occurrence of GP emergent phenomena as a result of the relationship between selection and the genetic operators. They also showed that through crossover a second level of replicator – the subtree is present within the population. The replication rate of full programs is controlled by their fitness, but the replication rate of subtrees is controlled by their effect on the change of fitness. In ending, Altenberg observed that as completely formal systems, emergent phenomena observed in GP are essentially mathematical by nature, believing them to be solvable sometime in the future.

Thus, evolvability is an important emergent property of evolutionary systems. In evolutionary computation, understanding the phenomena that influence the population's potential to evolve can be cast as a problem of understanding the relationship between representation, variation-producing operators and selection mechanism.

A closely related concept to that of evolvability is the concept of *robustness*, defined as a system's property to maintain its function against perturbations. More formally, we may accept the definition of (Alderson and Doyle, 2010), which states that: "a [property] of a [system] is robust if it is [invariant] with respect to a [set of perturbations]".

As (Kitano, 2004) points out, robustness – a fundamental feature of complex evolvable systems – should not be misunderstood to mean staying unchanged; rather, to maintain its specific functionality against perturbations, a system is often required to adjust its mode of operation in a flexible way. In other words, robustness allows changes in the structure and components of the system owing to perturbations, but specific functions are maintained. Thus, in an evolutionary system we may define phenotypic robustness as the invariance of fitness with respect to genotypic changes.

At a glance, robustness and evolvability share an antagonistic relationship, since by definition robustness decreases phenotypic variability. Therefore, robustness may potentially lead to evolutionary stasis. For example, in their simulations of evolving RNA structures, (Ancel and Fontana, 1999) observed a dramatic loss of phenotypic plasticity under natural selection, leading to extensive modularity and robustness, and ultimately leading to a loss of phenotypic variability where phenotypes remained locked in a suboptimal-state and the population became trapped in regions where most genetic variation is phenotypically neutral. Modularity supports robustness of the system as a whole by isolating the effects of perturbations to independent functional areas. However, (de Visser et al., 2003) suggest that the relationship between robustness and evolvability is unclear. They describe a number of scenarios in which robustness may enhance evolvability:

1) Hidden pleiotropic effects may promote evolvability in the genetic background of a robust trait. Then, evolvability becomes a selective advantage for the robust trait by facilitating adaptations of pleiotropically related characters.

2) Phenotypic robustness implies the existence of neutral networks in genotype space. The population may then move across these networks through genetic drift, reaching points with a higher adaptive potential. Therefore neutrality increases long-term population evolvability. Neutrality is also important from a purely theoretical perspective, as shown in (Igel and Toussaint, 2003a) (see Section 2.4).

3) Robust states may accumulate hidden genetic variation (or potential variation) that is only expressed when the genetic background changes (Rutherford, 2000). Such hidden variation may become (in the right conditions) the fuel for further evolution, thus increasing evolvability.

Another theory by (Kitano, 2004) suggests that robustness is a prerequisite for evolvability, as evolvable systems need to be robust against environmental and genetic perturbations. They also describe the main mechanisms through which robustness is achieved, namely genetic buffering and modularity. The genetic buffering mechanism is also referred to as an evolutionary capacitor (see point 3 above). (Hayden et al., 2011) also suggest that robustness

and epistasis can have a positive role as accumulated cryptic genetic variation may facilitate adaptive evolution. At the same time, genetic buffering may also be one of the contributing forces for bloat in genetic programming (see Section 4.2.3).

Recent investigations by (Hu and Banzhaf, 2016a,b) show that robust genotypes play a crucial role in the evolutionary process as they are visited more often and can guide the search to their adjacent phenotypes. They also find that neutrality improves evolvability, enabling evolutionary systems to produce novel more adaptive phenotypes. Additionally, their quantitative study suggests that robustness and evolvability may manifest differently at the genotypic and phenotypic level.

In summary, evolvability, robustness and modularity represent evolved, emergent properties of biological systems, playing an essential role in understanding GP evolutionary dynamics. It is therefore important to study them in relation to the G→P map and its intrinsic properties (such as redundancy), and also in relation to the selection, recombination and mutation operators which shape the genetic make-up of the population.

5 Tracing of Evolutionary Search Trajectories

In the context of optimization algorithms, the notion of a search trajectory implies the existence of a finite sequence of points in the solution space which the algorithm explores in its path from initialization towards the final solution. Understanding the algorithm search trajectories is particularly important in the case of metaheuristics as they can offer valuable insight into the properties of the search space and into the dynamics of the algorithm itself. This aspect is particularly relevant in cases such as GP when a solid mathematical foundation is not readily available.

In genetic programming, the search progresses through the interplay of genotypic and phenotypic search operators. The genotype space is where the solution representations live and get combined, while the phenotype or fitness space represents the evolutionary arena where the solution candidates compete for survival. As already outlined in Section 4.4.1, the non-injective mapping of genotypes to phenotypes to fitness makes it difficult to investigate and understand how the actual genetic variation at the level of the solution representation is exploited, preserved and improved by the algorithm. Thus, GP evolutionary trajectories need to be analysed within the framework of genotype-phenotype maps on both genotypic and phenotypic levels.

The following aspects are particularly relevant for the investigation:

1) The evolution of genotypic and phenotypic diversities
2) The effectiveness of genetic operators in producing new useful variation
3) The importance of genetic robustness and evolvability in the population
4) The inheritance of genetic material across lineages
5) Self-organization and the occurrence of schemata and building blocks

This thesis presents a new methodology for the analysis of the above-mentioned aspects and their synergistic effects on population dynamics. The methodology, consisting of novel computational tools and algorithms, was developed using HeuristicLab (Wagner, 2009; Wagner et al., 2014), a free (opensource GPLv3), flexible and powerful optimization framework developed by members of the Heuristic and Evolutionary Algorithms Laboratory (HEAL) at the University of Applied Sciences Upper Austria.

Testing and validation of the new analysis methods and algorithmic improvements was performed on a class of symbolic regression benchmarks as well as real-world problems. The next section is dedicated to HeuristicLab, providing a general overview of the framework and a detailed description of symbolic regression within HeuristicLab. The implementation and main features of the genetic programming algorithm in HeuristicLab is also described.

5.1 Symbolic Regression in HeuristicLab

Symbolic regression represents the task of finding the best mathematical model that best fits a given dataset. The idea introduced by (Koza, 1992) is to evolve mathematical expressions that approximate the evolution of a given data series.

For an in-depth discussion of HeuristicLab architecture and capabilities, we turn the reader to (Wagner et al., 2014). The main features of HeuristicLab include:

- Plugin-based architecture: every new algorithm or methodology can be encapsulated in a plugin.
- Powerful algorithm model (Fig. 5.1) divided into three layers for maximum flexibility: the data model, the operator model and the execution model.
- Automatic experiment creation according to user-specified number of repetitions and parameter grid
- Comprehensive and easy to use analysis tools (charts, box plots, statistical measures)

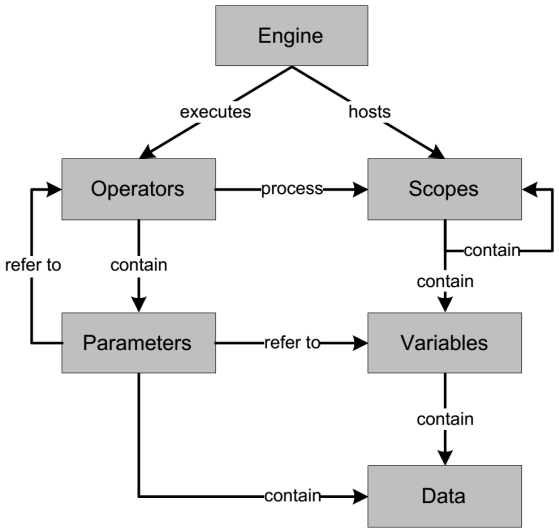


Figure 5.1: The HeuristicLab algorithm model. Source: (Wagner et al., 2014)

Symbolic regression in HeuristicLab is implemented according to the HL algorithm model, and consists of:

1) The symbolic expression tree encoding
2) The operators which manipulate the encoding (ie., tree creation, mutation, crossover)
3) The algorithms which define the rules by which the operators interact with the encoding and the data, such as genetic programming in its various incarnations: GP, OSGP, NSGA-II, SASEGASA, RAPGA, etc. More details in (Affenzeller et al., 2009).

In HeuristicLab, objects that represent functions acting on the encoding and executed by the execution engine are called operators. According to Fig. 5.1, operators are configured via parameters which are saved as scope variables, and can be executed sequentially or in parallel. The scopes themselves can be nested following a tree structure such that different operators can share the same parameters and retrieve their values by doing a scope tree lookup. A special category of operators are the *analyzers*, which are usually employed to gather information about the state of the population at any given moment. Examples of analyzers for the symbolic expression tree encoding are those which calculate the average population fitness, average tree length, symbol or variable frequencies, number of evaluated solutions, and so on.

At the level of the encoding, linear scaling (Keijzer, 2003) is employed in order to improve the evolvability of the population. In combination with the Pearson's R^2 fitness function which is independent of scale and offset, this allows the algorithm to avoid the additional effort of evolving the individual tree structures and parameters in the correct output range.

Another property of the symbolic expression tree encoding is that it can be configured to accept different configurations of symbols (functions and terminals) according to the desired function and terminal set. The symbol configuration, also called the *grammar*, defines the rules that the genetic operators need to respect, such as what kind of nodes to create, how many children and of what kind a node is allowed to have and so on. For implementation reasons, each symbolic expression tree includes two additional nodes at the top, a *program root* node and a *start node* which are immutable. This practically means that every tree in the population will have an *effective* length equal to the actual length minus the two fixed nodes.

Concerning the genetic operators, HeuristicLab supports most of the operators described in Section 4.2 such as:

- ◇ The grow, full, ramped-half-and-half and PTC2 tree creation (Section 4.2.1)
- ◇ Proportional, linear rank, tournament, random, gender-specific and offspring selection (Section 4.2.2)
- ◇ One-point, context-aware, semantic similarity and probabilistic functional crossover operators (Section 4.2.3)

◇ Point, uniform and structural mutation (subtree removal or replacement)

In summary, HeuristicLab offers all the tools and algorithms necessary for solving symbolic regression problems. In order to take advantage of them, the evolutionary tracking methodology described in this thesis was developed as a HeuristicLab plugin on top of existing functionality, adding new operators which offer new insight and new functionality to the genetic programming algorithm.

5.2 Analysis of Evolutionary Dynamics

5.2.1 Population Diversity

An overview of different approaches for preserving GP population diversity was given in Section 4.2.2. In this work, diversity at both phenotypic and genotypic levels was analysed with the help of two new measures: one correlation-based measure for semantic similarity and one structural measure (based on a distance similar to the edit distance) for genotypic similarity.

The similarity measures presented below are informally regarded as distances although they do not satisfy the triangle inequality (so in this regard, they are not true metrics). They are however symmetric, so that for a population of n individuals, only the upper triangle portion of the similarity matrix needs to be calculated, so a number of $\frac{n(n-1)}{2}$ similarity calculations needs to be performed. Therefore, the average similarity measure S will have the general formula:

$$\overline{S} = 2 \cdot \frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} S(T_i, T_j)}{n(n-1)} \tag{5.1}$$

To enable an easier analysis and display of similarity values, the similarity measures described below have been explicitly designed to return values in the interval $[0, 1]$, where 0 means complete dissimilarity and 1 means complete similarity. Average population diversity is obviously given by $1 - \overline{S}$.

Phenotypic Similarity

Phenotypic or *semantic* similarity provides a measure of how closely two individuals behave regarding their response on the training data. The similarity is calculated using the Pearson R^2 coefficient of determination:

$$R^2_{X,Y} = \left(\rho_{X,Y}\right)^2 = \left(\frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y}\right)^2 \tag{5.2}$$

Since the R^2 coefficient is undefined when $\mathrm{Var}(x) = 0$ or $\mathrm{Var}(y) = 0$, phenotypic similarity between two trees with constant responses on the training data cannot be

calculated directly. In this special case, the similarity between the two trees is considered to be 1 if both trees have a constant response, and 0 otherwise[1].

Thus, for two trees T_1 and T_2, with responses $r_1 = response(T_1)$ and $r_2 = response(T_2)$, we calculate phenotypic similarity using the formula:

$$PhenotypicSimilarity(T_1, T_2) = \begin{cases} 1 & \text{if } \mathrm{Var}(r_1) = \mathrm{Var}(r_2) = 0 \\ 0 & \text{if } \mathrm{Var}(r_1) = 0 \text{ or } \mathrm{Var}(r_2) = 0, \\ R^2_{r_1,r_2} & \text{otherwise} \end{cases} \tag{5.3}$$

Genotypic Similarity

Genotypic similarity between two tree individuals reflects the degree to which their respective structures overlap. Historically, the tree edit distance and its variants were used to calculate GP structural similarity (Ekárt and Németh, 2000; O'Reilly, 1997). However, calculating the edit distance between labeled, unordered trees was shown to be NP-complete (Zhang et al., 1992).

We employ a distance measure based on the largest common forest between two trees, called the *bottom-up tree distance* (Valiente, 2001). The bottom-up tree distance represents a particular case of the isolated-subtree distance (TANAKA and TANAKA, 1988). The algorithm (implemented in HeuristicLab) has the following advantages:

- Runtime complexity linear in the size of the trees
- Compatible with ordered or unordered labeled trees (with the same complexity)
- No extra edit operations or costs need to be defined
- Configurable tree matching behavior

The bottom-up tree distance is computed in two steps described below. We consider two trees T_1 and T_2 and the largest common forest $F = T_1 \uplus T_2$ (consisting of the disjoint union between the two trees).

1) The forest F is compacted to a directed acyclic graph G, during a bottom-up traversal of F (in the order of non-decreasing node height). Two nodes in F are mapped to the same vertex in G if they are at the same height and their children are mapped to the same sequence of vertices in G. The bottom-up traversal ensures that children are mapped before their parents, leading to $O(|T_1| + |T_2|)$ build time for G. This step returns a map $K : F \to G$ required to compute the bottom-up mapping.

[1] A drawback is that similarity will be zero if one of the variances is 0, regardless how close to zero the other one may get. Experiments with this method indicate that this scenario (when only one of the variances is zero) occurs in 5% to 10% of all similarity calculations, therefore have no meaningful impact on the average.

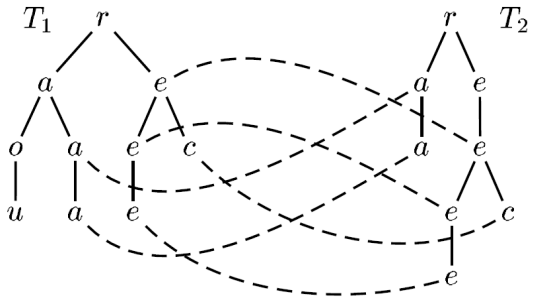


Figure 5.2: Example bottom-up tree mapping. From (Valiente, 2001)

2) The second step iterates over the nodes of T_1 in level-order and builds a mapping $M : T_1 \longrightarrow T_2$ using K to determine which nodes correspond to the same vertices in G. The level-order iteration guarantees that every largest unmapped subtree of T_1 will be mapped to an isomorphic subtree of T_2. Similarity between two trees T_1 and T_2 is computed from the mapping M using the Sørensen-Dice coefficient[2]:

$$BottomUpDistance(T_1, T_2) = \frac{2 \cdot |M|}{|T_1| + |T_2|} \tag{5.4}$$

The above formula will always return a similarity value between $[0, 1]$. Matching behavior can be configured as "strict" or "relaxed" (structural matching only, ignoring constants and variable weights) by changing the way node labels are defined in the algorithm.

5.2.2 Genetic Operator Effectiveness

The effectiveness of genotypic operators is measured in terms of fitness, in order to see how often an operator can produce adaptive (as opposed to deleterious) changes. Operator effectiveness can bring important insight about the behavior of the genetic operators and the dynamics of the evolutionary process.

- ◇ The *average fitness improvement* is calculated by subtracting the parent fitness from the child fitness. In the case of crossover, the root parent is considered. For a population of N individuals, the average fitness improvement is given by:

$$\overline{q} = \frac{1}{N} \sum_{i=1}^{N} (Fitness(t_i) - Fitness(p_i)) \tag{5.5}$$

where t_i is a child individual and p_i is its parent.

[2] Another possibility would have been to use the *Jaccard index* $J(T_1, T_2) = \frac{|M|}{|T_1 \cup T_2|}$, however this would've been slightly more computationally expensive due to the calculation of the set union $T_1 \cup T_2$.

◇ The *best fitness improvement* is given by the difference between the fitness values of the best individual and its parent:

$$q_{best} = Fitness(t_{best}) - Fitness(p_{best}) \tag{5.6}$$

The reasoning behind the formula is to get a more accurate picture of the underlying operator dynamics. For each of the operators (crossover or mutation) the best child was selected according to the best quality. This means that the best fitness improvement does not reflect the greatest relative parent-child quality difference, but rather the improvement that could be made in terms of the overall best quality among all offspring.

5.2.3 Heredity, Inheritance and Genealogy Analysis

Genealogy analysis, providing lineage and inheritance information, represents a less explored aspect of GP evolution that can offer important insight into the dynamics of the selection-variation loop and the propagation of genetic material. At its core, the methodology developed in this section allows us to:

1) Build the complete genealogies of all individuals in the population during an algorithmic run
2) Record all the inherited genetic information, consisting of genes (subtrees) and their positional information in the parent and child chromosomes

Genealogy Graph: Recording Evolution

The term genealogy comes from the Greek words γενεά (genea) and λόγος (logos), literally translated as "generation knowledge". Genealogies are also known as family trees, due to their tree-like shape where the root of the tree represents the individual whose ancestry is being investigated.

When communities of individuals reproduce over the generations with no significant influences from external factors such as migration, it is often the case that most individuals are related to a certain extent, sharing common ancestors. From a representation perspective, the presence of common ancestors in the family history means that the genealogy is no longer a tree but a graph. If we assign a direction to the edges in the graph according to the direction of inheritance (from ancestor to descendant), we obtain a directed graph structure in which vertices represent individuals and arcs represent hereditary relationships.

For the analysis of population genealogies in GP, a directed graph data structure was implemented in HeuristicLab for storing individual genealogies and hereditary information.

Terminology

The information stored by the genealogy graph in its arcs and vertices is described below:

- ◇ Vertices represent individuals in the directed graph and hold information regarding each individual's structure, fitness and generation.
- ◇ Arcs represent hereditary relationships and hold information regarding the genes inherited by the offspring from their parents.

The following terminology is used to describe the exchanges of genetic information recorded in the genealogy graph:

- ◇ When performing crossover between two parent individuals, the first parent will contribute its whole root structure minus one subtree which will be replaced by a subtree contributed by the second parent. We denote the first parent as the *root parent* and the second parent as the *non-root parent*.
- ◇ We use the name *genetic fragment* (or simply, *fragment*) to denote a subtree that was inherited through a genotypic operation. For example, the subtree swapped by crossover from the non-root parent or the one altered by mutation. *Fragments* are stored in the arcs of the genealogy graph (since arcs represent the direction of the inheritance) along with positional information that allows them to be readily accessed from the structure of the child individual or the structure of the non-root parent (when applicable).

Building the Genealogy Graph

When crossover is followed by mutation, the results of both operators are stored in the

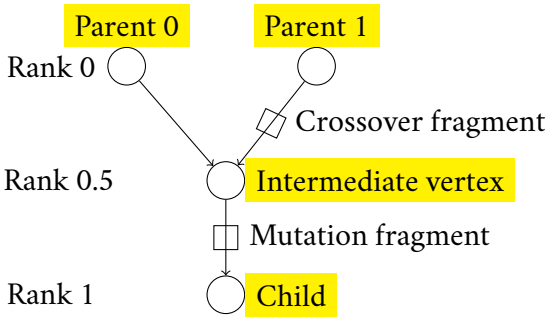


Figure 5.3: Intermediate crossover child saved as a vertex in the genealogy graph, where the vertex rank represents the generation. The intermediate rank value (0.5) means that the intermediate child is not part of the population at generation 1.

graph, first the intermediate crossover individual and then the final offspring after mutation (Fig. 5.3). As the population size, number of generations, mutation and crossover rates are fixed, we can approximate the number of vertices and arcs in the graph. For example, if we consider the genealogy graph $G(V, A)$ of a population of n individuals running for g generations with mutation probability p_m and crossover probability p_c, the expected values

for the number of vertices and arcs will be:

$$E\big[|V|\big] = p \cdot (g+1) + (p \cdot g \cdot p_c \cdot p_m) \tag{5.7}$$

$$E\big[|A|\big] = (2 \cdot p_c + p_m) \cdot p \cdot g \tag{5.8}$$

The two terms in Eq. (5.7) can be explained as follows:

- The first term $(g+1)$ represents the g generations of individuals plus the initial population. The $p \cdot (g+1)$ vertices correspond to the individuals in the population at each generation.
- The second term represents the number of intermediate vertices added to the graph when crossover and mutation are applied in succession. The probability of this event is $p_m \cdot p_c$. At the same time, since the last generation does not participate in reproduction, only g generations need to be considered, giving the quantity $p \cdot g \cdot p_c \cdot p_m$.

The expected number of arcs in Eq. (5.8) follows a similar line of reasoning:

- Two arcs are added from the two crossover parents to the child
- One arc is added when a new individual is obtained via mutation

Fig. 5.4 shows a genealogy graph obtained from a population of 30 individuals that ran for 20 generations. Fitness is represented in the graph with the help of a heatmap where warm colors represent better fitness. Fig. 5.4a shows the fitness distribution of the whole population per generation and Fig. 5.4b shows the genealogy of the best solution obtained by the algorithm.

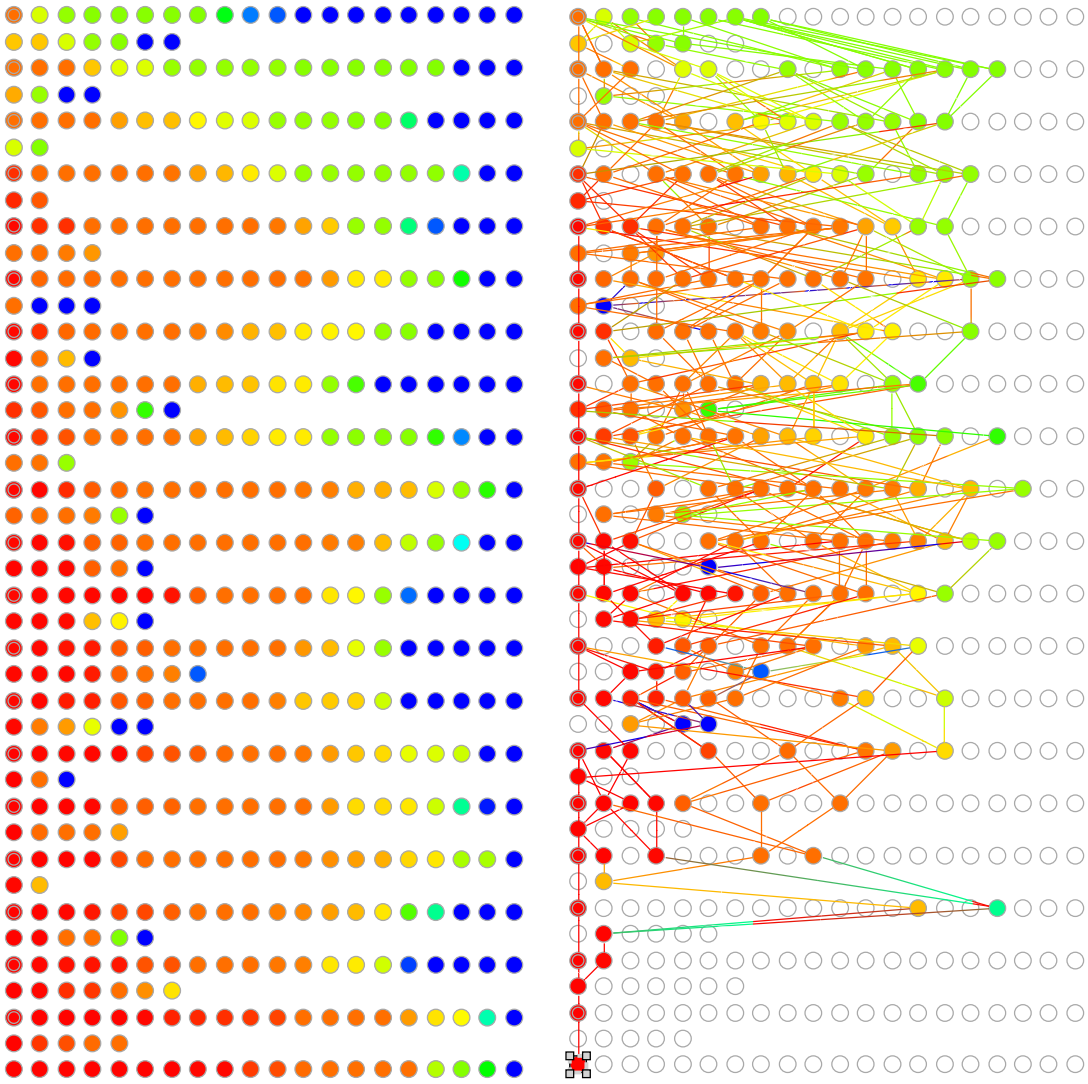
Ancestries and Root Lineages

At any given generation the genealogy graph can be queried for information regarding an individual's concrete evolutionary history, such as its ancestors, descendants or root lineage. Let $G = (V, E)$ be a genealogy graph and let $v \in V$ be a vertex representing an individual:

- We define the *ancestors* of v as the set of vertices $u \in V$ such that for each u, there exists a directed path from u to v.
- We define the *descendants* of v as the set of vertices $w \in V$ such that for each w, there exists a directed path from v to w.
- We define the *root lineage* of v as the directed path from v's oldest ancestor to v where every vertex is a root parent.

Genealogy graphs can be used to extract information about algorithm dynamics and genetic operator behavior, such as:

- Fragment length (ie., the average size of the subtrees swapped by crossover)



(a) Population genealogy **(b)** Genealogy of the best solution

Figure 5.4: Example genealogy graph

- ◇ Reproductive success, defined as the amount of children produced by each individual of the previous generation
- ◇ Effects of the selection mechanism on the population
- ◇ Evolution of quality across lineages

Trace Graphs: Evolutionary Trajectory Analysis

Genealogy analysis can offer important insight into the dynamics of the evolutionary search. However, further analysis is necessary to explain the exact mechanisms of inheritance and the influence of properties like evolvability (discussed in Section 4.4). For the detailed analysis of inheritance patterns a new methodology entitled *subtree tracing* was developed, which can decompose any subtree in the population (any genotypic structure) to the sequence of genotypic operations that have built up its structure.

We define a subtree's *trace graph* as a collection of vertices representing the ancestor individuals from which parts of its genotype were inherited, connected by arcs representing the genetic operations involved in the transfer of said parts. Therefore, the set of vertices included in a trace graph represents a subset of the vertices contained in the genealogy graph. In the following, we use the term *subtree* to refer to the subtree whose trace graph we want to calculate.

Essentially, the tracing algorithm walks the genealogy graph from a given starting point and follows a set of predefined rules for deciding which direction to consider. The rules depend on the relative positions of the traced subtree and the genetic fragment injected into the current individual's structure by one of the recombination operators. Finally, the algorithm selectively adds the visited vertices to the trace graph (connected by arcs showing gene inheritance). The resulting trace graph represents a compact enumeration of the events that have lead to the formation of the traced subtree.

Algorithm Description

It is clear that the subtree tracing algorithm needs to consider the effects of both crossover and mutation. We regard mutation and crossover as recombination operators which differ in the number of parents required for producing an offspring. Both operators produce offspring by applying a change on the structure of the root parent. The change can be of random origins (mutation), or it can originate from the non-root parent (crossover).

In the general case, the subtree that needs to be traced can be partially inherited from the root parent, with the other part either inherited from the non-root parent or introduced by mutation. Thus, the tracing procedure can be implemented as a recursive strategy decomposable to smaller subtrees that need to be individually traced. For simplicity and without loss of generality, we describe the procedure only for the more general case of crossover.

Since two parents and two inherited structures are involved, the algorithm needs to branch itself out and walk the genealogy graph in two directions: the direction of the root parent (tracing the original inherited structure) and the direction of the non-root parent (tracing the inherited genetic fragment).

In order to correctly identify the subtrees that need to be traced, the algorithm uses positional information stored in the genealogy graph as tree node indices in the ordering given by a preorder tree traversal. For example, in Fig. 5.5, a crossover operation exchanged the subtrees with preorder indices 4 and 1 in the root parent and non-root parent, respectively. In order to trace down the whole structure of the child individual (the subtree with the root at preorder index 0), the tracing algorithm will have to follow its structure in the root parent (at preorder index 0) and in the non-root parent (at preorder index 1). In the general case, the tracing algorithm must be able to start tracing from any given preorder index.

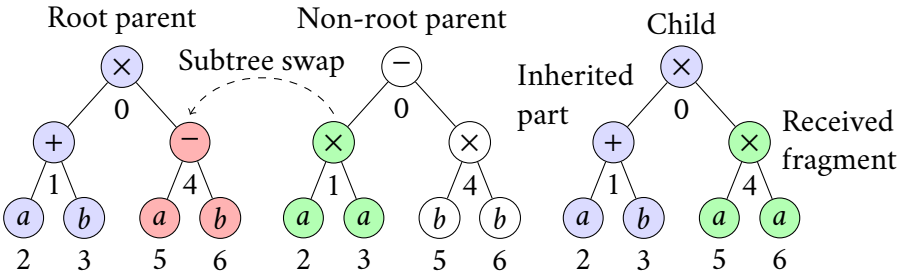


Figure 5.5: Preorder arithmetics for subtree inclusion

When tracing arbitrary subtrees, the rules for walking over the genealogy graph depend on the relative position between the traced subtree and the genetic fragment swapped by crossover. A set of simple arithmetic rules is used to determine the inclusion relationships that define the decision points in the algorithm:

1) The traced subtree contains the fragment
2) The fragment contains the traced subtree
3) The traced subtree and the genetic fragment are distinct

Let s_i, s_l be the preorder index and length of the traced subtree and f_i, f_l the preorder index and length of the fragment. Table 5.1 describes the algorithm's decision process, where the last column of the table represents the corresponding line numbers from Algorithm 3.

We introduce three methods:

- The `TraceRecursive` method represents the main entry point of the algorithm and implements the logic for walking the genealogy graph. Its arguments specify the starting point of the tracing procedure (genealogy graph vertex *current* and subtree index s_i) and the previously inserted trace graph vertex *last* along with the preorder

Condition	Description	Decision	Line
1) $(s_i < f_i) \wedge (f_i < s_i + s_l)$	Fragment $\subset$ Subtree	Trace both	26 – 30
2) $(f_i < s_i) \wedge (s_i < f_i + f_l)$	Subtree $\subset$ Fragment	Trace fragment	18 – 20
3) $(s_i + s_l < f_i) \vee (f_i + f_l < s_i)$	Subtree $\cap$ Fragment $= \emptyset$	Trace subtree	22, 23, 32, 33

Table 5.1: Tracing cases

indices $s_{i,last}$ and $f_{i,last}$ of the previously-traced subtree and fragment. The previously-inserted vertes *last* and its associated trace data are used by the procedure to connect vertices with arcs in the trace graph.

In cases 2) and 3) from Table 5.1 when no branching is involved (the fragment is not contained by the subtree), the algorithm can simply skip the current genealogy graph vertex and continue from the parent vertex. When doing so, the preorder index s_i of the traced subtree in the parent needs to be adjusted when $f_{i,p_0} + f_l < s_i$, since the fragment length may be different than the length of the subtree contained by the parent at the same index. This adjustment is done at line 23 in Algorithm 3.

Walking over genealogy graph vertices when their content is not relevant to the currently traced subtree represents a big advantage as it leads to compact trace graphs which show in a clear manner how structures are assembled by the algorithm. At the same time, compactness of the trace graph automatically implies the existence of genetic hitchhiking when for example an arc depicts the transfer of a genetic fragment from an ancestor to a descendant several generations apart.

◇ As the name implies, the `AddTraceNode` method (Algorithm 4) adds a new vertex to the trace graph. We remind that vertices of the genealogy and trace graphs uniquely represent individuals from the population. Since common ancestors are frequently encountered in GP lineages, the algorithm uses a caching mechanism to remember already added vertices.

◇ The `ConnectLast` method connects the newly added trace graph vertex to the previously added one (adding an arc between *current* and *last*) – see Algorithm 5. Another fine point in the implementation of the tracing algorithm is given by the handling of common ancestors, as their corresponding vertices in the trace graph can be seen as "intersection points" for multiple evolutionary paths. Multiple evolutionary paths going through the same pair of graph vertices can be distinguished from one another based on the genetic information that was transferred on each path. This gives rise to a couple of important consequences:

1) The trace graph is a *multigraph* (it can contain multiple arcs connecting the same pair of vertices, depending on the evolutionary path)

2) Arcs need to hold positional information of the *subtree* and *fragment* (see above

terminology)

The `ConnectLast` method takes all these requirements into account and performs a caching of arcs according to the subtree and fragment positions. Each arc is uniquely identified by the *trace data* tuple $\{s_{i,current}, f_{i,current}, s_{i,last}, f_{i,last}\}$ containing the subtree and fragment indices in the *current* and *last* vertices.

In addition to the caching mechanisms necessary to ensure the correct tracing of genetic fragments, the algorithm employs an additional cache (implemented as a hash set) for the argument sets of the `TraceRecursive` method. This is done simply to ensure that no effort is duplicated when the algorithm revisits some of the genealogy graph vertices[3]. Another speed optimization that was used in the implementation was to cache the preorder lists of tree nodes such that when a tree node is accessed by its preorder index, only the first access will be $O(n)$, with all the subsequent accesses from the same tree being $O(1)$.

Trace-based Analysis Methods

We have shown in the previous sections how genealogy information recorded during the run of the algorithm can be used to enable more powerful analysis methods based on subtree tracing.

The tracing methodology can identify the complete trace graph (showing gene propagation from ancestors to descendants) of any subtree in the population. When applied on the whole population (ie., by tracing the root of every tree), trace graphs can reveal the patterns of evolution (specific to the given problem instance) such as the most influential ancestors in the population (ie., those whose genes propagated with the greater frequency in the population) or the most sampled subtrees with respect to crossover and mutation.

Contribution Ratio

The idea behind the contribution ratio is to define a measure of the effort spent by the algorithm in order to achieve useful adaptation. Thus, we define the contribution ratio r as the percentage of an individual's ancestry from which its structure was inherited:

$$r = \frac{|Trace(individual)|}{|Ancestry(individual)|} \tag{5.9}$$

Identification of Problem Building Blocks

Trace graphs represent useful tools for representing the origins of inheritance in a compact way. Aggregating trace graphs over the whole population can provide useful statistical information regarding the most sampled subtrees in the population. The aggregated data is

[3] For a graph of ≈125.000 vertices, this optimization alone makes a difference from 10-11 minutes to under 1 second when tracing the best solution of the algorithm. Furthermore, using a persistent cache between the calls to `TraceRecursive` will bring increasing speed benefits as the cache grows.

```
1  Method TraceRecursive(current, s_i, last, s_{i,last}, f_{i,last})
       Input: Genealogy graph vertex current representing a tree, subtree preorder index
              s_i, trace graph vertex last, preorder indices s_{i,last}, f_{i,last}
       Output: The trace graph of the specified subtree
2      g ← current;
3      while deg⁻(g) > 0 do
4          arcs ← InArcs(g);
5          fragment ← fragment received from the non-root parent;
6          s_l ← the length (number of nodes) of the traced subtree;
7          p_0 ← the source of the first arc (the root parent);
8          f_{i,p_0} ← the preorder index of fragment in the root parent p_0;
9          f_l ← the length (number of nodes) of fragment;
10         if |arcs| = 2 then
11             p_1 ← the source of the second arc (the non-root parent);
12             f_{i,p_1} ← the preorder index of fragment in the non-root parent p_1;
13             if f_{i,p_0} = s_i then
14                 g ← p_1;
15                 s_i ← f_{i,p_1};
16                 continue;
17             if f_{i,p_0} < s_i then
18                 if f_{i,p_0} + f_l > s_i then
19                     g ← p_1;
20                     s_i ← s_i + f_{i,p_1} − f_{i,p_0};
21                 else
22                     g ← p_0;
23                     s_i ← s_i + |NodeAt(g, f_{i,p_0})| − f_l;
24                 continue;
25             if f_{i,p_0} > s_i then
26                 if f_{i,p_0} < s_i + s_l then
27                     last ← AddTraceNode(g);          // current node becomes last
28                     TraceRecursive(p_0, s_i, last, s_i, f_{i,p_0});
29                     TraceRecursive(p_1, f_{i,p_1}, last, s_i, f_{i,p_0});
30                     break;
31                 else
32                     g ← p_0;
33                     continue;
34         if |arcs| = 1 then
35             Trace mutation – see Algorithm 6 ;
36     if last ≠ null then
37         current ← AddTraceNode(g);
38         td ← ConnectLast(current, last);
```

Algorithm 3: Recursive subtree tracing - Trace Method

```
1  Method AddTraceNode(g)
       Input: Genealogy graph vertex g.
       Output: The trace graph vertex v corresponding to g.
2      v ← null;
3      if g has already been mapped to a node in the trace graph then
4          v ← trace node corresponding to g;
5      else
6          v ← Copy(g);
7          Add v to the trace graph;
8      return v;
```

Algorithm 4: `AddTraceNode()` method

```
1  Method ConnectLast(current, last, td)
       Input: Current trace graph node current, last node last, trace data td
       Output: The trace graph node corresponding to g
2      if no arc with trace data td exists from current to last then
3          add an arc current → last;
```

Algorithm 5: `ConnectLast()` method

```
1  if |arcs| = 1 then
2      if s_i = f_{i,p_0} ∨ (s_i < f_{i,p_0} ∧ f_{i,p_0} < s_i + s_l) ∨ (f_{i,p_0} < s_i ∧ s_i < f_{i,p_0} + f_l) then
3          last ← AddTraceNode(g);                    // current node becomes last
4          i ← min{s_i, f_{i,p_0}};
5          TraceRecursive(p_0, i, last, s_{i,last}, f_{i,last});
6          break;
7      else
8          g ← p_0;
9          if f_{i,p_0} < s_i then
10             s_i ← |NodeAt(g, f_{i,p_0})| − f_l;
```

Algorithm 6: Subtree tracing – Mutation

calculated using a number of simple additions to the implementation of Algorithm 3 such that:

- ◇ The `TraceRecursive` procedure uses a persistent caching scheme when tracing all the individuals in the population
- ◇ A node weight associated to each tree node is incremented for each subtree found at index s_i by the tracing procedure

Since the weights are incremented for each of a subtree's nodes, the weights of child nodes will always be greater or equal than the weights of their parents. To account for this additive effect, we calculate a node's sample count as the difference between its own weight and the weight of its parent.

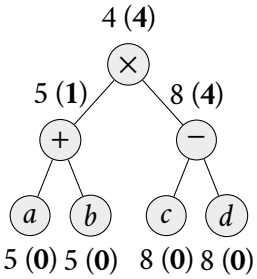


Figure 5.6: Subtree weights (above each node) and sample counts (bold font, in brackets)

Node weights are calculated during the run of the algorithm by computing the trace graphs of all individuals in the population at every generation.

5.2.4 Schema-based Analysis Methods

According to GP schema theories (see Section 4.3), schemas represent tree patterns of above-average fitness. Their frequency in the population varies depending on the statistical properties of the genotypic operators such as selection, crossover and mutation. Schemas are defined as tree patterns where nodes marked by the symbols {=, #} represent wildcards. The '=' symbol represents a single node which can be a function or a terminal[4]. The '#' symbol represents a node which can be matched by any other subtree, including leafs.

The occurrence and propagation of schemas through the interaction of genetic operators at the phenotypic and genotypic level is fundamentally connected with the evolvability of the population and the properties of the G→P map. In this section, we introduce a methodology for the analysis of common schemas in the population.

The schema analysis methodology can be broken down into the following essential components:

[4] '=' nodes will always match another node of the same type and arity – functions with functions and terminals with terminals

1) Pattern matching for unordered trees with wildcards
2) Schema template generation from the existing population

Tree Pattern Matching

Trees are fundamental data structures used for the representation of information in a structured, organised way. With the advent of web-based technologies and the increasing popularity of tree-structured document formats, tree pattern matching algorithms have become a necessity for fast information search and retrieval. In particular, the field of XML query matching has produced efficient matching algorithms for wildcard unordered tree patterns.

For our schema-based analysis, the algorithm by (Götz et al., 2009) was implemented in HeuristicLab. This algorithm performs a bottom-up matching of data trees against a wildcard query pattern and decides if a data tree D matches query tree Q. The answer is *yes* if a non-injective mapping exists between Q and D, such that each parent-child pair in Q has a matching ancestor-descendant pair in D. Time complexity is $O(|D| \cdot |Q| \cdot depth(Q))$ using a stack of depth bounded by $O(depth(D) \cdot branch(D))$.

Since schemas need to be matched top-down as well as bottom-up, additional restrictions were added to the algorithm implementation so that two nodes are matching only if:

1) They are on the same level in the tree.
2) Their parent nodes and child nodes are matching as well.

Note that in this particular use case, the algorithm will not be exact in all cases. The added restrictions cannot completely eliminate cases when the mapping between Q and D remains non-injective. This can happen in rare situations especially when Q is small and is included in D, or when D contains identical repeating patterns within its structure. However the above criteria ensure a low probability of mismatches, which only occur when the compared trees are structurally almost the same.

Schema Generation

Schema frequencies vary in the population according to the statistical properties of the genetic operators. In particular, "at the level of the microscopic degrees of freedom, the strings, we established that the action of crossover by its very nature introduces the notion of a schema" (Stephens and Waelbroeck, 1999).

We start our search for common patterns in the population by looking at the distribution of genetic material from one generation to the next in the genealogy graph. Individuals that were favored by selection will have produced more offspring via crossover or mutation. Thus, a logical approach would be start from the changes produced in the tree structure of the root parent. A heuristic approach is used to generate possible schemas:

1) Group individuals based on their common root parent

2) Identify all genetic fragments and their respective positions in the root parent

3) For each fragment f with preorder index f_i in the root parent, replace the node at position f_i with a wildcard. The replacement is done according to a minimum schema length requirement that can be defined as a parameter for the procedure. Positions f_i that would cause the schema length to fall under the requirement are skipped.

Algorithm 7 provides a detailed pseudocode description of the schema generation algorithm. The number of generated schemas and their structure depends on the minimum schema length limit. An important implementation detail is given by the fact that the cutpoint preorder indices are sorted in descending order so that subtree replacement does not invalidate remaining indices. Additionally, replacing wildcards bottom-up tends to produce more specific schemas and not very general ones.

```
1  Method GenerateSchemas(genealogy graph, minimum schema length)
2      schemas ← new list;                    // list holding the generated schemas
       // use genealogy information to group offspring with common parents
3      group all children of the current generation based on their common root parent;
4      foreach root parent p do
5          if length(p) < minimum schema length then
6              continue;
7          schema ← copy of p;
8          replaced ← false;
9          indexes ← preorder indices of the crossover cutpoints in all children;
10         sort(indexes);   // sort indices by cutpoint level in descending order
11         foreach index i from indexes do
12             subtree ← the subtree at position i in schema;
13             if length(schema) - length(subtree) + 1 < minimum schema length then
14                 continue;
15             replacement ← new wildcard node;                  // either = or #
16             ReplaceSubtree(subtree, replacement);   // replace the subtree with
                the wildcard in the parent's structure
17             replaced ← true;
18         if replaced then if the schema contains at least one wildcard
19             add schema to schemas;
20     return schemas;
```

Algorithm 7: Schema Generation

Schema frequency analysis matching generated schemas against the population can reveal new detailed insight concerning the convergence behavior of genetic algorithms and

can provide a theoretical basis for new algorithmic improvements. We explore below the possibility of enhancing algorithm performance using a schema-based method of controlling population diversity and preventing premature convergence during the evolutionary run.

5.2.5 Schema-based GP Diversification Strategies

Diversity is considered an essential factor for GP performance. Many approaches for preserving diversity and "good genetic material" have already been tried in the literature and discussed in Section 4.2.2. However, existing strategies based on heuristics such as "avoid too similar parents when performing crossover", "avoid individuals with a common lineage", or other strategies at the level of the selection scheme are often too inflexible and cannot be effectively tuned for a wider range of fitness landscapes.

In order to be effective, genetic diversification strategies need to be implemented at the level of the genotypic operators themselves, exploiting not only inheritance information in the population but also the structural and semantic characteristics of the individuals.

In this section, we introduce a novel diversity-improving method which takes into account all the above aspects:

- It is *structural*, as it considers individuals belonging to common schemas
- It is *hereditary*, as it uses the genealogy graph for schema generation
- It is *semantic*, as it considers phenotypically-similar individuals
- It is *adaptive*, as differential mutation rates can be applied within locally converged clusters of individuals

The schema-based diversification strategy implemented using the methods introduced in Sections 5.2.1 and 5.2.4 can be readily applied to existing GP systems. The idea is to achieve a finer balance between the exploration and exploitation aspects of the search by detecting the moments when the algorithm begins to converge towards (local) optima points and injecting new diversity in the population via mutation. Convergence is detected with the help of schemas and phenotypic similarity: an additional mutation step is performed when a schema starts to dominate in the population (above a certain frequency threshold) and the individuals matching it are also semantically similar.

The diversification strategy can be tuned using a set of parameters that are described below:

- The *minimum schema length* parameter controls the size of the generated schemas. This parameter is useful to avoid situations where the replacement of cutpoints close to the root of the tree would lead to very generic schemas such as for example `(+ # #)` (in postfix notation).

- The *minimum schema frequency* parameter sets a threshold for the relative number of matching individuals. Intuitively, this requirement helps establish the conditions under which a schema can be said to dominate the population.
- The *minimum phenotypic similarity* parameter is used to set additional trigger conditions for the diversification. Setting the phenotypic similarity threshold to a high value means that diversification will be applied only when highly similar individuals belonging to the same schema are present in the population.
- The *replacement ratio* parameter determines what proportion of the individuals matching a schema are mutated during the diversification phase. This parameter can be set to a fixed value which remains the same during the algorithm run, or it can be dynamically adjusted as a function of schema frequency. The dynamic replacement ratio rule can be set from a number of standard functions shown in Fig. 5.7. Intuitively, the proportion of mutated individuals within a schema should go up with the schema frequency as schema frequencies should be negatively correlated with population diversity.
- The *random replacement* parameter specifies whether or not the individuals matching a schema should be selected for mutation at random or based on their quality. When quality is used as a selection criteria, the individuals with the lower quality are picked first for mutation.

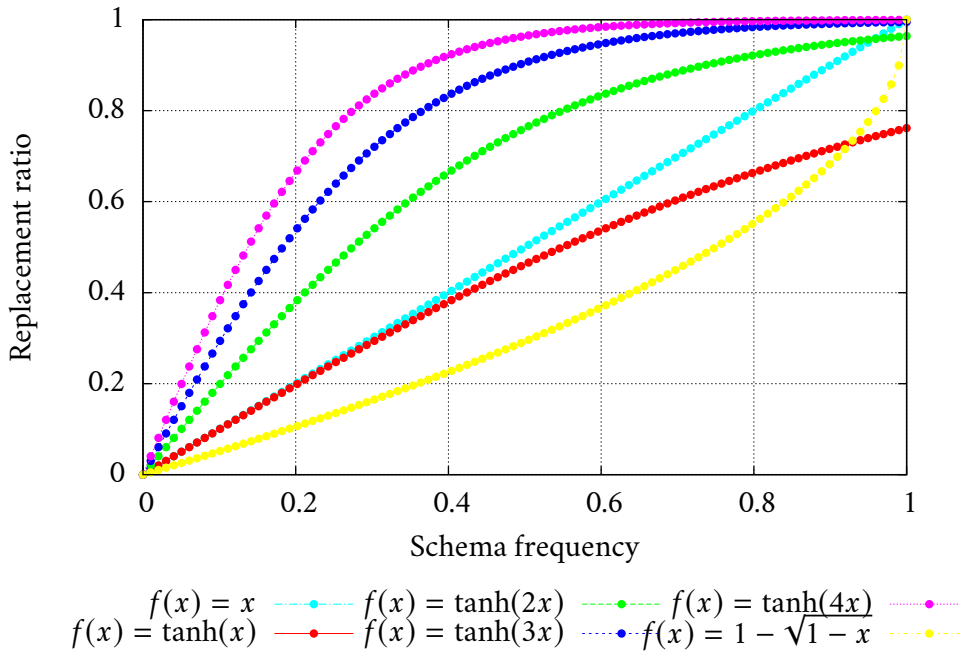


Figure 5.7: Dynamic replacement ratio rules

Schema diversification affects the dynamics of the search in more subtle ways:

- Diversification is applied at the end of the algorithm main loop (after selection, crossover, mutation), therefore it breaks the guarantee that all individuals in the next generation will be fitter than their parents (according to the offspring selection comparison factor).
- As diversification usually damages the quality of the individuals, a selection mechanism more tolerant with respect to these lower-quality individuals is needed (such as random selection or gender-specific). In order to allow the algorithm to converge even when a lower selection pressure is applied on the population, it makes sense to select not for fitness but for *adaptive fitness* that is, using an offspring selection mechanism as described in Section 4.2.2 in order to ensure that the fitness of the child individual is always better than at least one of the parents.

The new algorithm, called OSGP-S (Offspring Selection GP with Schemas) is compared with the other algorithms in Section 6.3.4.

6 Empirical Analysis

In this chapter experimental results obtained using the analysis methods introduced in Chapter 5 will be presented. According to the specific methodology, we distinguish two types of analysis methods:

1) Online methods for the analysis of evolutionary dynamics such as tree and fragment length, phenotypic and genotypic diversity, genetic operator improvement or parent distribution according under different selection mechanisms

2) Offline (aposteriori) methods for the analysis of gene and building block propagation in the population. These methods require the complete genealogy graph of the population for their calculations, and can be quite constly in terms of memory usage and computational effort. We also include in this category the tracing methodology which can be applied to the individuals in the population in order to investigate their genetic history.

All the algorithms were tested on a collection of benchmark and real-world problems as described in Table 6.1. The last column in the table refers to the best achievable quality considering noise on the training data.

Name	Function	Training	Test	Best R^2
Poly-10	$f(\mathbf{x}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$	250	250	1.00
Pagie-1	$f(x, y) = \frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$	676	1000	1.00
Friedman-II	$f(\mathbf{x}) = 10\sin(\pi x_1x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5 + noise$	500	5000	≈0.96
Tower	Unknown function of 25 variables	3136	1863	unknown

Table 6.1: Problem formulas, training and test partitions and best achievable quality.

We begin our analysis with the Standard GP (SGP) algorithm, followed by the Offspring Selection GP (OSGP) algorithm and the Offspring Selection GP with Schemas (OSGP-S) which employs the diversification strategy introduced in Section 5.2.5.

For each algorithm, similar configurations using a different selection scheme were compared with the purpose of highlighting the differences in their dynamics determined by the interplay of genetic operators.

6.1 Standard GP

6.1.1 Experiment Configuration

The standard GP experiments were performed using a population of 1000 individuals and a generation limit of 500 generations. The depth and length limits for the trees were set to 12 and 50 nodes, respectively. Subtree-swapping crossover was applied with a probability of 100% and mutation was applied with a probability of 25%. Every time mutation was applied, of the following manipulations was uniformly chosen to be performed on the tree individual:

- ◇ Parametric (point and multi-point) mutation
- ◇ Change node type mutation
- ◇ Replace subtree mutation
- ◇ Remove subtree mutation

Finally, the function set used by the algorithm was composed of the arithmetic functions $\{+, -, \times, \div\}$. Contribution ratios, phenotype and genotype similarities were sampled every 10 generations, while all other measurements were performed every generation.

For the standard GP, two algorithmic configurations using proportional and tournament selection with a group size of 4 individuals were tested. To avoid sampling artifacts, the results presented in the following sections were averaged over 50 algorithmic runs for each configuration. In each figure, the black curve represents the average value while the gray curves represent the individual values for each of the 50 runs.

An overview of the best and average solution qualities achieved by each algorithm configuration is provided in Tables 6.2 and 6.3.

6.1.2 Best and Average Solution Quality

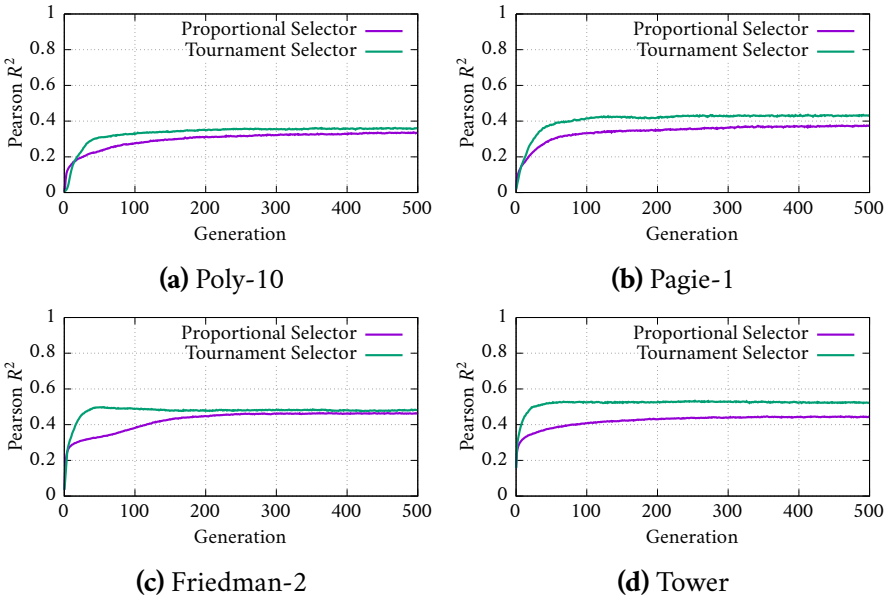
Concerning the best solution quality, Table 6.2 shows that SGP with tournament selection outperforms SGP with proportional selection on every test problem, although on average, as shown in Table 6.3, SGP with proportional selection produced solutions with better generalization capabilities for the Poly-10 and Pagie-1 problems.

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	0.8970	0.9866	0.7878	0.8738
	0.8993	0.9864	0.8106	0.8777
Tournament	0.9782	0.9999	0.9057	0.8898
	0.9813	0.9999	0.8990	0.8902

Table 6.2: SGP Best Solution Qualities

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	0.7719 ± 0.1471	0.9077 ± 0.5200	0.7084 ± 0.0229	0.8341 ± 0.0184
	0.7423 ± 0.1855	0.7905 ± 0.2657	0.7202 ± 0.0632	0.8388 ± 0.0193
Tournament	0.7620 ± 0.1581	0.9142 ± 0.0591	0.8150 ± 0.0481	0.8610 ± 0.0153
	0.6878 ± 0.2461	0.7201 ± 0.3090	0.7791 ± 0.1197	0.8621 ± 0.0150

Table 6.3: SGP Average Solution Qualities



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

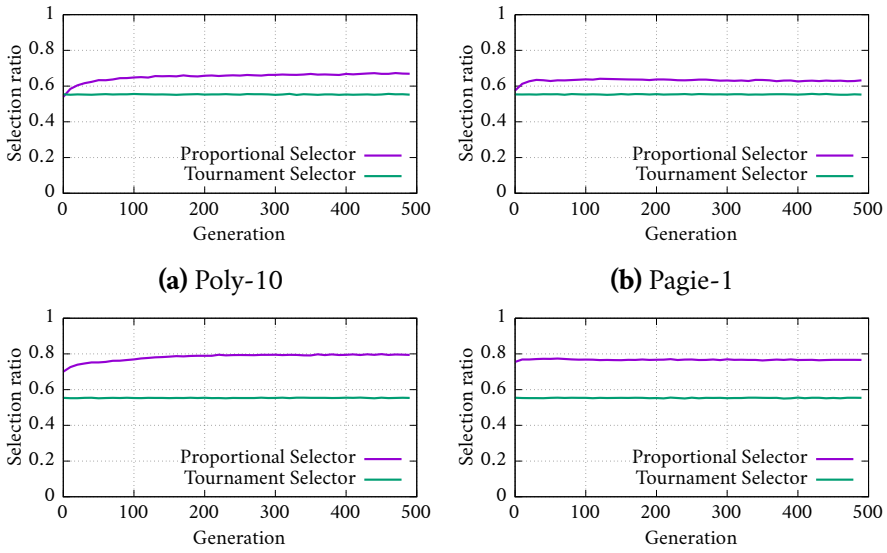
Figure 6.1: SGP Average Population Quality

Fig. 6.1 shows that in all cases, tournament selection leads to a higher average quality in the population and a shorter duration of the exploratory phase of the algorithm, marked by a steeper increase in average quality. Looking at operator improvement charts, we notice that a higher average quality is correlated with a higher negative average improvement for both crossover and mutation. We can conclude that in the absence of an offspring selection mechanism, both crossover and mutation will, on average, worsen the fitness of the individuals they are applied on.

6.1.3 Selection Ratio and Parent Distribution

Figs. 6.2 to 6.4 reveal significant differences between the two selection mechanisms in the number of unique parents selected for recombination at each step and their distribution curves.

Unsurprisingly, the stochastic nature of the tournament selector makes its behavior



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.2: SGP Average Selection Ratio

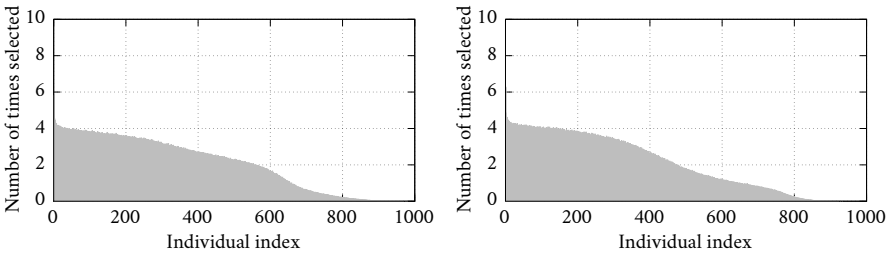
completely independent from the shape and characteristics of the fitness landscape, leading to identical selection ratios and parent distributions for all test problems. On the other hand, proportional selection is more sensitive to the distribution of fitness values in the population and is therefore influenced by the specific problem instance.

6.1.4 Average Tree and Fragment Length

The evolution of GP tree sizes shown in Fig. 6.5 is consistent with the crossover bias theory proposed by (Poli et al., 2007b): crossover bias pushes tree sizes to a Lagrange distribution of the second kind, where smaller trees are much more likely to be sampled by the crossover operator. As smaller programs tend to be less fit, the selective advantage of larger programs has the effect of increasing the average population size. This effect is more pronounced with high selection pressure, explaining the observed differences between proportional and tournament selection.

While surprising at first, the larger mutation fragment size can be explained by the presence of the multi-point mutation operator which affects all the leaf nodes in the tree, in which case the mutation fragment is considered to be the subtree rooted into the lowest common ancestor of all the mutated nodes. Since random mutation samples trees uniformly, the difference in average mutation fragment size reflects the difference in average tree size between the two selectors.

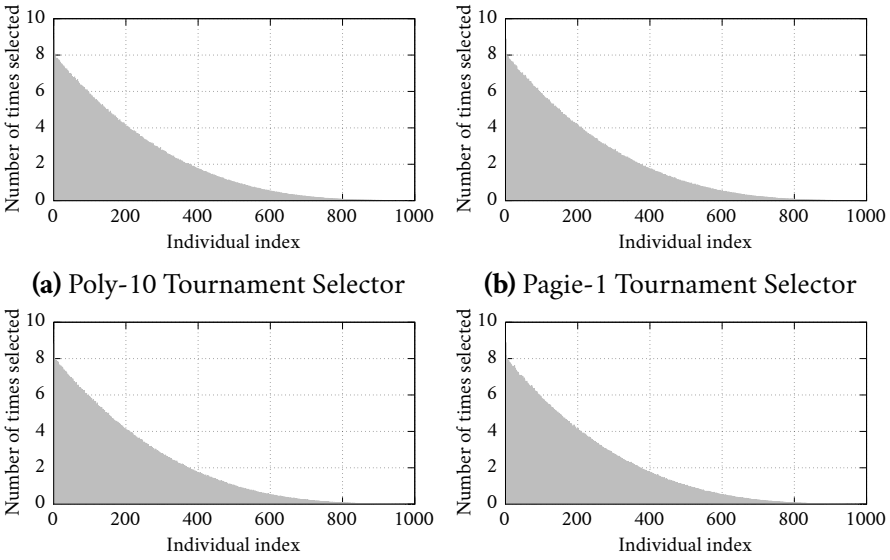
Crossover bias also explains the decrease of average tree size after the first generation observable especially in Figs. 6.5b to 6.5d, as the crossover operator transforms the tree



(a) Poly-10 Proportional Selector **(b)** Pagie-1 Proportional Selector

(c) Friedman-2 Proportional Selector **(d)** Tower Proportional Selector

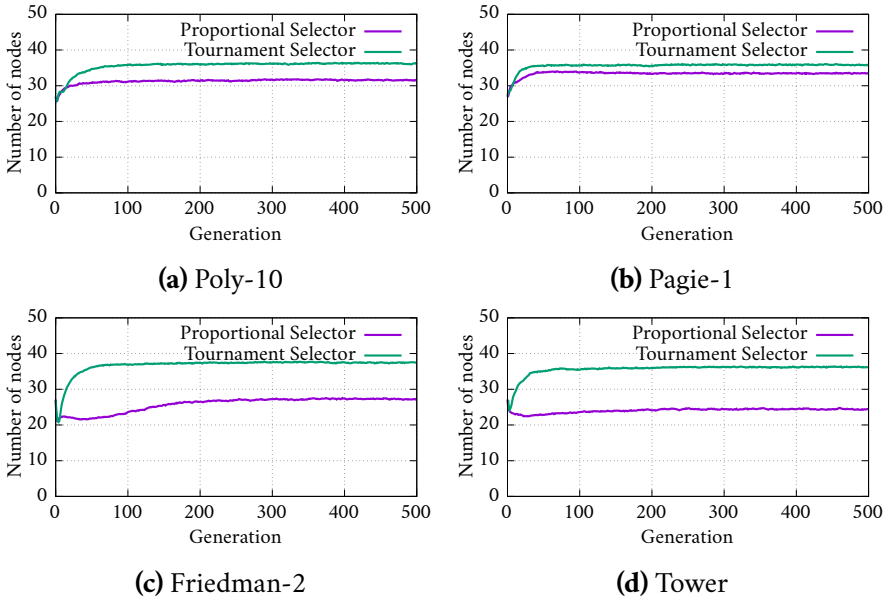
Figure 6.3: SGP Proportional Selector Average Parent Distribution



(a) Poly-10 Tournament Selector **(b)** Pagie-1 Tournament Selector

(c) Friedman-2 Tournament Selector **(d)** Tower Tournament Selector

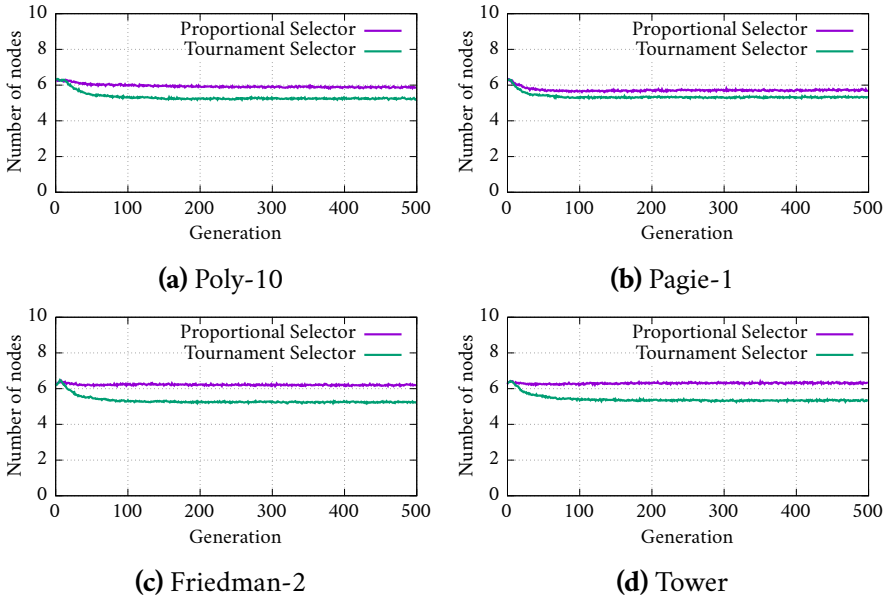
Figure 6.4: SGP Tournament Selector Average Parent Distribution



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

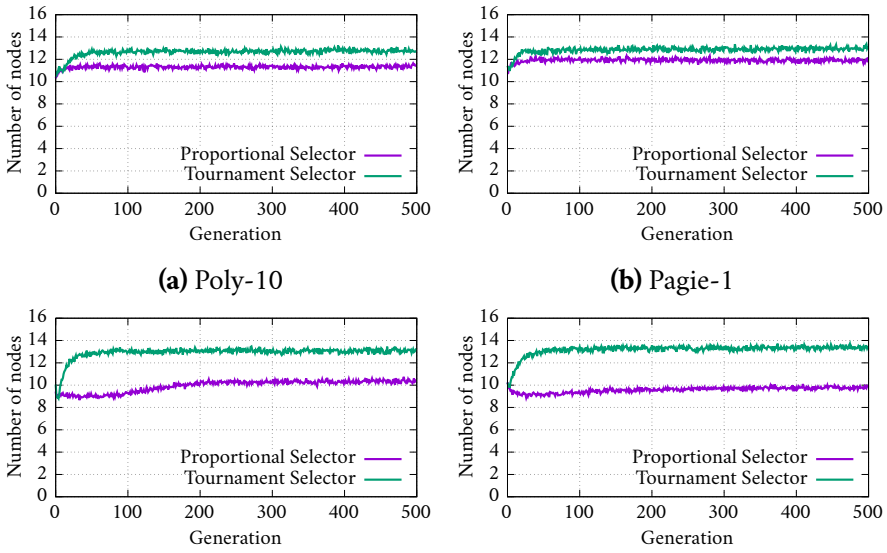
Figure 6.5: SGP Average Tree Length



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.6: SGP Average Crossover Fragment Length



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.7: SGP Average Mutation Fragment Length

size distribution generated by the PTC2 tree creator. The same bias is responsible for the decrease in average crossover fragment size after the first generation.

Fig. 6.6 suggests a negative correlation between average tree length (indirectly determined by the selection pressure) and average crossover fragment size. To explain this phenomenon, we performed a smaller experiment where we calculated the tree size distribution in the population (Fig. 6.8a), the size distribution of crossover fragments (Fig. 6.8b) and the size distribution of the subtrees that they replace in the root parents (Fig. 6.8c). These results were averaged over 50 algorithmic runs and 100 generations for each run.

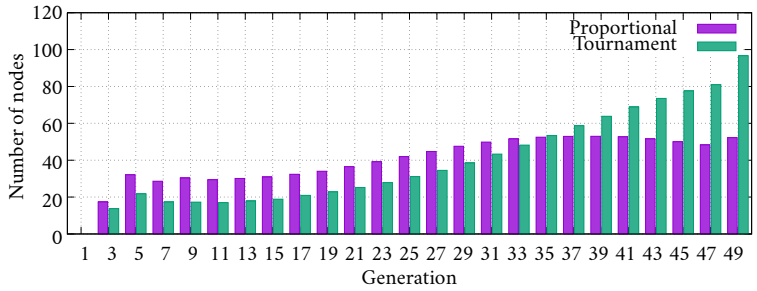
Fig. 6.8a shows that the higher average tree length determined by the tournament selection scheme has an underlying size distribution skewed towards larger trees. For example, we see that the higher average tree size under tournament selection is mostly due to the increased frequency of trees with length between 37 and 49 nodes[1]. Figs. 6.8b and 6.8c show that on average, for both selection methods, larger subtrees are replaced with smaller ones. This confirms that the increase of average tree size in the population is entirely due to selection.

Despite the difference in the distribution of tree sizes, we notice that the replaced subtree sizes are distributed the same (ie., crossover's choice of cutpoint in the root parent produces on average the same distribution regardless of actual tree size distribution and average length). The observed behavior suggests an additional crossover bias imposed by the

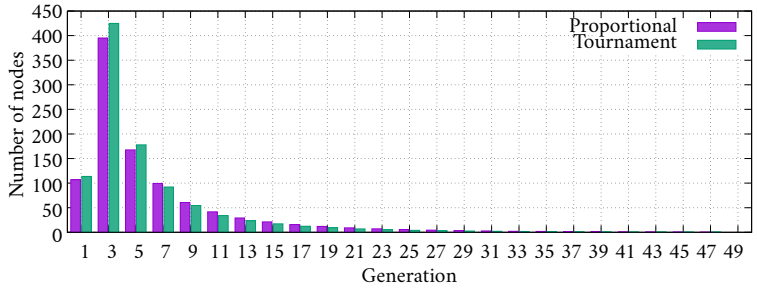
[1] Note that since the used primitive set has an arity of 2, all trees and subtrees are binary trees and have an odd number of nodes.

maximum tree size restriction towards smaller fragments sampled from the non-root parent, which explains the crossover fragment size difference observed between the two selection schemes.

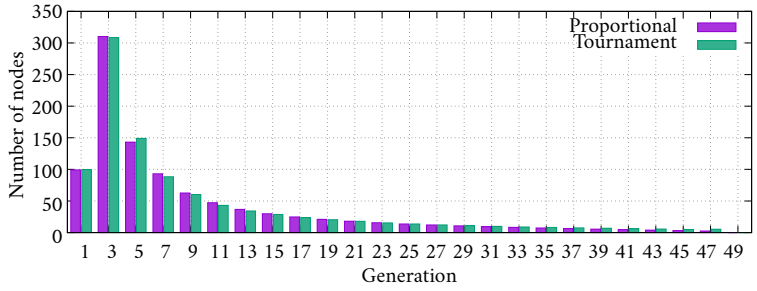
This behaviour is typical of the size-restricted crossover operator used in HeuristicLab, where the choice of the second crossover point in the non-root parent is subject to size restrictions so that the offspring does not exceed the maximum size limit.



(a) Size distribution of trees in the population

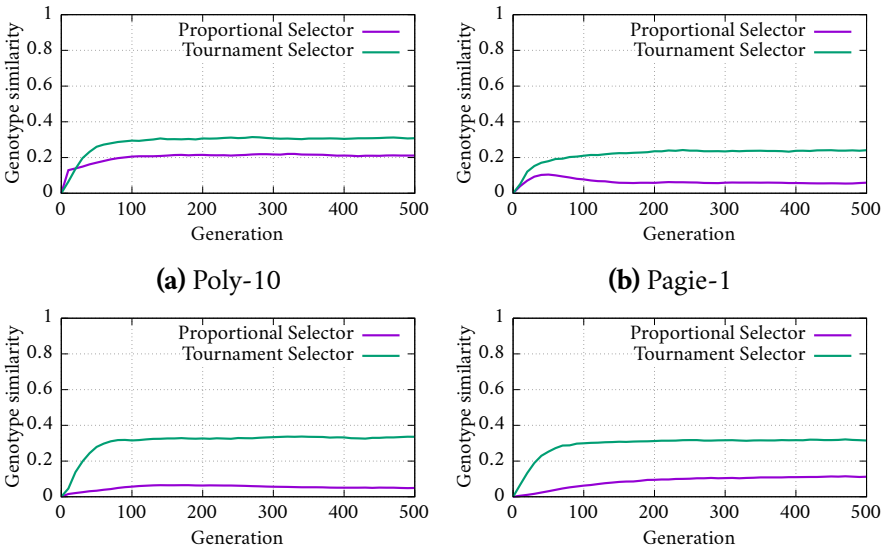


(b) Average crossover fragment size



(c) Average crossover replaced subtree size

Figure 6.8: Size distribution of individuals in the population and subtrees replaced by the crossover operator



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.9: SGP Average Genotype Similarity

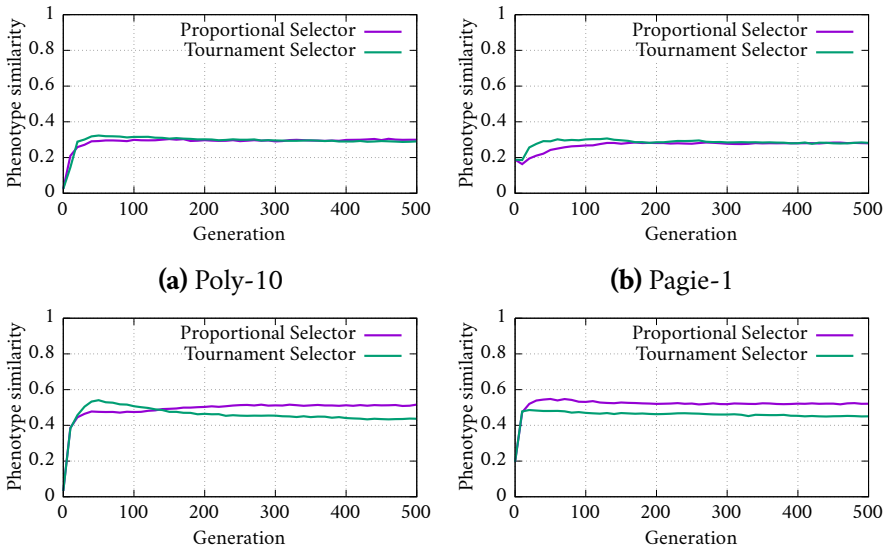
6.1.5 Population Diversity

Figs. 6.3 and 6.4 have already outlined important differences between the behavior of the two selection operators in terms of how children are produced. We saw that overall, tournament selection produces the same parent distribution independently of the concrete problem instance and the relative fitness differences between the individuals in the population. Obviously, the strong bias towards fitter individuals – as determined by the tournament group size of 4 – leads to increased genotypic similarity between the child individuals as shown in Fig. 6.9.

With regard to the phenotype similarity shown in Fig. 6.10, we notice similar similarity levels for both selection schemes on the Poly-10 and Pagie-1 problems, and higher phenotypic similarity under proportional selection on the Friedman-2 and Tower problems, despite the fact that average population quality was higher on all problems when using the tournament selection scheme. This shows that actual fitness bears little influence on the similarity between the semantics of individuals (observed at phenotype level).

6.1.6 Genetic Operator Effectiveness

Figs. 6.11 to 6.14 show the average crossover and mutation improvement calculated using Eq. (5.5). We notice that the average parent-child fitness difference is always negative which means that on average, the crossover and mutation operators are actually worsening the fitness of the individuals they act upon. This negative improvement is also influenced by



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.10: SGP Average Phenotype Similarity

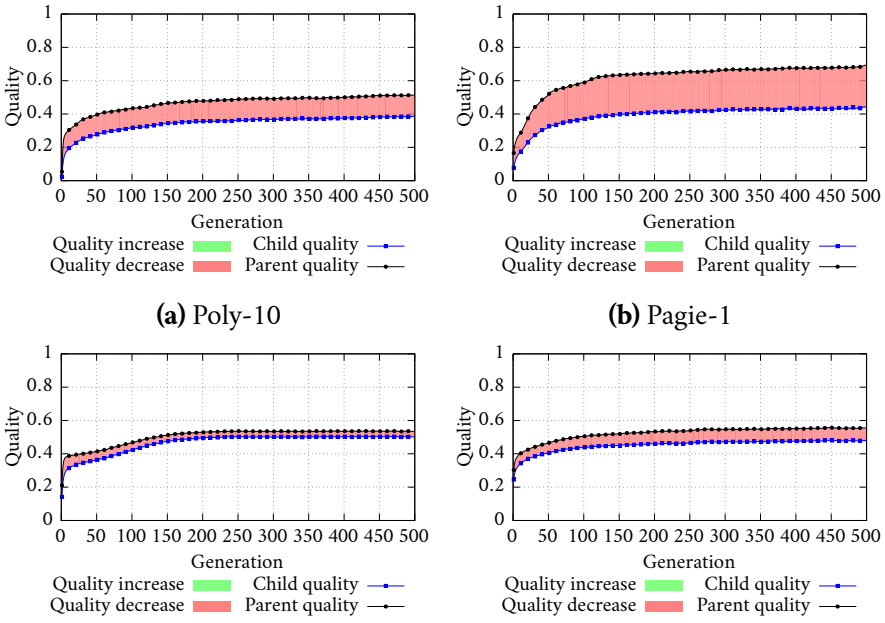
the average population quality (Fig. 6.1), since higher fitness leads to a decreased probability of adaptive change and to increased chances of deleterious genotype changes.

For the reasons detailed above, we observe a more pronounced loss of fitness in the case of the tournament selector. We can conclude that both crossover and mutation are not effective in producing adaptive change (ie., that would lead to increased fitness). Considering a typical scenario where adaptive changes are produced in a small fraction of offspring while the remaining ones will be less fit than their parents, it is easy to imagine how selection – favoring the more fit phenotypes – can lead to pronounced loss of diversity in the population. In other words, if genotypic operators are not effective, diversity loss becomes an unfortunate side effect of maintaining an overall fitness improvement over the generations (driven by selection).

6.1.7 Schema Frequency Analysis

In order to investigate schema propagation within the GP population, we selected for analysis the best run from each algorithmic configuration. The best qualities obtained by each algorithmic configuration are displayed in Table 6.2. Schemas with a minimum length of 10 nodes were generated and matched against the population.

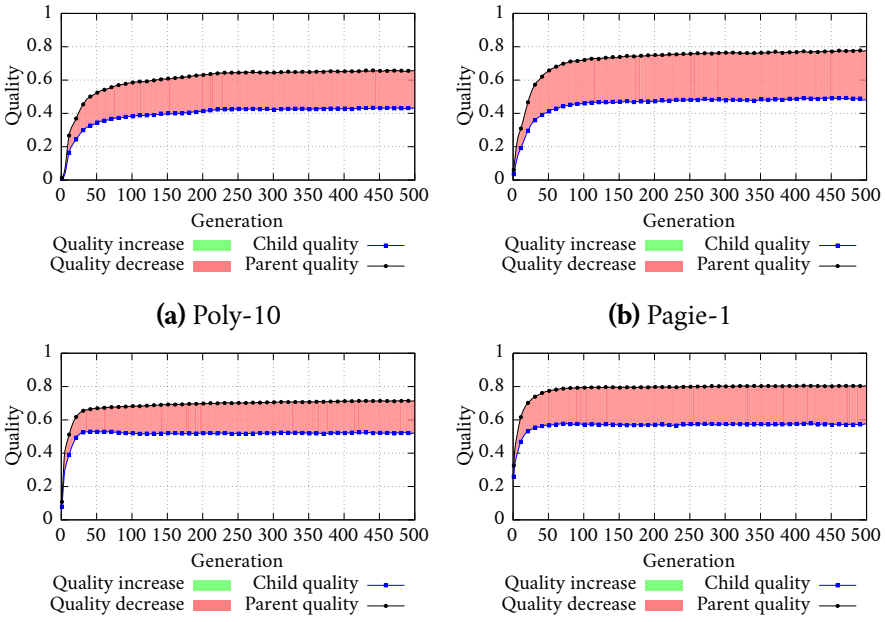
For each algorithmic run, the relative frequency of the most common schema, its average quality compared to the average population quality and the average genotype and phenotype similarity of its matching individuals were calculated. Due to the length of the runs, the relative schema frequencies, average quality and the similarities of their matching



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

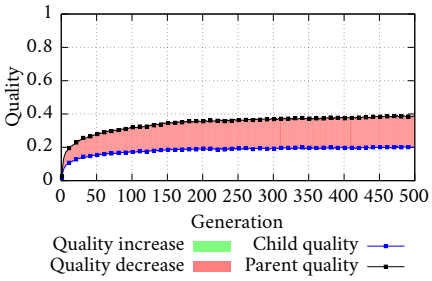
Figure 6.11: SGP Proportional Selector Average Crossover Improvement



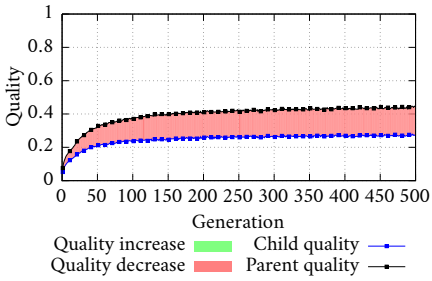
(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

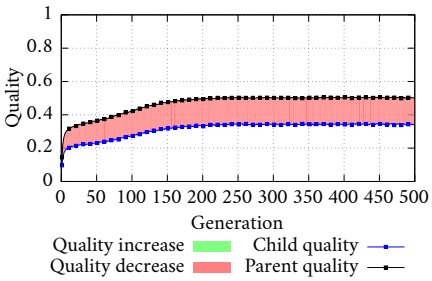
Figure 6.12: SGP Tournament Selector Average Crossover Improvement



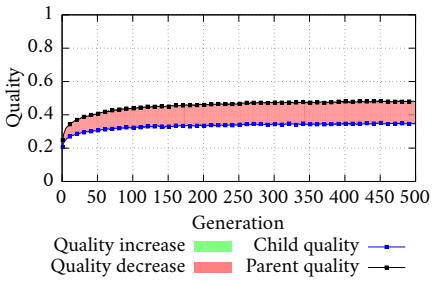
(a) Poly-10 Proportional Selector



(b) Pagie-1 Proportional Selector

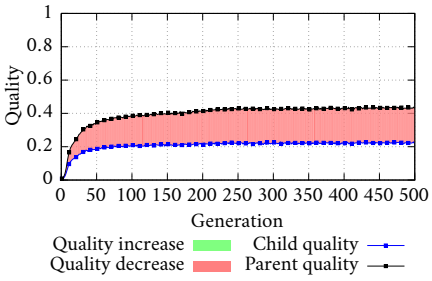


(c) Friedman-2 Proportional Selector

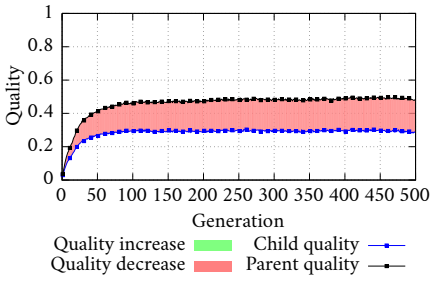


(d) Tower Proportional Selector

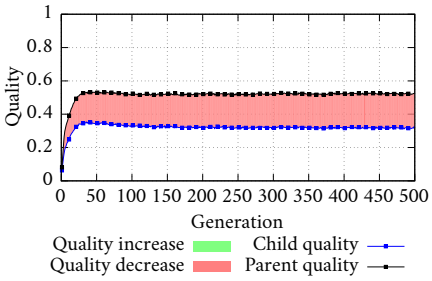
Figure 6.13: SGP Proportional Selector Average Mutation Improvement



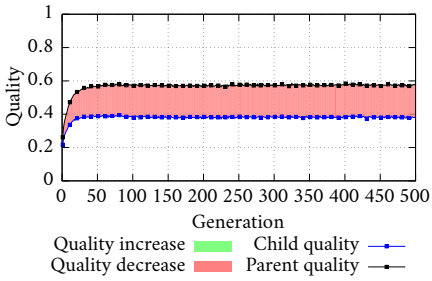
(a) Poly-10



(b) Pagie-1



(c) Friedman-2



(d) Tower

Figure 6.14: SGP Tournament Selector Average Mutation Improvement

individuals (genotypic and phenotypic) were calculated every 10 generations.

Due to their size, the tables showing the concrete schema instances identified in each problem instance are omitted in this section and shown in Section 7.3. They can be correlated with the frequency and similarity measurements shown below.

The results aggregated in Figs. 6.15 to 6.18 reveal a number of interesting aspects regarding the differences between the dynamics of the two selection schemes:

1) Schemas generated using the method described in Section 5.2.4 are of above-average quality. For both selection schemes, the average schema quality curve exceeds the average population quality, as seen in Figs. 6.15 and 6.16.

2) The evolution of the relative schema frequency (calculated as the ratio of matched to total number of individuals) shows clear differences between the two selection schemes: for all the tested problems, the higher selection pressure applied by the tournament selector on the population leads to a higher relative frequency of the most common schema.

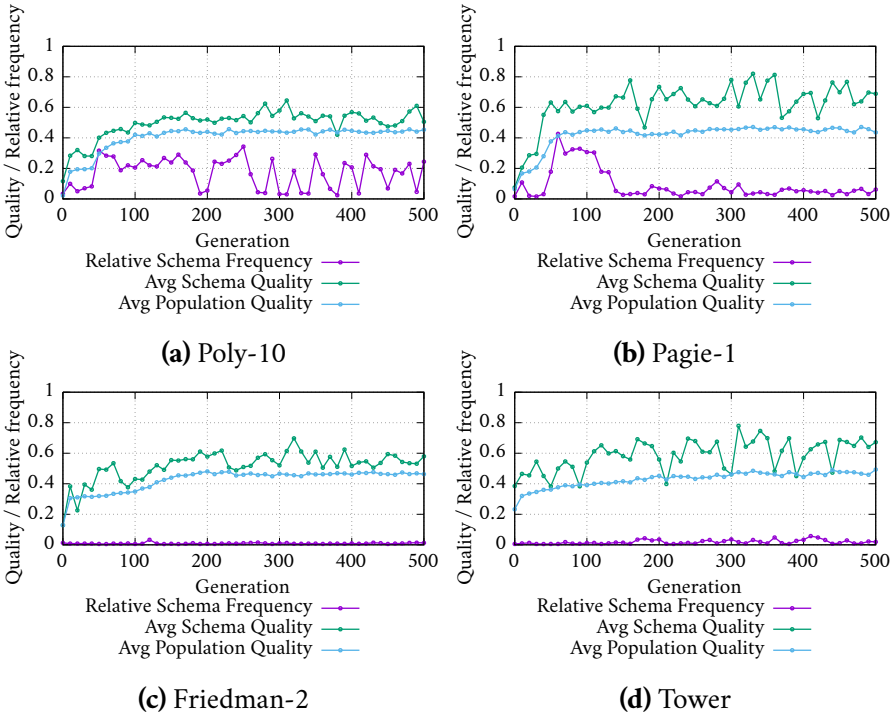
3) The evolution of schema frequency can be correlated with the obtained population quality. For example, the more significant quality differences in favor of the tournament selector on the Friedman-2 and Tower problems can be linked with superior convergence measured as increased relative schema frequency. We notice that in the case of proportional selection no schema convergence could be observed on the Friedman-2 and Tower problems.

4) With respect to population diversity, we compared the results in figures Figs. 6.17 and 6.18 with those in Figs. 6.9 and 6.10. We notice that average similarity within schemas is always considerably higher than the average genotype and phenotype similarities over the whole population. Correlated with schema frequency information, this offers more detailed insight on the rate of diversity loss in the population and on the suitability of the various selection schemes on different problem instances.

Overall, schema analysis represents a powerful instrument for investigating aspects such as rate of convergence and diversity loss under different selection mechanisms.

6.1.8 Analysis of Solution Contribution Ratio

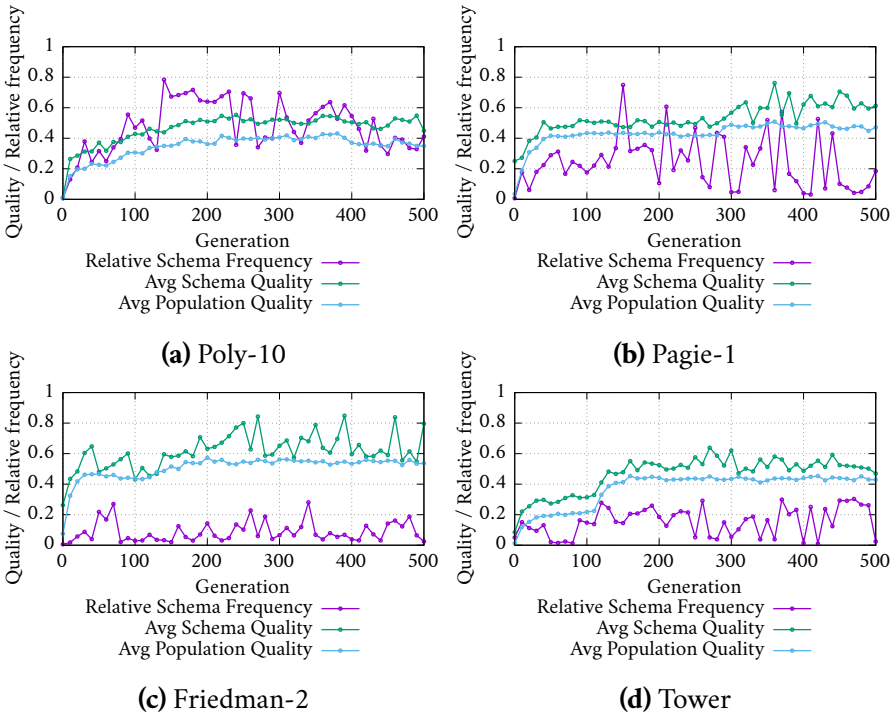
Contribution ratio, defined as the proportion of an individual's ancestors which contributed with genes to its structure, represents a new measure of the evolutionary effort expended by the algorithm for finding solutions. This aspect of the search concerning the way in which better genotypes are assembled is intimately related to the G→P map and the evolution of population diversity. The expression for the contribution ratio given in Eq. (5.9) requires the computation of individual trace graphs. Thus, the subtree tracing algorithm described in Section 5.2.3 was applied each generation on the whole popula-



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

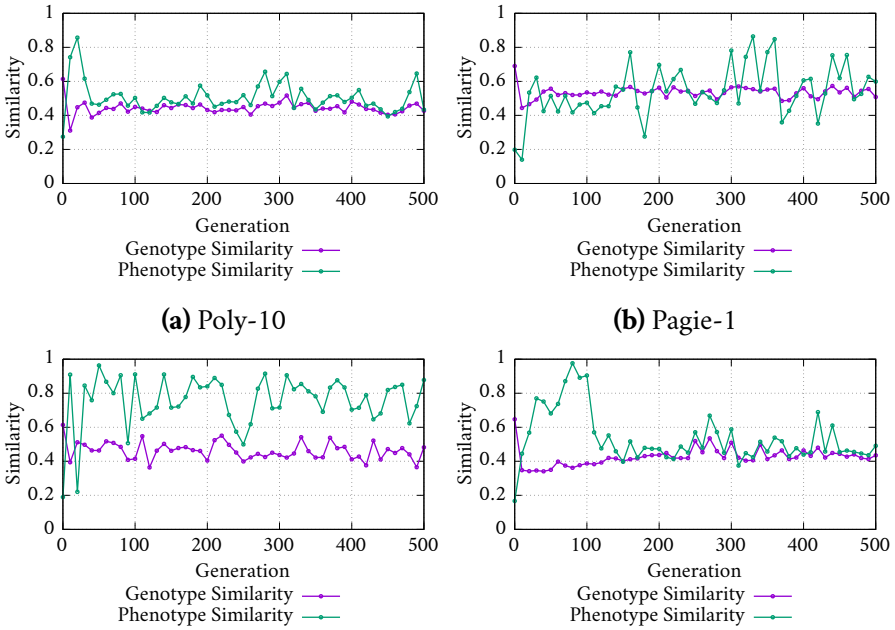
Figure 6.15: SGP Proportional Selector Schema Frequencies and Quality



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

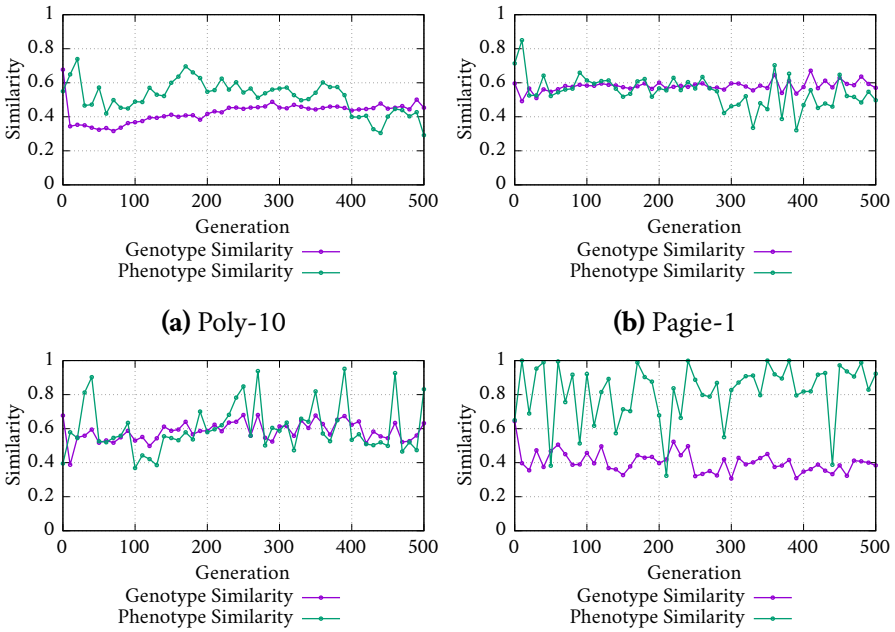
Figure 6.16: SGP Tournament Selector Schema Frequencies and Quality



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.17: SGP Proportional Selector Schema Similarities



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.18: SGP Tournament Selector Schema Similarities

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	19.13%	29.8%	27.58%	32.3%
Tournament	12.22%	8%	3.9%	6.9%

Table 6.4: SGP Best Solution Contribution Ratio

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	53831 281324	74553 250484	77750 281921	122302 379558
Tournament	21918 179427	15521 192924	6838 173279	13589 196297

Table 6.5: SGP Best Solution Trace v. Ancestry Size

tion, simultaneously constructing the trace graph of each individual and incrementing the weights of the traced subtrees.

In what follows, we show for each problem and algorithm configuration the contribution ratio for the best solution and the most sampled subtrees from the whole genealogy of individuals and subtrees.

The results from Tables 6.4 and 6.5 show that a relatively small percentage of individuals from the best solution ancestry had an actual contribution to its structure (in terms of inherited genes). Tournament selection in particular leads to a very low contribution ratio as the high selection pressure leads to fewer unique individuals involved in the offspring creation process.

The trace graph and ancestry sizes seen in Table 6.5 can vary depending on several factors such as the number of common ancestors within an individuals lineage, the actual number of times when the individual was being passed as an elite from one generation to the next, or the generation number when the best solution quality was achieved.

Results in Figs. 6.19 and 6.20 reveal the relationship between selection acting at phenotype level and the variation-producing operators acting at the genotype level. The graphs were obtained in two steps:

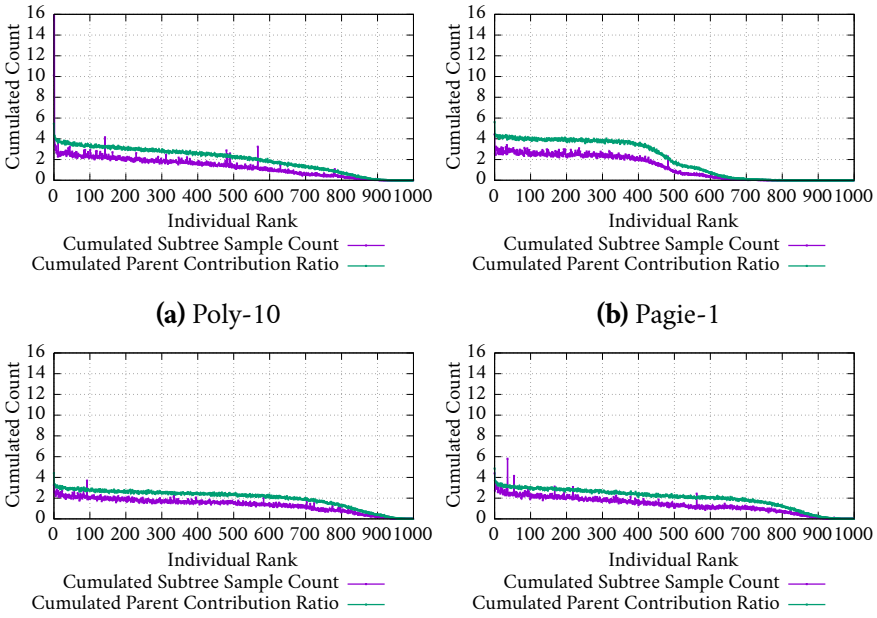
1) For each generation, rank the population according to fitness, so that rank 0 corresponds to the best individual, rank 1 to the second best, and so on.

2) Sum the average node weight and the number of produced offspring of all individuals of the same rank.

The correlations between the curves, displayed in Table 6.6 validate, on the one hand, the correctness of the tracing methodology and on the other hand, bring additional details about the dynamics of the search. For example, the spikes in the cumulated subtree sample count curve show that, at some point during the evolution one of the subtrees was sampled for a

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	0.6517	0.9837	0.9430	0.9080
Tournament	0.9150	0.9295	0.8084	0.9207

Table 6.6: Correlation between the cumulated subtree sample count and the cumulated parent contribution ratio curves from Figs. 6.19 and 6.20



(a) Poly-10 **(b)** Pagie-1

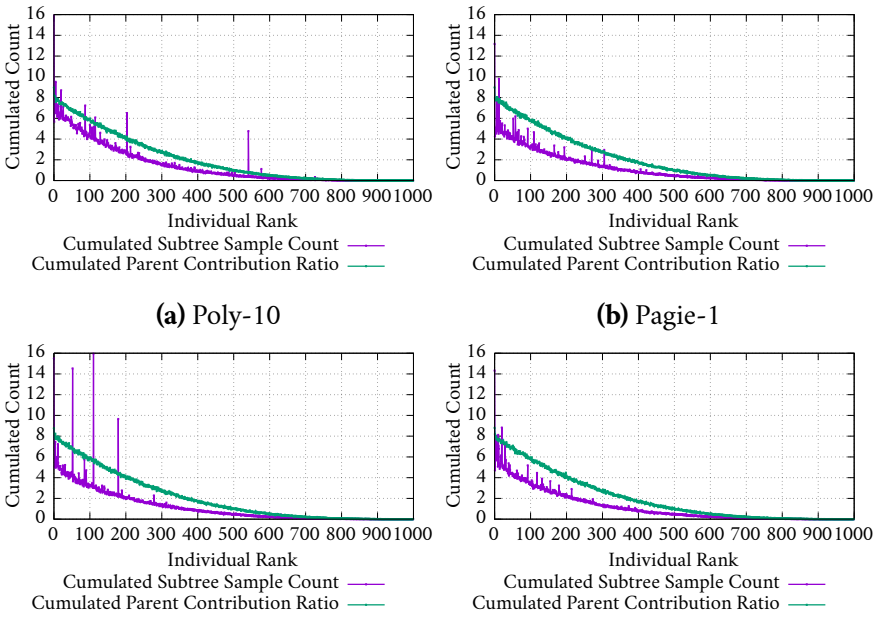
(c) Friedman-2 **(d)** Tower

Figure 6.19: SGP Proportional Selector Cumulated Subtree Sample Count and Parent Contribution Ratio

large number of times propagating itself in future individuals and having a big influence on the search outcome. In this way, numerous large spikes are connected, through the action of selection and crossover, with the low contribution ratios observed in Table 6.4.

The analysis of the solution contribution ratio using trace graph can also reveal details regarding the significant solution building blocks which could be identified by the algorithm during the evolutionary search. These building blocks identified as the most sampled subtrees in the population can for example offer insight to the practitioner about the structure and characteristics of the fitness landscape for unknown instances of symbolic regression problems.

To illustrate the identification of relevant building blocks, we show for each problem the 10 most sampled subtrees together with the total number of times they were sampled by the genotypic operators. These measurements were computed for the best runs corresponding



(a) Poly-10

(b) Pagie-1

(c) Friedman-2

(d) Tower

Figure 6.20: SGP Tournament Selector Cumulated Subtree Sample Count and Parent Contribution Ratio

to each problem and algorithmic configuration using the tracing and subtree sample count methodology described in Section 5.2.3. The best solution qualities extracted from the best runs in each algorithm and problem configuration are shown in Table 6.2.

It is clear that the most sampled subtrees will more accurately represent the actual problem building blocks when the solution found by the algorithm gets as close as possible to the actual formula. Therefore, since the best solutions obtained by SGP with tournament selection are clearly superior to those obtained by SGP with proportional selector, we expect the former to have produced the most relevant subtree sample count measurements.

Subtree	Count
(* x_2 x_1)	130902
(* x_5 x_6)	112361
(* x_4 x_3)	59896
(+ (* x_5 x_6) (* x_2 x_1))	13592
(+ (* x_2 x_1) (* x_5 x_6))	9108
(+ (* x_4 x_3) (* x_2 x_1))	6287
(+ (* x_2 x_1) (* x_2 x_1))	5883
(+ (* x_2 x_1) (* x_4 x_3))	5285
(+ (* x_5 x_6) (* x_5 x_6))	5117
(+ (* x_5 x_6) (* x_4 x_3))	4990

$f(\mathbf{x}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$

(a) Poly-10

Subtree	Count
(* x y)	59352
(* y y)	50828
(* y x)	36468
(* x x)	30.496
(* x (* y y))	21168
(* y (* x y))	20691
(* x (* x y))	14866
(* (* y (* x y)) x)	13363
(* (* x (* y y)) x)	12979
(* y (* x x))	11826

$f(x, y) = \frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$

(b) Pagie-1

Subtree	Count
(+ x_4 x_2)	33800
(+ x_1 x_2)	13016
(+ x_1 x_4)	12317
(+ x_2 x_4)	9827
(+ x_4 x_4)	9076
(+ x_5 x_4)	7546
(+ x_1 x_5)	7093
(+ x_4 x_5)	6918
(+ x_4 x_1)	6728
(* x_1 x_2)	4066

$f(\mathbf{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5 + noise$

(c) Friedman-2

Subtree	Count
(* x_{23} x_6)	64568
(/ x_6 x_1)	18208
(/ (* x_{23} x_6) x_1)	7824
(* x_6 x_{23})	5032
(* x_6 x_6)	2361
(* c (* x_{23} x_6))	1940
(* x_{23} (/ x_6 x_1))	1575
(* (* x_{23} x_6) x_6)	1449
(+ (* x_{23} x_6) (* x_{23} x_6))	1432
(* x_{23} (* x_{23} x_6))	1292

Unknown function of 25 variables

(d) Tower

Table 6.7: SGP Proportional Selector Most Sampled Subtrees

Subtree	Count
(* x_9 x_7)	84237
(* x_4 x_3)	51600
(* x_3 x_{10})	47703
(* x_8 x_3)	24688
(* c (* x_9 x_7))	23141
(+ x_5 (* x_3 x_{10}))	15727
(+ x_1 x_7)	12191
(+ x_5 (* x_4 x_3))	9028
(+ x_5 (* x_8 x_3))	8933
(+ x_5 (* x_9 x_7))	8921

$f(\mathbf{x}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$

(a) Poly-10

Subtree	Count
(* y y)	99933
(* x x)	96250
(- (* y y) c)	24346
(/ c (* x x))	20203
(* (* x x) x)	15167
(/ (* c c) (* y y))	15012
(* (* (* x x) x) x)	9973
(- (* (* (* x x) x) x) c)	7625
(- (* x x) c)	7078
(/ (* c c) (- (* y y) c))	6835

$f(x, y) = \frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$

(b) Pagie-1

Subtree	Count
(+ x_4 x_5)	88868
(* x_1 x_2)	88049
(* x_2 x_1)	64208
(* (* x_1 x_2) (* x_2 x_1))	18964
(+ (* x_1 x_2) (+ x_4 x_5))	17645
(- (* (* x_1 x_2) (* x_2 x_1)) c)	15616
(* x_3 x_3)	8865
(* (+ (* x_1 x_2) (+ x_4 x_5)) (- (* (* x_1 x_2) (* x_2 x_1)) c))	6561
(+ x_2 (+ x_4 x_5))	6353
(+ (* x_2 x_1) (+ x_4 x_5))	5463

$f(\mathbf{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5 + noise$

(c) Friedman-2

Subtree	Count
(+ x_{13} x_{15})	56169
(/ x_{23} (* c x_1))	55440
(* x_1 x_1)	54137
(* x_6 (/ x_{23} (* c x_1)))	37365
(/ (* x_1 x_1) x_{11})	31569
(* x_1 x_{10})	8564
(/ x_2 (* c x_{10}))	5885
(/ x_2 x_{23})	4825
(/ x_2 x_{24})	3728
(/ (* c x_1) x_{11})	3256

Unknown function of 25 variables

(d) Tower

Table 6.8: SGP Tournament Selector Most Sampled Subtrees

Tables 6.7 and 6.8 show the problem-specific elements of the formula the algorithm was able to identify:

- ◇ On the Poly-10 problem, the most sampled subtrees are the ones representing the two-variable products such as x_1x_2, x_3x_4 or x_5x_6, followed by sums of these products such as $x_1x_2 + x_3x_4$ and so on. This shows that the algorithm is at least able to find and assemble the basic building blocks of the formula. However, only SGP with tournament selector was further able to find the more difficult variable combinations such as x_3x_{10} or x_7x_9.
- ◇ On the Pagie-1 problem, the most sampled subtrees are those who encode products or powers of the x and y variables. In particular SGP with tournament selection was able to find the term x^4 among other combinations of factors.
- ◇ The target formula for the Friedman-2 problem contains a sinus term which cannot be modeled directly by the algorithm since no trigonometric functions were included in the primitive set. However, the algorithm correctly identified the term x_1x_2 (the argument of the sinus) as the most sampled subtree in the population, considering its two instances in Table 6.8c, where it is present both as x_1x_2 and x_2x_1. The second most sampled subtree, namely $x_4 + x_5$ accounts for the last two terms of the formula, also representing a genuine problem building block.
- ◇ We cannot say anything about the Tower formula since the objective function is unknown. However, the variables x_1, x_2, x_6, x_{10}, x_{11}, x_{13}, x_{15}, x_{23} and x_{24} were identified by the algorithm as being the most relevant. This illustrates the practical benefits of the approach: even for unknown problems, the subtree sample counts methodology can offer information regarding the most important solution elements.

In conclusion, the tracing methodology brings important benefits for the study of GP evolutionary dynamics by allowing us, one the one hand, to get more detailed insight into genetic operator behavior and the inheritance patterns they determine, and on the other hand to identify relevant problem building blocks via the subtree sample counts approach, that can potentially be translated into new knowledge about the problem by domain experts.

6.2 Offspring Selection GP

6.2.1 Experiment Configuration

The Offspring Selection GP algorithm (OSGP) was used with a population of 500 individuals. Strict offspring selection was used after recombination, meaning that only the offspring which outperformed both parents were accepted into the population. The tree length and depth limits were the same as for standard GP, namely 12 levels depth and 50

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	0.9999	0.9999	0.8841	0.8885
	0.9999	0.9999	0.8772	0.8917
Gender Specific	0.9999	0.9999	0.8967	0.8959
	0.9999	0.9999	0.8836	0.8975

Table 6.9: OSGP Best Solution Qualities

nodes maximum length. A crossover probability of 100% and a mutation probability of 25% were used along two different selection schemes: proportional and gender-specific. In gender specific selection, the first parent is chosen proportionally while the second parent is chosen at random. A maximum selection pressure of 100 was used as a termination criteria for the algorithm (see Section 4.2.2).

Due to the dynamic termination criteria and the algorithm's inherent stochasticity, OSGP runs using the same configuration will not finish in the same number of generations. For this reason, the results shown below are not aggregated into averages like in the case of the standard GP. An exception was made only for the genetic operator improvement charts where averages per generation of the remaining runs were calculated in order to be able to present the data in a compact manner.

6.2.2 Best and Average Solution Qualities

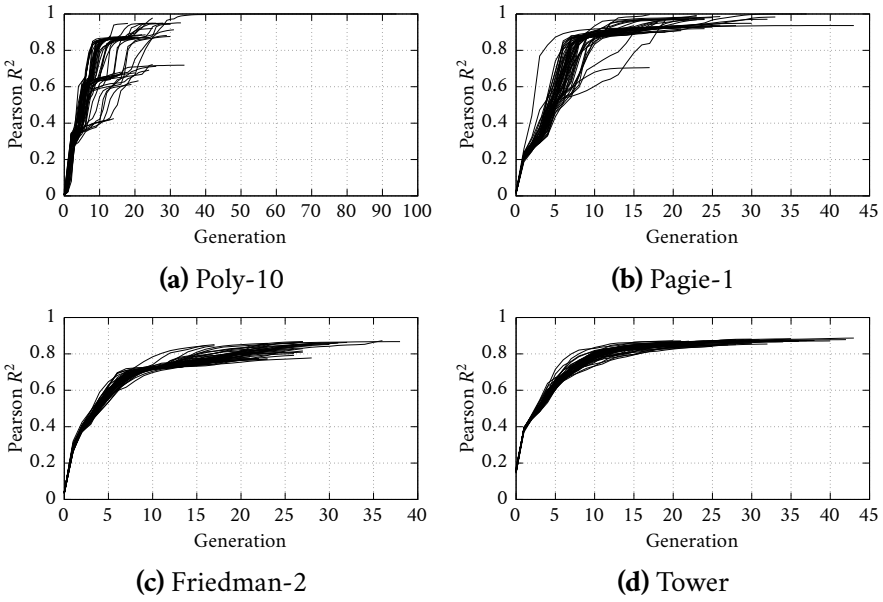
An overview of the best and average solution qualities achieved by each algorithm configuration is provided in Tables 6.9 and 6.10. Overall, the OSGP algorithm performs significantly better than SGP (especially on the Poly-10 and Pagie-1 problems) due to the fact that OSGP is able to more efficiently exploit the initial genetic diversity present in the population. The offspring selection criteria makes sense from an evolutionary perspective: just like in nature, deleterious changes at the genotype level, leading to decreased fitness (compared to the parents) are not favored at all by selection. If in SGP the selection mechanism still gave chances to weaker individuals – accepting non-adaptive genetic variation – in OSGP only *adaptive change* is allowed to be transmitted from one generation to the next. Therefore, the success of the OSGP algorithm is directly influenced by the evolvability of the population (the ability to produce adaptive change). From this perspective, the maximum selection pressure termination criteria acts as an indicator for when the population has lost its adaptive potential.

Looking at the qualities alone, we can conclude from Tables 6.9 and 6.10 that the two configurations perform the same, with no statistically-significant differences between the average best solution qualities. We can thus infer that the strict offspring selection criteria has a bigger influence on the overall behavior of selection than the specific methods of parent selection (proportional or gender specific).

Due to the strict offspring selection criteria, in the case of OSGP the average population

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	0.8424 ± 0.1253	0.9477 ± 0.0396	0.8334 ± 0.0343	0.8713 ± 0.0080
	0.7511 ± 0.2050	0.8024 ± 0.2494	0.7916 ± 0.1108	0.8703 ± 0.0106
Gender Specific	0.8586 ± 0.1194	0.9560 ± 0.0282	0.8438 ± 0.0315	0.8768 ± 0.0081
	0.7712 ± 0.1971	0.7935 ± 0.2735	0.8006 ± 0.1168	0.8759 ± 0.0117

Table 6.10: OSGP Average Solution Qualities



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

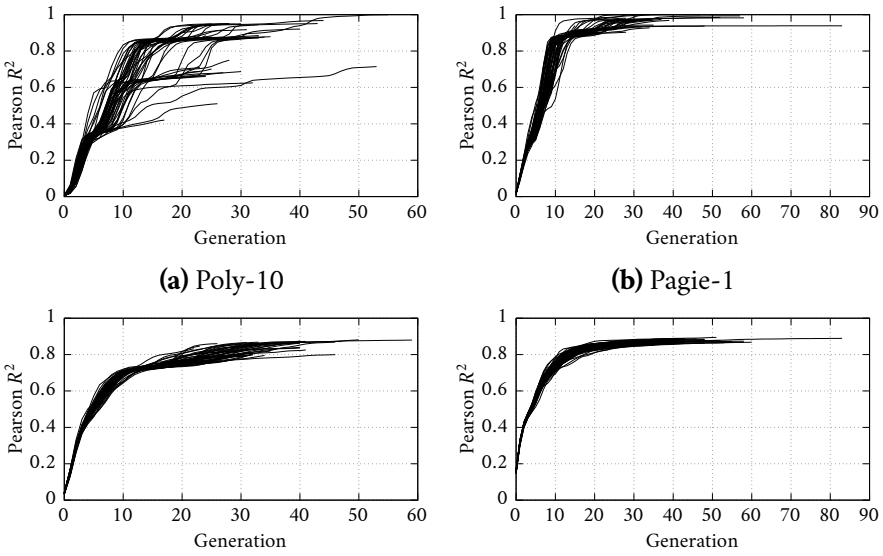
Figure 6.21: OSGP Proportional Selector Average Population Quality

quality is much higher throughout the evolutionary run compared to SGP, as shown in Figs. 6.21 and 6.22 .

6.2.3 Selection Ratio and Parent Distribution

The evolution of selection ratio shown in Fig. 6.23 validates the hypothesis that most of the initial population diversity is lost in the early stages of evolution. This happens due to the fact that in the early generations big jumps in fitness occur in a small number of individuals, leading to a highly uneven distribution of fitness values in the population, ultimately causing selection to discard a large number of individuals.

In the case of proportional selection, we notice that the selection ratio drops as low as 0.2 (for the Pagie-1 problem) or 0.4 (for the Friedman-2) problem. The situation is better in the case of OSGP with gender specific selection, where the random selection of the second parent alleviates this phenomenon, although it can still be quite pronounced for example



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.22: OSGP Gender Specific Selector Average Population Quality

on the Pagie-1 problem (Fig. 6.24b).

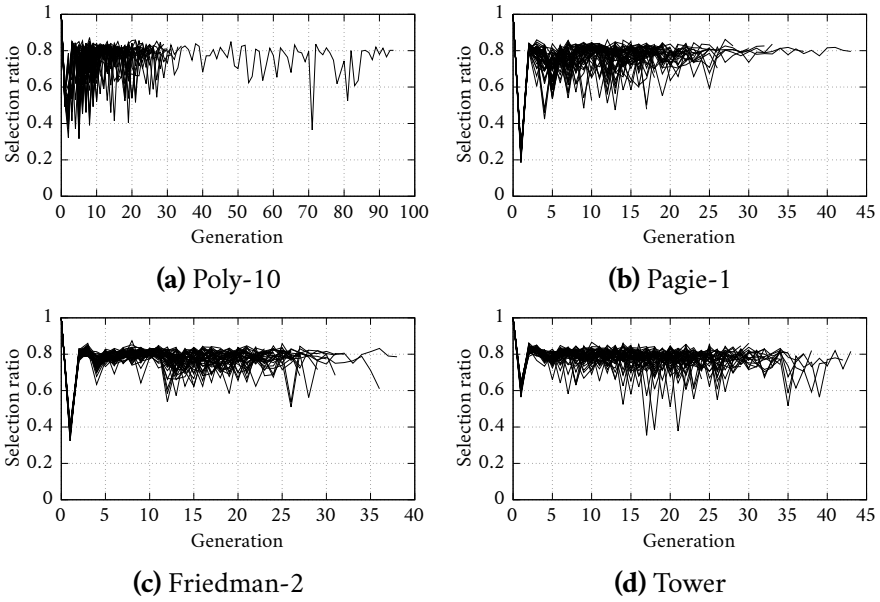
After the initial drop, the selection ratio increases and stabilizes to a value around 0.8 which is generally larger than the SGP selection ratio shown in Fig. 6.2. The comparison between the parent distribution charts between SGP (Figs. 6.3 and 6.4) and OSGP (Figs. 6.25 and 6.26) shows that selection behaviour in OSGP is similar with the behaviour of proportional selection in SGP, however with the important difference that OSGP selection (especially in the case of gender specific selection) also samples the tail of the parent quality distribution consisting of the least fit parents. This can be explained by the fact that OSGP selection acts not only on fitness but also on evolvability, making it possible for less fit individuals to participate as parents if they can produce children fitter than themselves, while high fitness individuals may be out of the evolutionary race if they can no longer be improved.

6.2.4 Average Tree and Fragment Length

Fig. 6.5 and Figs. 6.27 and 6.28 show that compared to SGP, OSGP runs are characterised by a higher average tree length.

Since SGP and OSGP use the same `PTC2` tree creator, we observe the same decrease in average tree size in the first few generations. The remaining differences between SGP and OSGP regarding average tree and fragment size can be explained by the different behavior of the strict offspring selection mechanism.

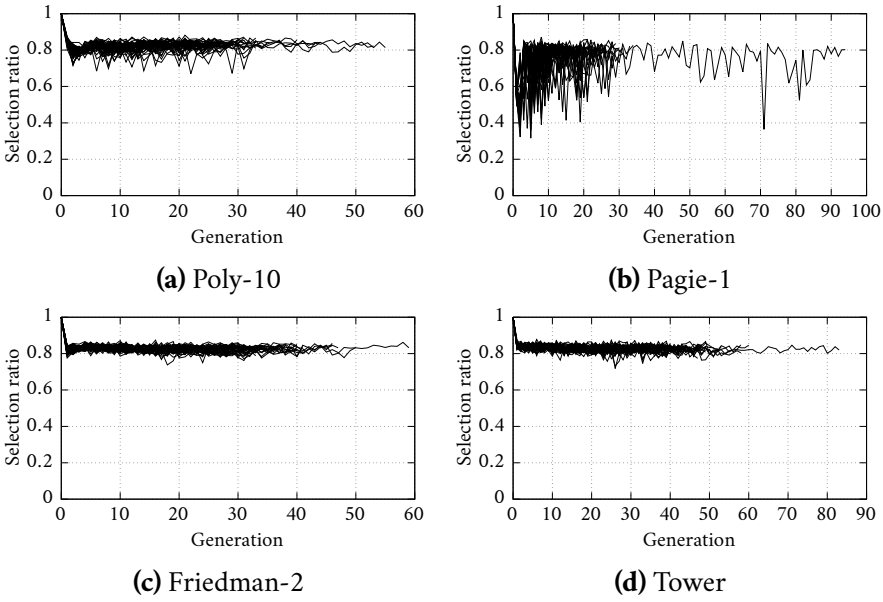
Strict offspring selection rejects deleterious changes and only accepts offspring fitter



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

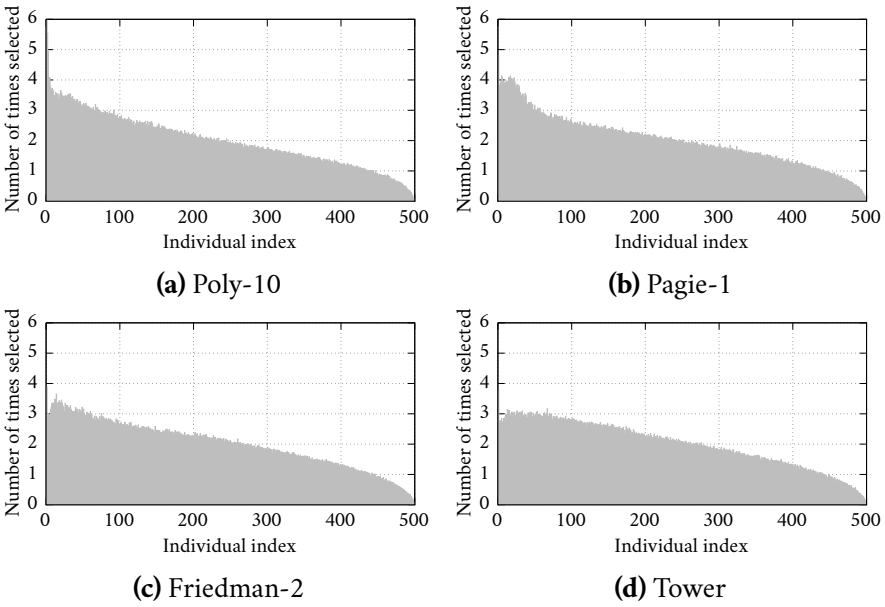
Figure 6.23: OSGP Proportional Selector Average Selection Ratio



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

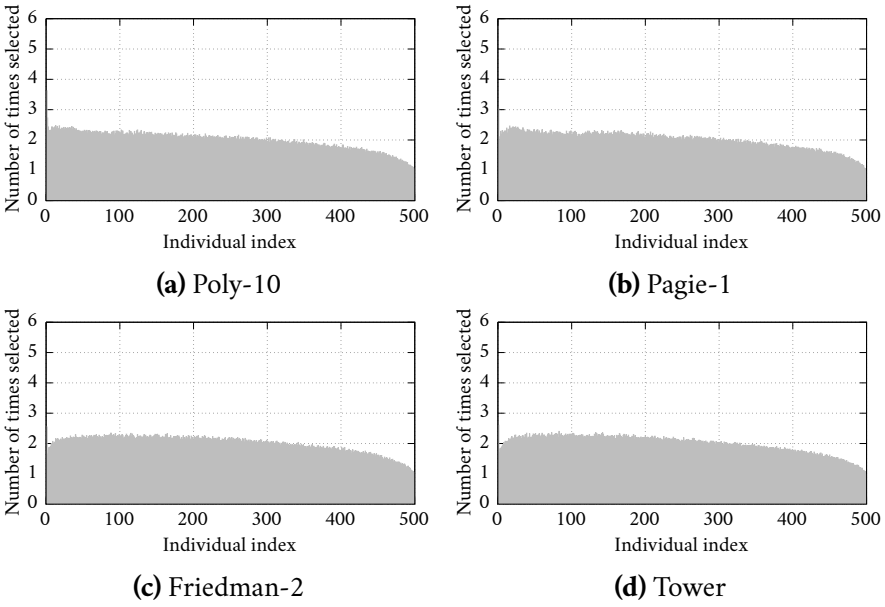
Figure 6.24: OSGP Gender Specific Selector Average Selection Ratio



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

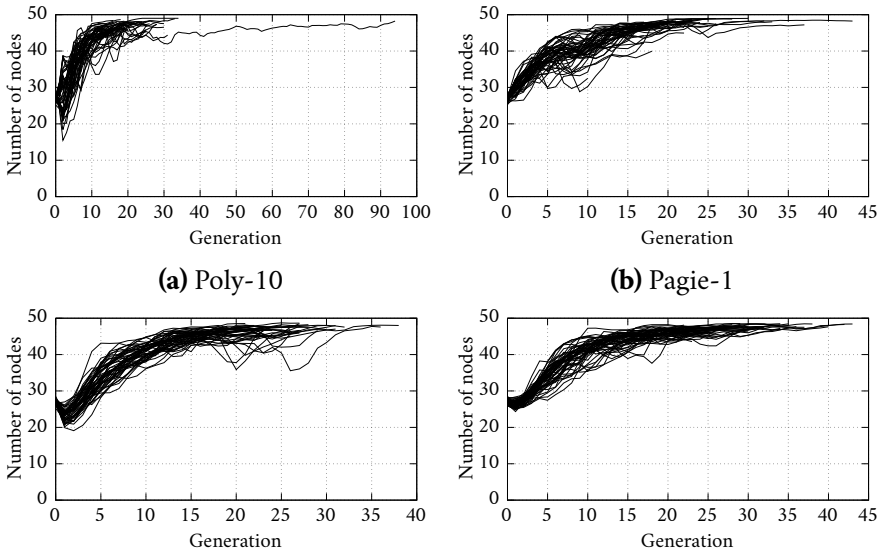
Figure 6.25: OSGP Proportional Selector Average Parent Distribution



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.26: OSGP Gender Specific Selector Average Parent Distribution



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.27: OSGP Proportional Selector Average Tree Length

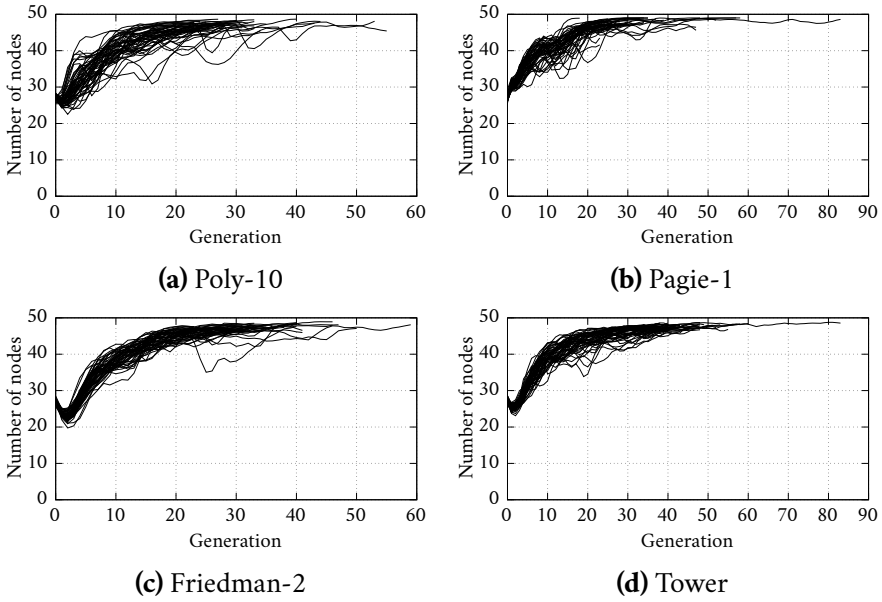
than their parents. Therefore, the increased OSGP average tree size compared to SGP can be attributed to the pressure for adaptive change leading to more genotype buffering and consequently, larger trees being favored by selection.

The increase in average crossover fragment size in the early generations (Figs. 6.29 and 6.30) is explained by the selection bias during the exploratory phase of the algorithm, when the algorithm begins to lock onto the local optima points. Intuitively, larger genetic changes are necessary to jump into different areas of the genotype space and when the overall fitness of the population is low (as it happens in the first few generations), these changes are the most likely to produce significant fitness improvements. As average fitness increases and adaptive changes are harder to come by, selection begins to favor smaller changes which are both less likely to damage fitness and less likely to dramatically improve it.

The buffering of genotypes and the high average fitness determines the average crossover fragment length to stabilize to a lower value for OSGP. Finally, the size of mutation fragments tends to be significantly lower for OSGP as most mutations (except the smaller ones) are likely to be rejected during the offspring selection phase.

6.2.5 Population Diversity

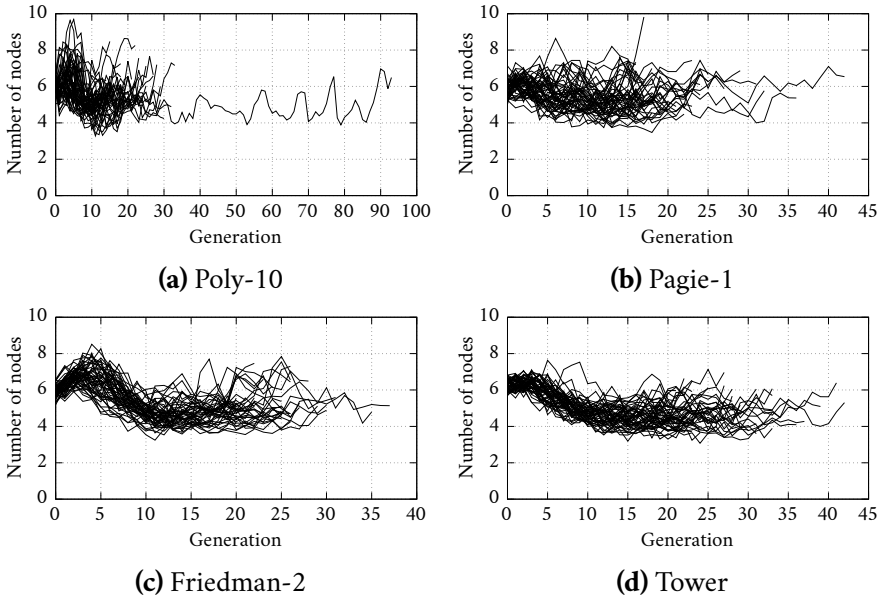
The genotype similarity curves in Figs. 6.33 and 6.34 show that OSGP populations are generally more genotypically similar than their SGP counterparts, although no great differences occur between the proportional and gender specific selection methods compared



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

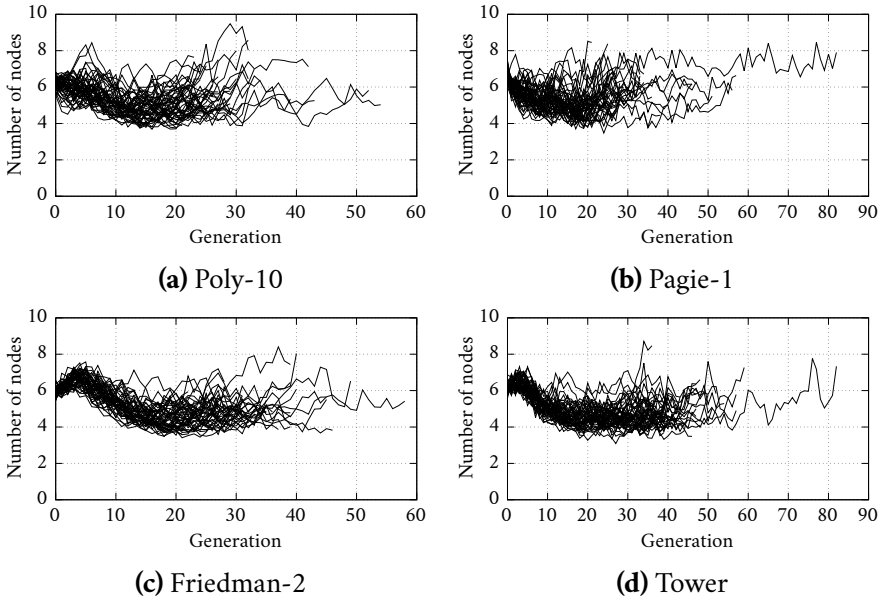
Figure 6.28: OSGP Gender Specific Selector Average Tree Length



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

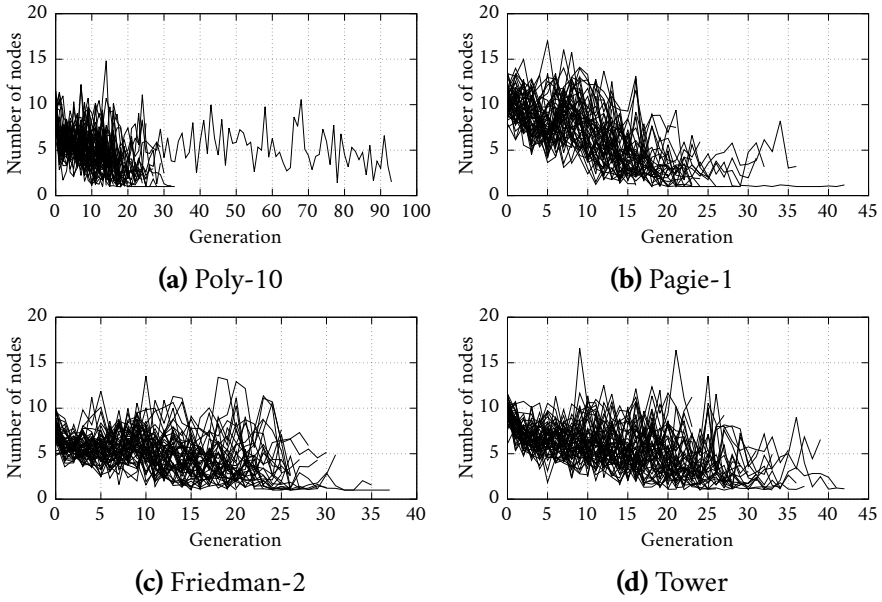
Figure 6.29: OSGP Proportional Selector Average Crossover Fragment Length



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

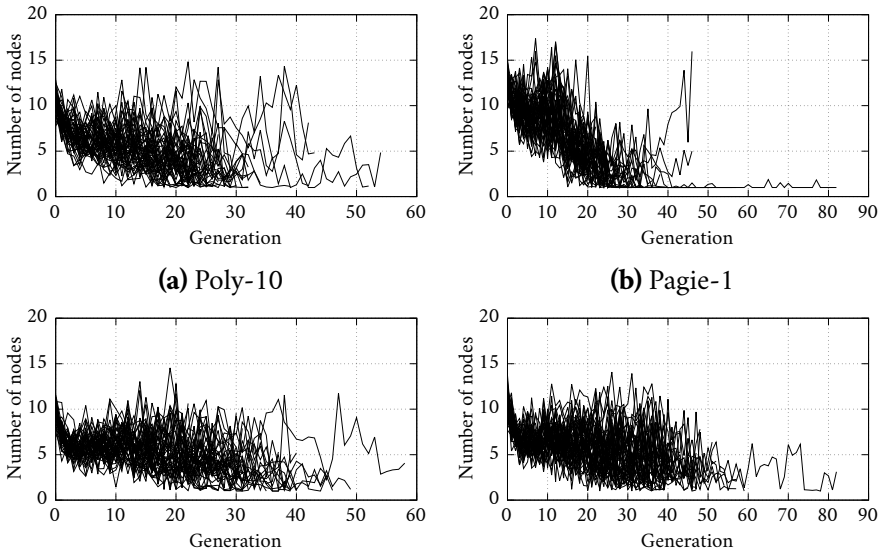
Figure 6.30: OSGP Gender Specific Selector Average Crossover Fragment Length



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.31: OSGP Proportional Selector Average Mutation Fragment Length



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.32: OSGP Gender Specific Selector Average Mutation Fragment Length

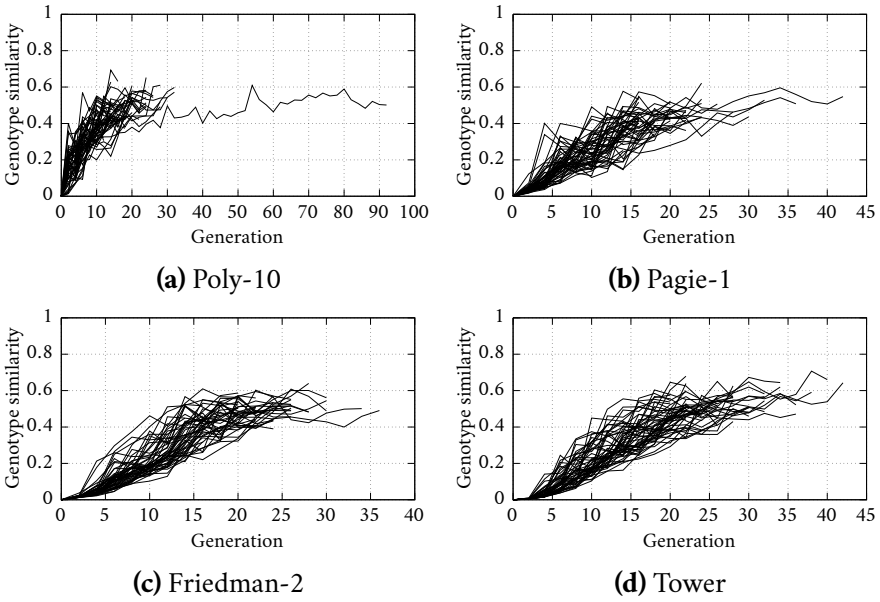
to the differences between SGP with proportional and tournament selection.

Phenotype similarity shown in Figs. 6.35 and 6.36 marks an important difference between SGP and OSGP. Due to the strict offspring selection criteria, most individuals in the population end up having similar semantics, leading to situations where the phenotypes in the population are 100% similar.

6.2.6 Genetic Operator Effectiveness

Figs. 6.37 to 6.40 show the effectiveness of genetic operators within the OSGP algorithm. While the genotypic operators themselves do not act any differently than for example in the case of SGP, the additional offspring selection criteria accepting only adaptive change makes them "effective", in the sense that deleterious changes are simply rejected. Therefore, the figures show us the rate at which fitness can be improved with the help of the offspring selection criteria.

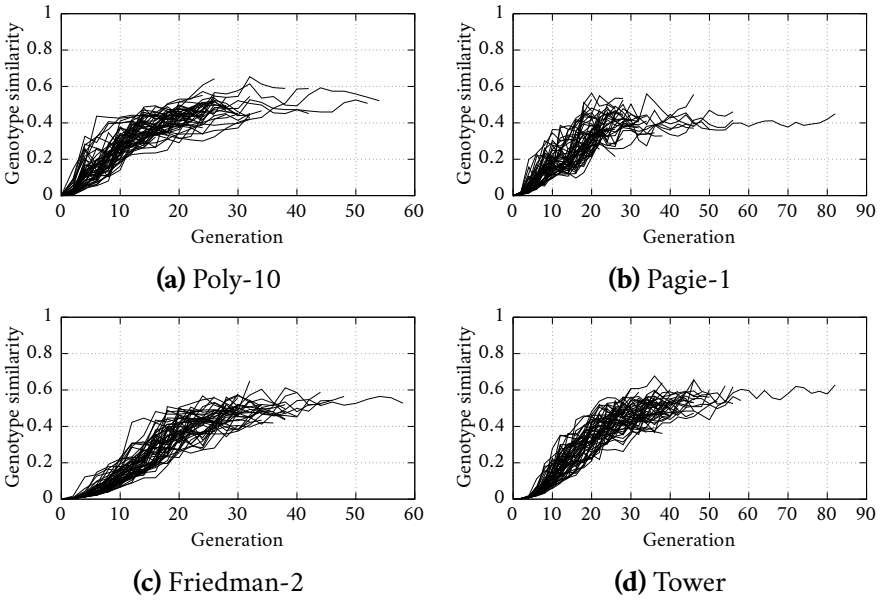
The results show that on average qualities are quite slowly improved regardless of problem or algorithm configuration. Therefore, the offspring selection criteria seems to play a protective role against deleterious effects, rather than actively encouraging adaptation at genotype level. Nevertheless the quality curves show that these incremental improvements can make a significant difference in the overall performance of the algorithm, allowing OSGP to obtain superior results compared to SGP on all problem instances and with a lower population size.



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

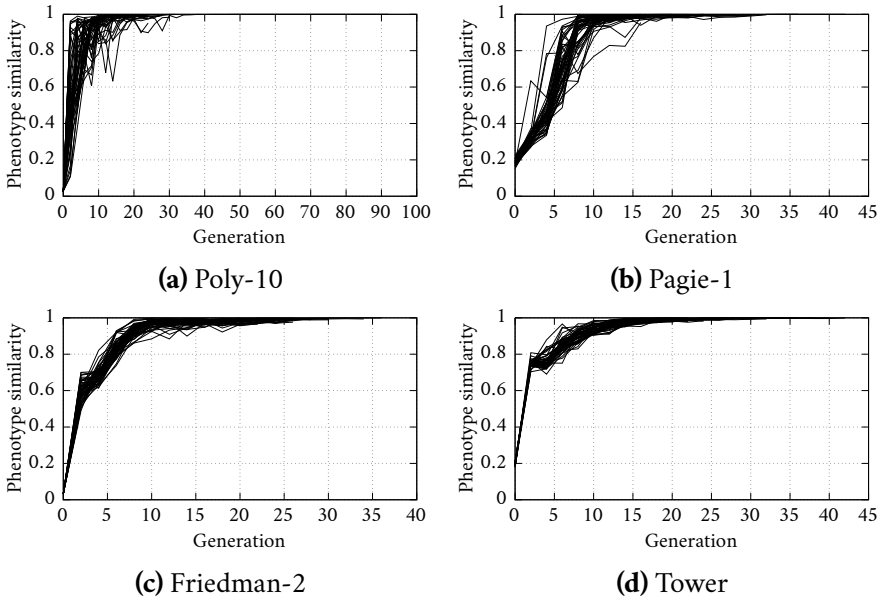
Figure 6.33: OSGP Proportional Selector Average Genotype Similarity



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

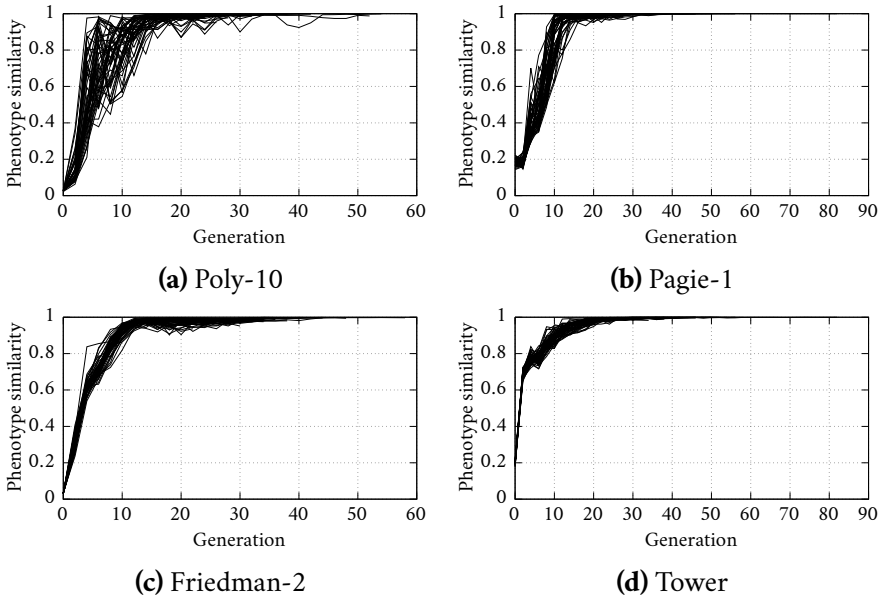
Figure 6.34: OSGP Gender Specific Selector Average Genotype Similarity



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

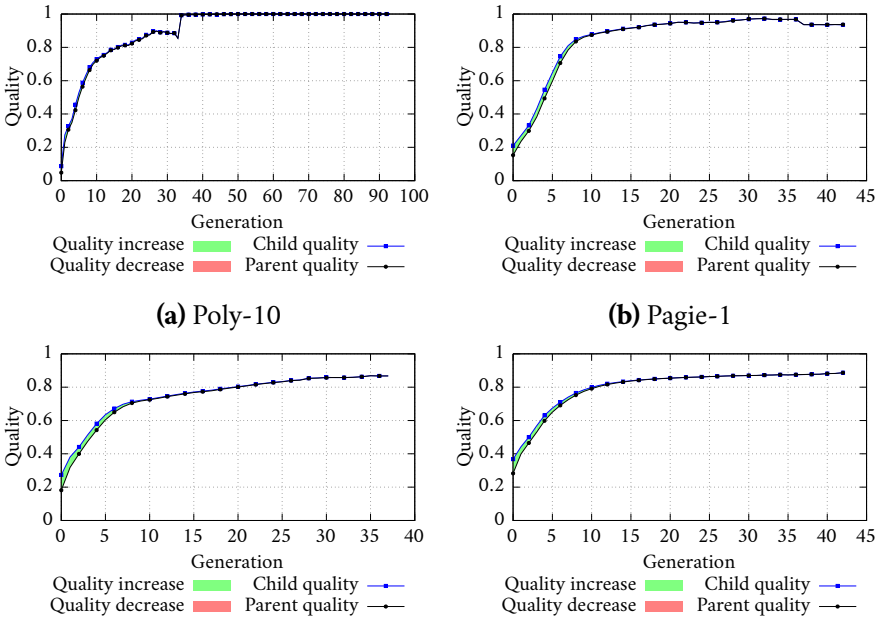
Figure 6.35: OSGP Proportional Selector Average Phenotype Similarity



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

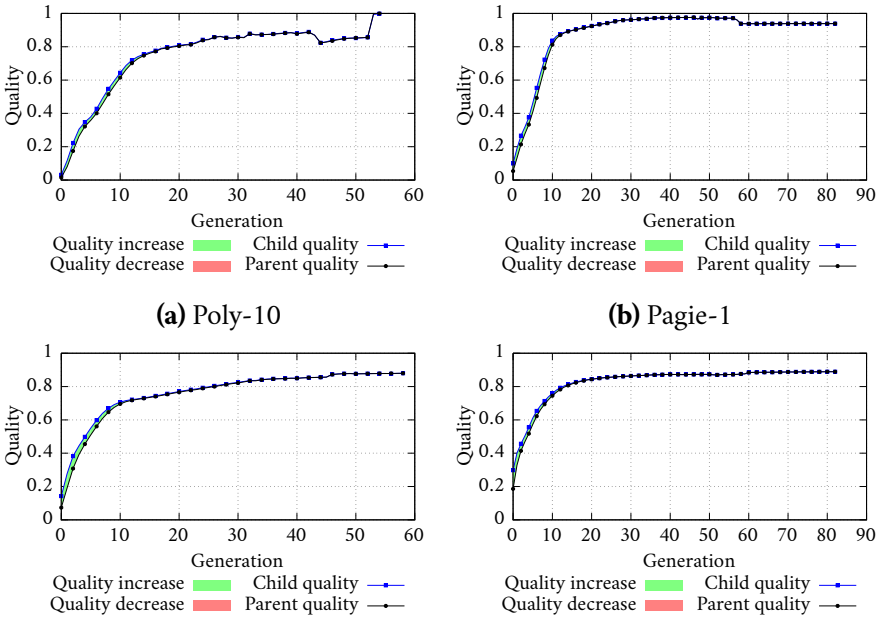
Figure 6.36: OSGP Gender Specific Selector Average Phenotype Similarity



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

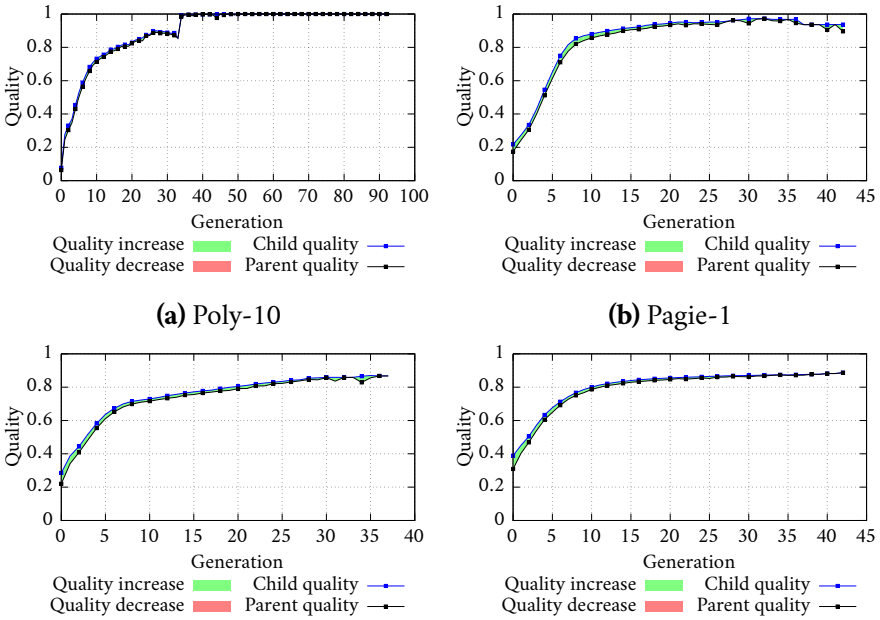
Figure 6.37: OSGP Proportional Selector Average Crossover Improvement



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

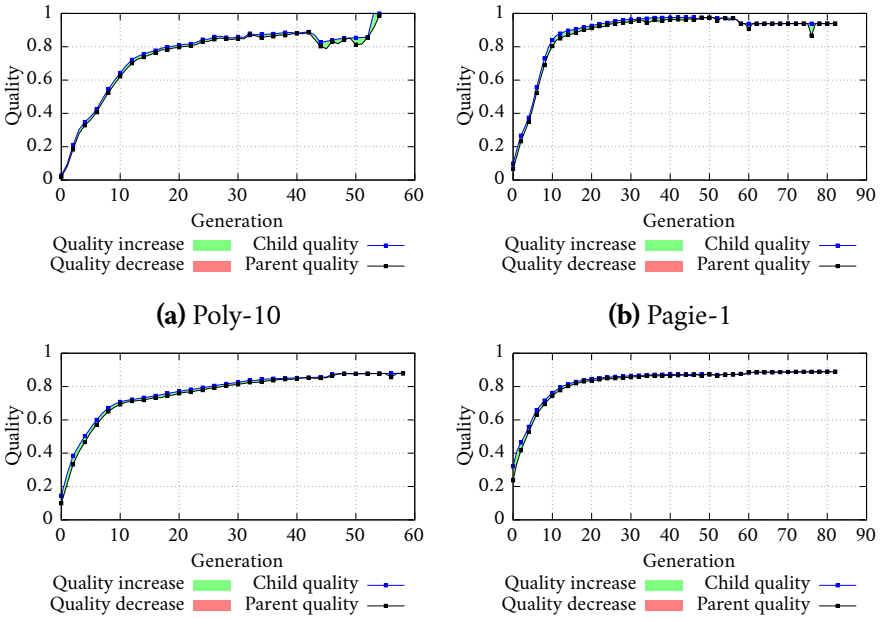
Figure 6.38: OSGP Gender Specific Selector Average Crossover Improvement



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

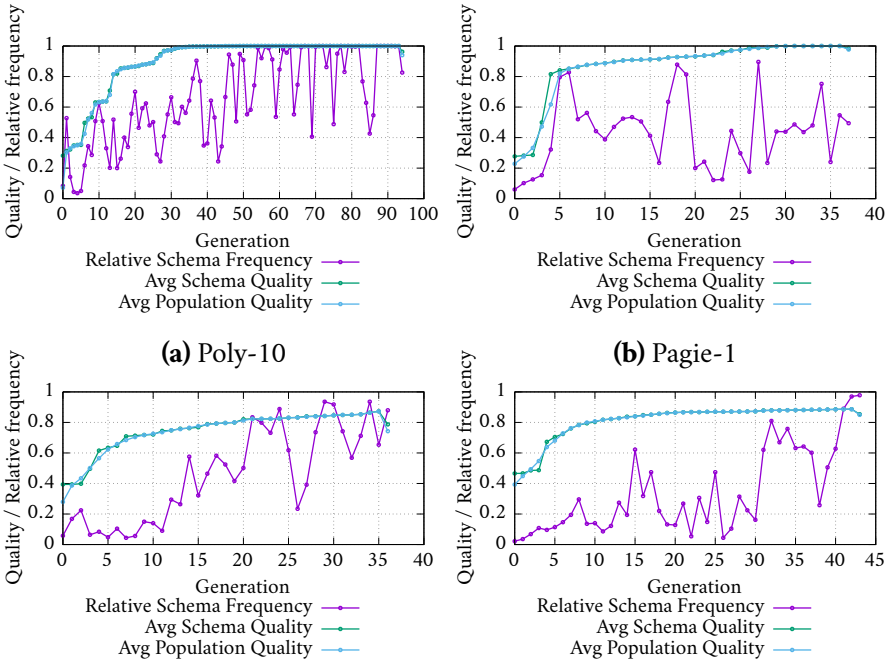
Figure 6.39: OSGP Proportional Selector Average Mutation Improvement



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.40: OSGP Gender Specific Selector Average Mutation Improvement



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

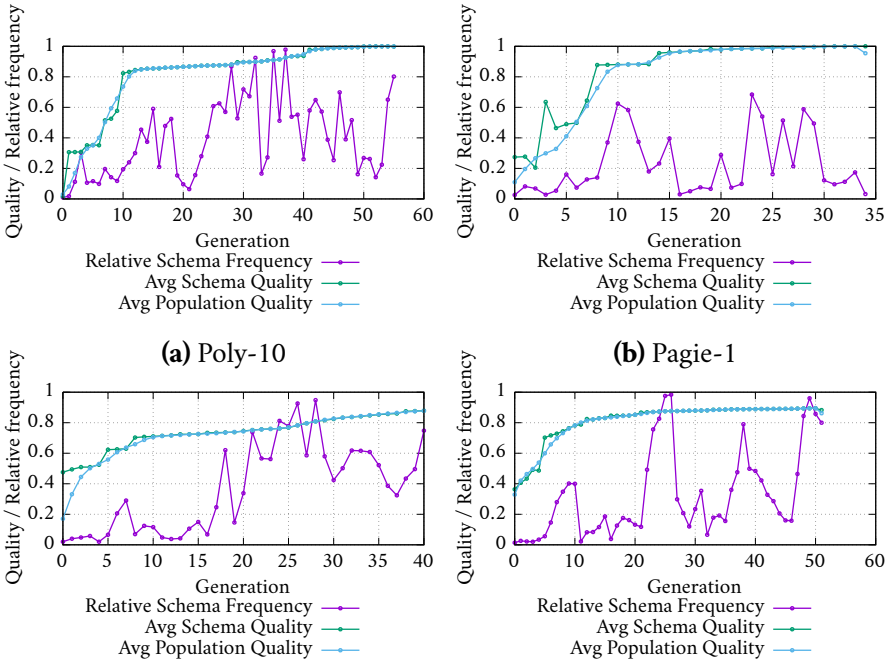
Figure 6.41: OSGP Proportional Selector Schema Frequencies and Quality

6.2.7 Schema Frequency Analysis

Schema frequencies for the best OSGP runs in Figs. 6.41 and 6.42 show the average quality and frequency of the most common schema of the best run from each algorithmic configuration:

- ◇ Average schema qualities follow closely the average population quality, as strict offspring selection leads to a more uniform distribution of qualities in the population and a higher degree of genotypic interrelatedness.
- ◇ In many instances the most common schema matches a very high proportion of individuals. Spikes in relative schema frequency that go even above 0.9 (ie., the schema matches more than 90% of the population) can be observed in Figs. 6.41a to 6.41d, 6.42a, 6.42c and 6.42d.

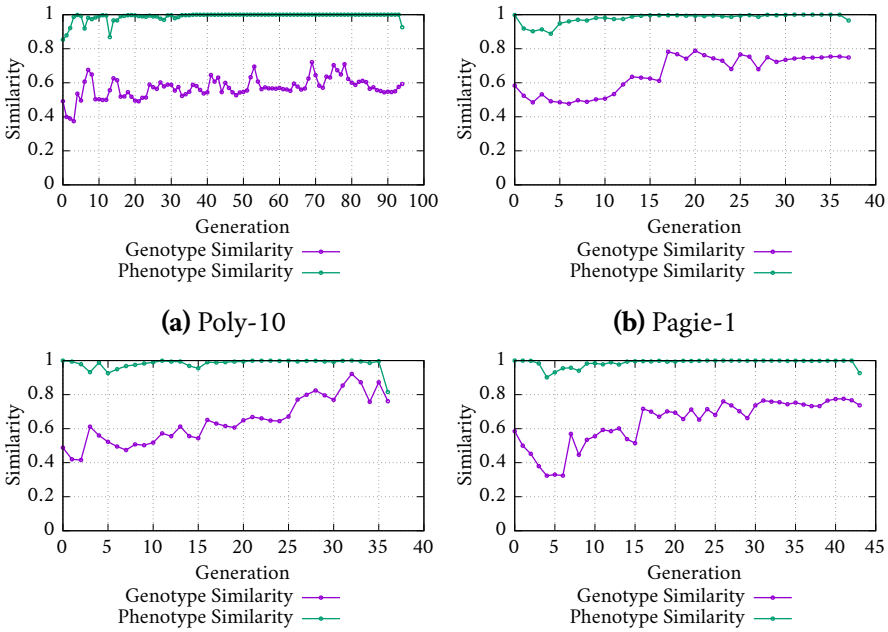
At the same time, schema similarities in Figs. 6.43 and 6.43 show that the individuals matching OSGP schemas represent identical phenotypes (as the phenotypic similarity curves are very close to 1) and share a high degree of genotypic similarity. From this we can conclude that offspring selection leads not only to high levels of genotype and phenotype similarity but also to a common genetic template underlying the whole population, from which it may prove difficult to evolve further, once the search begins to converge.



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

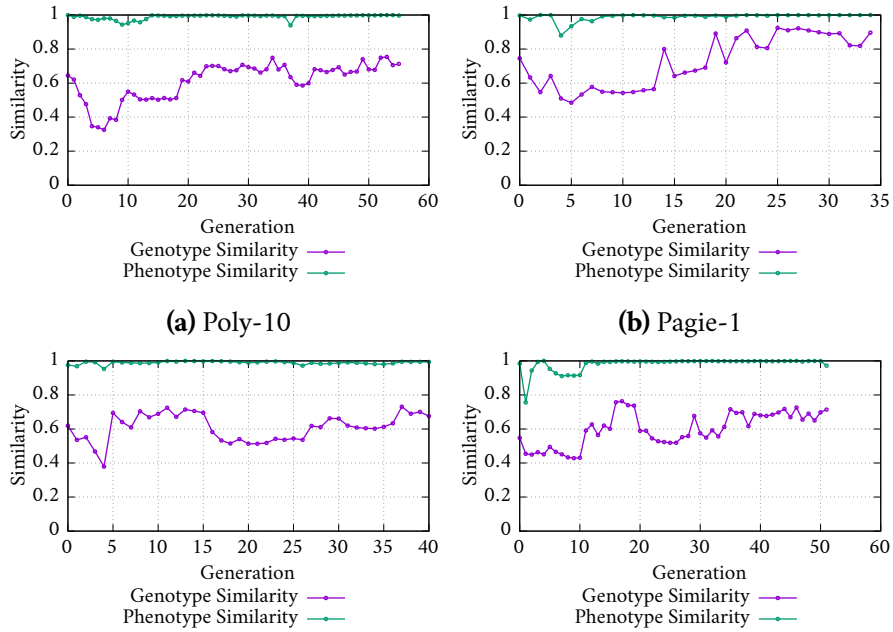
Figure 6.42: OSGP Gender Specific Selector Schema Frequencies and Quality



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.43: OSGP Proportional Selector Schema Similarities



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.44: OSGP Gender Specific Selector Schema Similarities

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	2%	3%	3.2%	1.7%
Gender Specific	2.2%	1.6%	1.9%	3.6%

Table 6.11: OSGP Best Solution Contribution Ratio

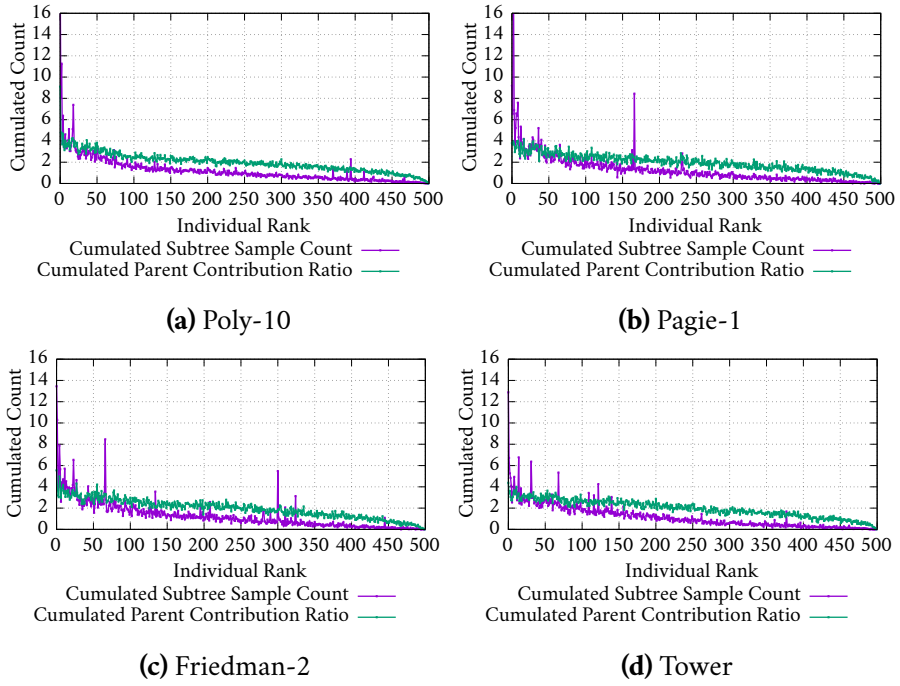
6.2.8 Analysis of Solution Contribution Ratio

Table 6.11 shows the solution contribution ratio calculated for the best solutions of the OSGP algorithm. As the contribution ratio is calculated solely on the basis of the genotypes in the population and their hereditary relationships, it gives a good indication of the amount of genetic diversity lost through the action of selection during the evolutionary search. At the same time, the contribution ratio acts as a measure of the effort expended by the algorithm in producing the best solution, since it directly quantifies how many of the total genotypic operations that acted on the ancestors of the best solution had a direct contribution to its structure.

For the sake of completeness, Table 6.12 shows the trace graph size on the first line and ancestry size on the line below for the best solution of each tested configuration. In this case, the size variation can also be attributed to the fact that OSGP runs can differ in the number of generations until the maximum selection pressure termination criteria is

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	481	293	276	207
	24367	9681	8500	11903
Gender Specific	396	148	238	612
	18174	9512	12748	17172

Table 6.12: OSGP Best Solution Trace v. Ancestry Size



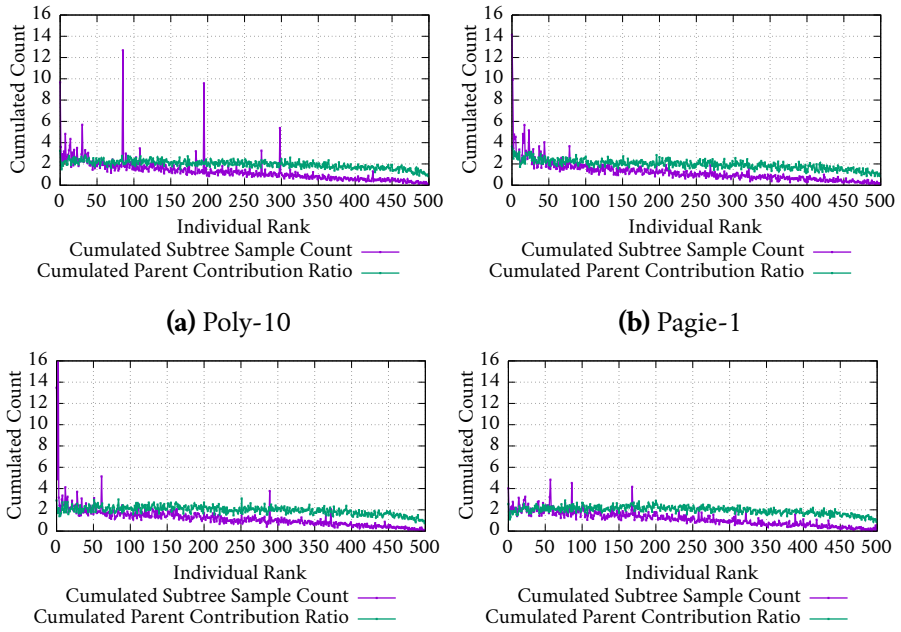
(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.45: OSGP Proportional Selector Cumulated Subtree Sample Count and Parent Contribution Ratio

satisfied.

Considering the mapping of genotypes into phenotypes due to the G→P map, a lower contribution ratio corresponds to a smaller region of the genotype space sampled by the genetic operators during the algorithm's attempt of producing adaptive genetic change. This region is continuously reduced by loss of diversity (every generation), up to the point where fitness is locally maximized and the available pool of genetic material is limited to a reduced common set of inherited genes, originating from a small number of "super-ancestors". This behaviour can be also noticed on the subtree sample count curves in Figs. 6.45 and 6.46. The spikes in these curves correspond to very fit building blocks propagated in the whole population by the selection operator. Through their multiplicity and high fitness, such genes end up dominating the population, thus reducing the algorithms exploratory potential.



(a) Poly-10 **(b)** Pagie-1

(c) Friedman-2 **(d)** Tower

Figure 6.46: OSGP Gender Specific Selector Cumulated Subtree Sample Count and Parent Contribution Ratio

Subtree	Count
(* x_9 x_7)	5584
(* x_6 x_5)	5060
(* x_1 x_{10})	2896
(* x_1 x_1)	2643
(+ (* x_9 x_7) x_2)	1804
(* x_1 x_2)	1482
(+ x_7 x_2)	894
(* x_3 x_{10})	883
(* x_1 (+ (* x_9 x_7) x_2))	807
(* (* x_1 x_1) x_9)	471

$f(\mathbf{x}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$

(a) Poly-10

Subtree	Count
(/ (/ c y) y)	1861
(/ (/ c x) x)	447
(/ (/ c y) (+ (/ c y) y))	400
(+ (/ c y) y)	324
(- (+ c c) (/ (/ c y) y))	319
(- y y)	288
(/ c (- (+ c c) (/ (/ c y) y)))	281
(* y y)	233
(/ (/ (/ c x) x) x)	199
(/ (/ (/ (/ c x) x) x) x)	191

$f(x, y) = \frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$

(b) Pagie-1

Subtree	Count
(* x_1 (* c x_2))	1185
(* x_5 x_5)	821
(+ x_4 x_2)	398
(- x_1 x_5)	316
(* x_1 (* x_1 (* c x_2)))	309
(* x_5 x_2)	260
(* (* (* c x_2) (/ c c)) x_1)	219
(* (/ x_1 (* x_3 c)) (* c x_2))	184
(- c (- (* (* x_1 (* x_1 (* c x_2))) (* (* c x_2) (/ c c))) c))	173
(+ (+ x_5 c) (+ x_4 x_2))	161

$f(\mathbf{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5 + noise$

(c) Friedman-2

Subtree	Count
(- x_{12} x_{12})	589
(* x_{15} (/ x_1 c))	570
(- x_1 x_6)	534
(+ x_{16} x_{23})	438
(/ x_{22} x_{15})	384
(+ (* x_6 c) (* x_{15} (/ x_1 c)))	309
(- x_9 x_1)	301
(* (+ c x_{23}) (- (+ (* x_6 c) (* x_{15} (/ x_1 c))) x_{23}))	296
(+ x_{24} x_{16})	264
(- x_{22} x_{12})	236

Unknown function of 25 variables

(d) Tower

Table 6.13: OSGP Proportional Selector Most Sampled Subtrees

Subtree	Count
(* x_6 x_5)	4043
(* x_2 x_1)	2876
(* x_6 x_{10})	1522
(+ (* x_2 x_1) (* x_6 x_5))	875
(* x_9 x_1)	832
(* (* x_6 x_{10}) x_3)	642
(* x_9 (* x_1 c))	530
(+ (* x_6 x_5) x_6)	410
(* x_3 (* x_4 c))	313
(- c (* (* x_4 c) (* x_6 x_{10})))	277

$f(\mathbf{x}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$

(a) Poly-10

Subtree	Count
(* y y)	975
(* y (* y c))	839
(* x (* c x))	602
(/ c (- c (/ (* (/ (* (+ c c) c) (* y (* y c))) c) (* y (* y c)))))	421
(- c (/ (* (/ (* (+ c c) c) (* y (* y c))) c) (* y (* y c))))	395
(/ (* (/ (* (+ c c) c) (* y (* y c))) c) (* y (* y c)))	363
(/ (- c c) (- c (- (* x (* x (* c x))) x)) c)))	350
(* (* x (* c x)) x)	350
(* x (* (* x (* c x)) x))	344
(/ c (* y y))	341

$f(x, y) = \frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$

(b) Pagie-1

Subtree	Count
(* x_2 x_1)	3404
(+ x_4 x_1)	2091
(+ (* x_2 x_1) c)	1362
(+ x_4 x_5)	861
(+ (+ (* x_2 x_1) c) c)	587
(+ x_6 x_5)	404
(+ (* x_2 x_1) (+ (* x_2 x_1) c))	343
(* (+ x_4 x_1) c)	322
(+ (+ x_6 x_5) x_2)	315
(* x_2 (* x_2 x_1))	302

$f(\mathbf{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5 + noise$

(c) Friedman-2

Subtree	Count
(+ x_6 x_{23})	1426
(+ x_6 x_1)	1138
(/ (+ x_{12} c) x_6)	994
(/ x_2 x_1)	889
(/ (/ (+ x_{12} c) x_6) x_{15})	547
(/ (- x_9 c) x_6)	176
(/ (- x_9 c) (+ x_6 x_1))	164
(+ x_8 x_1)	152
(+ x_6 x_{25})	116
(/ x_3 x_1)	116

Unknown function of 25 variables

(d) Tower

Table 6.14: OSGP Gender Specific Selector Most Sampled Subtrees

Tables 6.13 and 6.14 show the most sampled subtrees by the OSGP algorithm:

- On the Poly-10 problem, both OSGP configurations using the proportional and the gender-specific selection mechanisms were able to produce perfect solutions, in which all the terms of the formula were correctly identified by the algorithm. These terms are represented by the most sampled subtrees in Tables 6.13a and 6.14a, including $x_1x_7x_9$ and $x_3x_6x_{10}$.
- On the Pagie-1 problem, we notice the presence of the terms x^4 and y^4 in the most sampled subtrees from both configurations (Tables 6.13b and 6.14b), along with other combinations of constants and powers of x and y. When the essential building blocks are present in the population, the algorithm is able to find the exact solution, as shown by the perfect quality obtained in Table 6.9.
- On the Friedman-2 problem, OSGP is able to find the relevant variable combinations such as x_1x_2, $x_4 + x_5$, although the most sampled subtrees list does not offer any evidence about the presence of x_3 as a standalone term in the formula.
- On the Tower problem, the most sampled subtrees by the OSGP algorithm contain a similar set of variables to those identified by SGP: x_1, x_2, x_3, x_6, x_8, x_9, x_{12}, x_{15}, x_{22},

x_{23}, x_{24}, x_{25}. The sample counts indicate x_{12} and the sum $x_6 + x_{23}$ as being the most important for the obtained solutions.

6.3 Offspring Selection GP with Schemas

We tested the algorithmic improvement suggested in Section 5.2.5 using the same OSGP configuration from Section 6.2, and using a gender specific selection scheme. In addition to a maximum selection pressure of 100, a maximum of five million evaluated solutions was also set as a termination criteria. This limit was necessary to prevent very lengthy algorithmic runs, as the extra diversification step acts against local convergence by introducing new genetic diversity in the population (through mutation). Then, the more diverse parents, combined with the lower qualities caused by mutation (see Sections 6.1.6 and 6.2.6) make it easier to produce offspring better than their parents under gender specific selection, effectively reducing the necessary effort (thus, the active selection pressure) to obtain adaptive change. Ultimately, this has the consequence of allowing the algorithm to run for a long time fueled by continuous mutation.

6.3.1 OSGP-S Adaptive Replacement Ratio

Using an adaptive replacement ratio means that the amount of mutation applied to the matching individuals of schemas that fulfil the diversification criteria is calculated as a function of the relative schema frequency in the population. The idea is to allow lower frequency schemas to propagate further by applying less mutation and conversely, to prevent higher frequency schemas from dominating the population by applying more mutation, with the overall effect of improving population diversity while allowing schemas to grow.

Table 6.15 shows the configuration for the first experiment, using a number of different adaptive replacement ratio rules, as well as two different schema matching rules called "strict" and "relaxed" (non-strict). The tested replacement ratio rules are described in Fig. 5.7. The schema matching rules refer to the conditions under which two tree nodes are considered to be equal by the pattern matching algorithm. The following requirements need to be satisfied:

1) Both nodes have to be of the same kind, either function nodes (representing the same function) or leaf nodes (either constants or variables). Variable tree nodes have to reference the same variable in the dataset.

2) In addition, strict matching requires leaf values to be exactly the same. By leaf values, we mean the numerical values of constant nodes or the numerical weights of the variable nodes.

Replacement Ratio	Adaptive (all rules)
Minimum Phenotypic Similarity	90%, 95%
Minimum Schema Length	10
Minimum Schema Frequency	0%
Schema Matching	Strict, Relaxed

Table 6.15: OSGP-S Adaptive Replacement Ratio Experiment Configuration

We use the notation OSGP-S-Adaptive to denote the schema-based diversification strategy with adaptive replacement ratio. The effects of the schema diversification strategy on solution quality are displayed in Table 6.16. The results show that solution quality is mainly influenced by the schema matching parameter, while the quality differences between the tested schema replacement rules remain small. Relaxed schema matching (Table 6.16a) produces the better results as it provides a better indication of local convergence in combination with the minimum phenotypic similarity criteria.

Having established that OSGP-S actually performs significantly better using non-strict schema matching, the next step is to test the influence of the minimum phenotypic similarity parameter on the search outcome. We repeated the previous experiment using an OSGP-S configuration with a minimum phenotypic similarity threshold of 95% and relaxed matching. The results in Table 6.17 show that too restrictive diversification criteria such as strict matching or high thresholds for phenotypic similarity may limit the method's potential to shift the focus of the search (when already in a state of advanced convergence) from local to global. The overall qualities are comparable to the ones from Table 6.16 with the exception of the Poly-10 problem, where they are significantly worse. This leads us to conclude that a too high threshold for the phenotypic similarity within schemas does not bring any practical benefits.

6.3.2 OSGP-S Fixed Replacement Ratio

The matter was further investigated by testing the two phenotypic similarity thresholds (90% and 95%) using a fixed replacement ratio set to 90% of the matching individuals and two different settings for the minimum schema length parameter: 10 and 25. For a clearer notation, we abbreviate this algorithmic variant as OSGP-S-Fixed.

Table 6.18 shows that OSGP-S-Fixed produced better results with a minimum schema length of 25. The minimum schema length affects the diversification strategy in two ways:

1) It tends to lead to the creation of more specific schemas, ie., schemas where the wildcards symbols are situated lower in the tree. Fewer individuals will match these higher-order schemas.

2) Higher-order schemas matched by fewer individuals will tend to have higher average phenotypic similarity (between the matching individuals).

Replacement rule	Poly-10	Pagie-1	Friedman-2	Tower
$f(x) = x$	0.9765 ± 0.0788	0.9767 ± 0.0219	0.9087 ± 0.0441	0.8839 ± 0.0080
	0.9549 ± 0.1401	0.8810 ± 0.2187	0.8757 ± 0.1017	0.8823 ± 0.0096
$f(x) = \tanh(x)$	0.9517 ± 0.0860	0.9767 ± 0.0179	0.9044 ± 0.0473	0.8828 ± 0.0067
	0.9352 ± 0.1244	0.8430 ± 0.2606	0.8814 ± 0.0797	0.8765 ± 0.0381
$f(x) = \tanh(2x)$	0.9581 ± 0.0991	0.9763 ± 0.0214	0.9224 ± 0.0365	0.8849 ± 0.0054
	0.9390 ± 0.1385	0.8298 ± 0.2941	0.9054 ± 0.0622	0.8838 ± 0.0077
$f(x) = \tanh(3x)$	0.9737 ± 0.0474	0.9828 ± 0.0193	0.9052 ± 0.0411	0.8825 ± 0.0059
	0.9517 ± 0.1146	0.8496 ± 0.2743	0.8941 ± 0.0562	0.8798 ± 0.0237
$f(x) = \tanh(4x)$	0.9670 ± 0.0998	0.9779 ± 0.0191	0.9013 ± 0.0413	0.8856 ± 0.0083
	0.9431 ± 0.1902	0.8365 ± 0.2742	0.8627 ± 0.1064	0.8850 ± 0.0095
$f(x) = 1 - \sqrt{1 - x}$	0.9611 ± 0.0542	0.9723 ± 0.0197	0.9040 ± 0.0470	0.8840 ± 0.0074
	0.9477 ± 0.0737	0.8475 ± 0.2458	0.8808 ± 0.0856	0.8831 ± 0.0087

(a) Relaxed Schema Matching

Replacement rule	Poly-10	Pagie-1	Friedman-2	Tower
$f(x) = x$	0.8948 ± 0.1272	0.9721 ± 0.0227	0.8846 ± 0.0436	0.8840 ± 0.0087
	0.8028 ± 0.2804	0.8316 ± 0.2474	0.8707 ± 0.0514	0.8814 ± 0.0138
$f(x) = \tanh(x)$	0.8795 ± 0.1571	0.9704 ± 0.0225	0.8844 ± 0.0522	0.8832 ± 0.0074
	0.8211 ± 0.2423	0.8578 ± 0.2400	0.8432 ± 0.1215	0.8823 ± 0.0096
$f(x) = \tanh(2x)$	0.9181 ± 0.1252	0.9619 ± 0.0266	0.8986 ± 0.0405	0.8858 ± 0.0077
	0.8486 ± 0.2572	0.7827 ± 0.2847	0.8646 ± 0.1129	0.8853 ± 0.0083
$f(x) = \tanh(3x)$	0.9403 ± 0.0766	0.9671 ± 0.0227	0.8884 ± 0.0482	0.8848 ± 0.0072
	0.9097 ± 0.1537	0.7204 ± 0.3483	0.8725 ± 0.0591	0.8835 ± 0.0092
$f(x) = \tanh(4x)$	0.9247 ± 0.0964	0.9645 ± 0.0384	0.8781 ± 0.0451	0.8840 ± 0.0088
	0.8829 ± 0.1655	0.8767 ± 0.2209	0.8626 ± 0.0571	0.8818 ± 0.0120
$f(x) = 1 - \sqrt{1 - x}$	0.8561 ± 0.1481	0.9600 ± 0.0273	0.8700 ± 0.0510	0.8823 ± 0.0078
	0.7534 ± 0.2801	0.8654 ± 0.1959	0.8429 ± 0.0945	0.8811 ± 0.0087

(b) Strict Schema Matching

Table 6.16: OSGP-S Adaptive Replacement Ratio, Minimum Phenotypic Similarity 90%. The results are expressed as $(\mu \pm \sigma)$ values of the training and test qualities in the first and second line, respectively.

Replacement rule	Poly-10	Pagie-1	Friedman-2	Tower
$f(x) = x$	0.9194 ± 0.1351	0.9752 ± 0.0219	0.9148 ± 0.0444	0.8873 ± 0.0069
	0.8806 ± 0.2180	0.8029 ± 0.2996	0.9011 ± 0.0588	0.8803 ± 0.0477
$f(x) = \tanh(x)$	0.9274 ± 0.1133	0.9778 ± 0.0180	0.9164 ± 0.0421	0.8870 ± 0.0091
	0.8825 ± 0.2085	0.8722 ± 0.2361	0.8962 ± 0.0885	0.8861 ± 0.0112
$f(x) = \tanh(2x)$	0.9567 ± 0.0942	0.9774 ± 0.0181	0.9183 ± 0.0412	0.8842 ± 0.0071
	0.9104 ± 0.2184	0.8170 ± 0.2606	0.9055 ± 0.0651	0.8809 ± 0.0144
$f(x) = \tanh(3x)$	0.9327 ± 0.1219	0.9817 ± 0.0180	0.9083 ± 0.0394	0.8860 ± 0.0067
	0.8978 ± 0.2003	0.8666 ± 0.2325	0.8834 ± 0.0783	0.8844 ± 0.0089
$f(x) = \tanh(4x)$	0.9578 ± 0.0667	0.9792 ± 0.0201	0.9063 ± 0.0528	0.8846 ± 0.0071
	0.9223 ± 0.1623	0.7772 ± 0.3231	0.8913 ± 0.0734	0.8839 ± 0.0082
$f(x) = 1 - \sqrt{1 - x}$	0.8896 ± 0.1296	0.9742 ± 0.0309	0.9070 ± 0.0472	0.8853 ± 0.0079
	0.8140 ± 0.2446	0.7949 ± 0.3087	0.8928 ± 0.0689	0.8836 ± 0.0085

(a) Relaxed Schema Matching

Table 6.17: OSGP-S Adaptive Replacement Ratio, Minimum Phenotypic Similarity 95%. The results are expressed as ($\mu \pm \sigma$) values of the training and test qualities in the first and second line, respectively.

Similarity Threshold	Poly-10	Pagie-1	Friedman-2	Tower
90%	0.9906 ± 0.0316	0.9676 ± 0.0215	0.8430 ± 0.0415	0.8777 ± 0.0072
	0.9714 ± 0.1356	0.8921 ± 0.2063	0.7825 ± 0.1073	0.8785 ± 0.0095
95%	0.9947 ± 0.0175	0.9666 ± 0.0199	0.8439 ± 0.0412	0.8789 ± 0.0067
	0.9928 ± 0.0235	0.8483 ± 0.2478	0.7989 ± 0.1038	0.8799 ± 0.0080

(a) Minimum Schema Length = 10

Similarity Threshold	Poly-10	Pagie-1	Friedman-2	Tower
90%	0.9809 ± 0.0605	0.9688 ± 0.0214	0.8668 ± 0.0395	0.8770 ± 0.0057
	0.9568 ± 0.1437	0.8848 ± 0.2289	0.8367 ± 0.0824	0.8776 ± 0.0074
95%	0.9848 ± 0.0292	0.9648 ± 0.0226	0.8596 ± 0.0420	0.8785 ± 0.0080
	0.9804 ± 0.0376	0.8799 ± 0.2079	0.8191 ± 0.1212	0.8789 ± 0.0106

(b) Minimum Schema Length = 25

Table 6.18: OSGP-S Fixed Replacement Ratio 90%. The results are expressed as ($\mu \pm \sigma$) values of the training and test qualities in the first and second line, respectively.

Looking at the differences in quality, we may conclude that the restriction on the minimum schema size acts as a compensating force for the high value of the fixed replacement ratio, preventing the diversification phase from affecting algorithm convergence.

The differences in solution quality between OSGP-S-Adaptive and OSGP-S-Fixed suggest that it is beneficial to adjust the mutation intensity based on the relative population frequencies of the given schemas. Setting a high fixed replacement ratio from the beginning may in some cases

6.3.3 Comparison with OSGP

The OSGP-S methodology (Tables 6.16 to 6.18) performs clearly better in comparison with OSGP (Table 6.10). However, Tables 7.19, 7.20a and 7.20b (in Appendix, see Section 7.3) show that the OSGP-S runs consisted of a significantly higher number of generations, amounting to a significantly higher effort in terms of solution evaluations. The question arises whether or not the superior performance can be explained by the extra effort or by the increased efficiency of the search due to the schema-based diversification strategy. To test this hypothesis, we repeated the OSGP experiment without imposing any restrictions on the active selection pressure values. A simple termination criteria set to a maximum number of five million evaluations was used instead.

The results in Table 6.19 show that OSGP with five million evaluations (abbreviated as OSGP-5M) does not produce the same overall level of performance:

- Compared to OSGP-S-Adaptive, OSGP-5M shows inferior solution qualities on the training data for all problems except the Tower problem. Qualities on the test data also show that the algorithm tends to produce overfit solutions as a result of loss of diversity.
- Compared to OSGP-S-Fixed, OSGP-5M is able to produce better solutions for the Friedman-2 problem when the minimum schema length for OSGP-S-Fixed is set to 10 nodes. At the same time, OSGP-5M produced significantly worse qualities on the Poly-10 and Pagie-1 problems, with poor generalization capabilities as a result of overfitting.

6.3.4 Overview of the Results

A non-exhaustive experiment on the diversification strategy parameter space was performed using different values for the minimum schema length, minimum phenotypic similarity, replacement ratio, replacement rule and schema matching parameters.

The results have shown that the schema-based diversification strategy introduced in Section 5.2.5 can effectively promote population evolvability by preventing the degradation of the search to narrow regions of the genotype space which promote essentially the same

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	0.8826 ± 0.1530	0.9611 ± 0.0247	0.8796 ± 0.0385	0.8843 ± 0.0091
	0.7843 ± 0.2937	0.7284 ± 0.3026	0.8343 ± 0.1255	0.8824 ± 0.0100
Gender Specific	0.9301 ± 0.1000	0.9603 ± 0.0245	0.8858 ± 0.0389	0.8872 ± 0.0083
	0.8836 ± 0.1547	0.7780 ± 0.2820	0.8325 ± 0.1671	0.8779 ± 0.0485

Table 6.19: OSGP Average Solution Qualities - 5M Evaluations. The results are expressed as $(\mu \pm \sigma)$ values of the training and test qualities in the first and second line, respectively.

phenotype. The comparison with the OSGP algorithm using the same number of evaluated solutions has proven the practical benefits of the diversification concept.

The variations in quality for each OSGP-S configuration and each test problem highlight the fact that each fitness landscape has different characteristics and may be exploited more efficiently by slightly different algorithms. In this regard, schema diversification provides a useful means to fine-tune the search and perhaps identify the particularities of each problem.

The superior qualities obtained by OSGP-S-Adaptive show the benefits of adaptive mutation rates targeted at locally converged clusters of individuals which are highly similar in both genotype and phenotype. Allowing the targeted mutation step to bypass the offspring selection step (by acting on the population itself after the recombination phase is completed) increases the population's adaptive potential by ensuring that new genetic material is present in the population. On the downside, it is likely that the newly-introduced genetic diversity is also accompanied by lower-than-average quality, reducing the mutants' chances of getting picked by selection.

Overall, we have shown a new idea for promoting population evolvability, implemented as a prototype algorithm which performed better on all test problems. Further improvements to the methodology are likely possible either during the schema generation step (detecting locally converged clusters) or at the genotype level where different ways of introducing genetic diversity while maintaining fitness could be investigated.

7 Final Remarks

7.1 Overview

The main topic of this work is the analysis of genetic programming evolutionary dynamics from a systems perspective, with a focus on genotype-phenotype relations and their influence on the fundamental properties of complex evolving systems: robustness and evolvability. We investigate these properties by looking at the inheritance patterns in the population, shaped by the interplay between the selection and recombination operators.

With these goals in mind, this thesis represents both a work of synthesis and an original contribution. The synthesis part consisting of the first four chapters aims to provide the necessary context for our endeavour:

- ◇ It provides a theoretical background for optimization problems and the ways they may be solved using algorithms, heuristics or metaheuristics. We begin the discussion with the introduction of basic concepts pertaining to algorithms and computational complexity theory, and explain why it is useful in many cases to look for approximate solutions (instead of exact ones).

- ◇ It explains the idea of metaheuristics as practical solving methods inspired by metaphor and fueled by randomness. We provide an overview of randomized algorithms and probabilistic methods, and a taxonomy of metaheuristics where we describe each main category.

- ◇ It discusses at length the implications of no-free-lunch (NFL) theorems on the search properties of metaheuristics in general and evolutionary search in particular. We conclude that the NFL theorem does not apply to many real-world problems where the objective function is not closed under permutation. Concerning genetic algorithms, the sharpened NFL theorem shows that global convergence is possible when elitism is employed, and that the average number of fitness evaluations (ie., the effort needed to arrive at the solution) is not greatly affected by neutrality.

- ◇ It introduces the reader to the basic vocabulary of evolution, consisting of notions and ideas (defined in the glossary) from the field of biology. We believe that an understanding of the biological principles of evolution represents an essential requirement for every computer scientist dealing with evolutionary computation. The biological perspective highlights the importance of a systems approach focused on the

developmental aspect of genetics and the non-linear, complex relationship between phenotypic and genotypic variation.

◇ It provides a general overview of evolutionary algorithms, and a comprehensive treatment of genetic algorithms (GA) and genetic programming (GP). We highlight the major impact of selection and recombination mechanisms on the genetic structure of the population and implicitly, on the ability to produce adaptive change.

◇ In the context of GP, we outline the principles of good representations (reflected on the choice of primitive set) and we discuss the particularities of tree creation, selection, crossover and mutation.

We explain harmful phenomena in GP like the loss of population diversity and code bloat as consequences of the interplay between selection at the phenotype level and the variation-producing operators acting on genotypes. The main responsible operators (crossover and mutation) are discussed in detail along with an extensive literature survey.

◇ We discuss schema theorems for genetic algorithms and genetic programming and adopt Poli and Langdon's schema definition for the development of our own methodology for schema-based analysis of GP dynamics.

◇ We explain the fundamental properties of complex evolving systems: evolvability and robustness. In this context, fitness landscapes and genotype-phenotype (G→P) maps represent powerful conceptual tools. Patterns of phenotypic development – determined by the intrinsic properties of the G→P map – are subtly influenced by epistatic effects and complex interactions within gene networks.

Since gene expression itself is under epigenetic control, robustness and evolvability represent emergent properties. Emergence results from the causal loop between the mechanism for phenotypic selection (top-down causation) and those for genotypic variation (bottom-up causation). Robustness implies redundancy (neutral networks) in genotype space, evolvability depends on the ability to walk across these networks towards points with higher adaptive potential.

We argue that current attempts to improve GP performance are in fact attempts to improve the system's potential to evolve. It is therefore essential to understand the developmental aspect of adaptation and the underlying mechanisms for adaptive change. For example, genetic robustness can evolve by two main mechanisms: buffering and modularity – both conferring phenotypes a selective advantage. Buffering, through the accumulation of hidden (cryptical) genetic variation leads to a size increase in the genotype and the occurrence of bloat, but at the same time protects phenotypes against deleterious genotype changes and acts as an evolutionary capacitance. Modularity is another way of maintaining phenotypic function against perturbation, as genotypes organised in a network of autonomous modules are less

likely to change their phenotypic expression when perturbed. Despite decreasing phenotypic variability, robustness is a prerequisite for evolvability.

Applying the developmental paradigm to GP makes us realize the importance of optimizing the design of artificial genetic operators with both robustness and evolvability in mind. At the same time, the study of existing operators should take into account the aspects that have an impact on evolvability, such as the frequency of deleterious changes due to crossover and mutation, the rate of diversity loss, the occurrence of bloat and the emergence of modularity (ie., building blocks) in the population.

7.2 Main Contribution

Our main contribution described in the second part of this thesis (starting with Chapter 5) consists of a methodology for the analysis of population genealogies and inheritance patterns within GP. For a more complete picture, a set of metrics for the analysis of population diversity at the genotypic and phenotypic level were developed to go along with the genealogy analysis. The main points are described below.

Population Genealogy Graphs

- ◇ We represent genealogies as directed graphs in which vertices represent individuals and arcs represent hereditary relationships.
- ◇ The information regarding the inheritance of genotype fragments from parent to child is embedded directly into the genealogy graph, through instrumentation of the crossover and mutation genotypic operators. This includes the positions of the replaced subtree in the root parent and the replacement subtree in the non-root parent, expressed as node indices in the prefix node ordering.
- ◇ Lineage continuity is preserved by also including in the genealogy graph the intermediate offspring created when crossover and mutation are applied in succession. This allows access to an individual's full set of ancestors or descendants, as well as its root lineage showing how the initial structure from generation zero was altered by genetic operations in the course of evolution.
- ◇ We introduce new measurements for the analysis of evolutionary dynamics, based on the parent-child hereditary relationships in the genealogy graph: average fragment length, genetic operator improvement or selection ratio.

Analysis of Population Dynamics

◇ We empirically show that GP systems suffer from low robustness as the majority of genotypic changes have deleterious effects on fitness. This leads to loss of diversity due to selection discarding the affected genotypes, reducing each generation the available pool of genetic material and the potential to produce new adaptive variation.

Attempts to increase robustness by allowing for additional genetic buffering (for example, by increasing the maximum tree length limit) would have the counter effect of reducing the quality of solutions in terms of interpretability and generalization ability, as bloat encourages overfit solutions.

In the case of offspring selection, the measurements of crossover and mutation improvement change their interpretation to indicate how easy it is for the GP system as a whole to produce adaptive change. We show that in this scenario the average improvement is minimal, suggesting that OSGP's advantage over the standard version of the algorithm results simply from avoiding the effort of dealing with deleterious changes, and not from significant improvements in terms of evolvability.

◇ We empirically investigate the rate of diversity loss for different selection schemes using the selection ratio measure (the percentage of individuals that get selected for reproduction). We show that selection ratio varies with problem instance for proportional selection and remains the same for tournament selection.

◇ We use the crossover bias theory to explain the evolution of average tree size and average crossover size as a result of the interaction between selection and crossover.

We explain the counter-intuitive relationship between average tree size and average crossover fragment size observed in the SGP experiments. We show that high selection pressure skews the distribution of tree sizes towards the size limit, which in turn imposes a bias on the crossover operator towards smaller subtrees (so that the resulting children do not exceed the maximum tree size). This bias, reflected on the choice of the second crossover point (from the non-root parent) implies a deeper relationship between crossover and selection with potentially interesting ramifications.

◇ We introduce new genotype and phenotype distance measures for the analysis of population diversity. Genotype similarity is calculated using the bottom-up tree distance, with a runtime complexity linear in the size of the trees. Phenotypic similarity is calculated using the Pearson R^2 correlation coefficient. Both measures express the degree of similarity between two individuals in the interval [0, 1], where 1 means completely similar and 0 means completely dissimilar. By this convention, similarity and diversity become complementary notions conveying the same information about the population.

◇ We show empirically that both methods represent viable instruments for the analysis of GP diversity. As expected, higher selection pressure leads to higher similarity (both genotypic and phenotypic) between solutions. Average similarity increases in the beginning of the run and stabilizes to a value depending on the actual selection pressure applied to the population.

An interesting aspect is revealed by the similarity measurements on the SGP algorithm, where proportional selection leads to lower genotypic similarity and higher phenotypic similarity, while the opposite is true for tournament selection. This suggests that structural similarity does not imply similar semantics. In terms of G→P map properties, this means that the genotype space is marked by transformational boundaries where small changes determine transitions from one phenotypic state to another.

This observation does not hold for the OSGP algorithm, where the strict offspring selection drives the population not only to higher levels of genotypic similarity (compared to SGP), but also to highly similar semantics. Phenotypic similarity increased to be asymptotically close to 1 on all tested problems and algorithmic configurations.

Tracing of Evolutionary Trajectories

◇ We introduce a new methodology and algorithm for tracing genotypic changes in the population using the inheritance information embedded in the genealogy graph.

◇ The algorithm traverses an individual's genealogy searching for genotypic changes with respect to a specified subtree to be traced. The result is a subset of the genealogy graph in which only the relevant operations (that contributed to the structure of the traced subtree) are recorded. This new graph called a *trace graph*, represents both the sequence of relevant genotypic operations and the ancestors individuals which those operations were applied on.

◇ We define the contribution ratio measure as the ratio between the size of an individual's trace graph (when tracing the root subtree of the tree individual) and the size of its ancestry. This measure has the interpretation of effort expended by the algorithm to produce solutions. A high contribution ratio implies frequent adaptive changes across an individual's ancestry, corresponding to incremental improvement. Since the changes are adaptive (they improve fitness), they are maintained by selection and present in the trace graph.

On the other hand, a low contribution ratio corresponds to neutral or deleterious changes which are not fixated by selection and may get rejected or simply drift away.

The analogy with effort is the following: when changes are easy to come by, we say that they are obtained by the evolutionary process with little effort. This corresponds

to a high contribution ratio as described above. But when it becomes difficult to produce adaptive changes – as the results show in SGP that the vast majority of genotypic changes are deleterious, we consider that they are obtained with considerably more effort. This is reflected by small values of the contribution ratio as shown by the experimental results.

The negative influence of selection pressure on the contribution ratio can be explained by the fact that higher selection pressure leads to fewer unique ancestors and smaller trace graphs where fit ancestors contribute multiple genetic fragments.

◇ We demonstrate how the tracing methodology can be used to identify the most sampled subtrees when applied on the whole population. In this case, the algorithm uses a caching mechanism to generate a complete trace graph of all the individuals and increment a weight value associated with every subtree. The weight difference between parent and child tree nodes represents the effective sample count for the given subtree.

The methodology is empirically validated by a) the correlation between the average subtree weight and average parent distribution curves and b) the high degree of correspondence between the most sampled subtrees (as identified by the algorithm) and the actual formula terms of the problem test function.

This allows us to identify the genetic building blocks used by the algorithm to solve different artificial or real-world regression problems. The main benefit of this method is that it is general enough to work on any regression problem, extracting information from GP's implicit ability of finding patterns in data.

Identification of Common Schemas

◇ Schema theorems represent important theoretical instruments for the analysis of GP dynamics. In this work we adopt Poli and Langdon's schema definition and use genealogy information to generate concrete schema instances as described in Section 5.2.4.

◇ The practical investigation of GP schemas faces the computational challenge of using tree pattern matching algorithms to compute schema frequencies. We overcome this challenge by employing an algorithm by (Götz et al., 2009) from the area of query processing and database programming (adapted and implemented in HeuristicLab) to match a population of trees against generated schemas.

◇ Our empirical investigation shows that the most frequent schemas generated by our approach are of above-average fitness and can be said to constitute in many cases the underlying genotypic pattern of the population (matching a high proportion of individuals). In this respect, schema-based analysis provides a more accurate

description of genotypic diversity in the population, compared to the bottom-up distance similarity metric.

◇ The results also confirm our intuition that individuals belonging to the same schema are of high (above-average) genotype and phenotype similarity. Therefore, the information about schema frequency and diversity provides a good indication of local convergence.

Using Schemas to Improve the Search

◇ We propose a new strategy for tuning the "exploration versus exploitation" aspect of the search by introducing localized mutation within locally converged clusters of individuals, guided by the information provided by the schema analysis methods described above.

A cluster of individuals is considered locally converged if it belongs to the same schema and shares a minimum degree of phenotypic similarity. The exploration aspect is improved by applying differential mutation rates within such clusters depending on cluster size. Intuitively, this helps the search by redistributing the clustered points more evenly across the search space.

We call this approach a "schema-based diversification strategy", and we argue that it provides all the necessary requirements for improving population evolvability:

1) It is *hereditary*, as it uses the genealogy graph for schema generation
2) It is *structural*, as it considers individuals belonging to common schemas
3) It is *semantic*, as it considers phenotypically-similar individuals
4) It is *adaptive*, as differential mutation rates can be applied within clusters

◇ We extended the OSGP algorithm to use the schema-based diversification approach and called the new variant OSGP-S (OSGP with Schemas). The diversification step is completely tunable through a series of parameters which control every aspect of its behavior such as the minimum schema length and frequency, minimum phenotypic similarity or the rules for applying differential mutation.

As mutation tends to damage individual quality, OSGP-S was used in combination with the gender-specific selection mechanism in order to give mutated individuals a fair chance to compete with the rest of the population. This represents a weak point of the algorithm which will be addressed in future research. We expect that improvements are possible to the initial proof of concept version of the algorithm either in the strategy itself or through better yet undiscovered parameterizations.

- We compared the performance of the new method against the OSGP algorithm on the four test problems: Poly-10, Pagie-1, Friedman-2 and Tower. The results show that OSGP-S outperforms OSGP on all test problems, producing solutions with better training quality, less overfitting and better generalization ability. The algorithm performs particularly well on problems where adaptation is more likely to be achieved through genotypic innovation rather than parameter tuning.

7.3 Future Research Ideas

This work demonstrates a set of novel techniques for the analysis – and improvement – of genetic programming. These techniques are inspired by theoretically-sound principles, such as preserving an optimal balance between exploration and exploitation and improving population evolvability.

The proposed schema analysis methodology represents a particularly promising approach for further theoretical and practical advancements in the field. On the one hand, it allows a more in-depth investigation of existing schema theory, and on the other hand, it provides a sound mechanism for the adaptive tuning of selection pressure and mutation rates.

Future enhancements of the OSGP-S algorithm should be achievable by improving the survivability rate of mutated individuals (for example, allowing them to evolve in separate demes), or by producing more accurate schemas from the available genealogical information. The tracing methodology comes to mind here as a method to generate potentially more accurate schemas.

Another promising research direction is in the area of robustness and genetic operator effectiveness. It has been shown that GP genotypes are generally not robust enough, leading to increased rates of diversity loss. Additionally, the search properties of the algorithm (eg., size of genetic fragments) are more subtly influenced by the imposed size limits than previously thought, through the interaction of crossover and selection. This represents a good basis for further research into more effective and more robust genetic operators.

Finally, the methodology developed in this thesis can be used to extend the analysis to other flavors of GP such as ALPS-GP (Hornby, 2006).

Appendix

Generation	Schema
1	(- (- (- -7.49 17.04) 0.12_X7) (- 0.52_X4 (* 14.79 #)))
10	(* 0.29_X1 (/ 0.54_X2 (+ (- # #) 9.58)))
20	(+ (- 13.93 (* 17.23 (* 2.61_X2 1.93_X1))) #)
30	(+ (- 13.93 (* 17.23 (* 2.61_X2 1.93_X1))) #)
40	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
50	(+ # (+ (* # 1.93_X1) (* 2.61_X2 1.93_X1)))
60	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
70	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
80	(+ # (+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #))
90	(+ (+ (* 2.61_X2 1.93_X1) #) (+ # #))
100	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
110	(+ (+ # #) (+ (* 2.50_X5 1.91_X6) #))
120	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
130	(+ (+ # (* 2.61_X2 1.93_X1)) (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)))
140	(+ # (+ # (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1))))
150	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
160	(+ # (+ (+ # (* 2.61_X2 1.93_X1)) (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1))))
170	(+ (+ (+ # (* 2.49_X4 1.99_X3)) #) #)
180	(+ # (+ (* 2.50_X5 1.91_X6) (+ # #)))
190	(+ (+ (+ (* 2.61_X2 1.93_X1) #) #) (* 2.61_X2 1.93_X1))
200	(+ (+ # (* 2.50_X5 1.91_X6)) (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)))
210	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
220	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
230	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #)
240	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
250	(+ (+ (+ (* 2.61_X2 1.93_X1) #) #) #)
260	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #)
270	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) (+ (* 2.61_X2 1.93_X1) #))
280	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.61_X2 1.93_X1)))
290	(+ (+ (+ (* 2.61_X2 1.93_X1) #) #) #)
300	(+ # (+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) (* 2.49_X4 1.99_X3)))
310	(+ (+ (* 2.49_X4 1.99_X3) (+ (* 2.50_X5 1.91_X6) #)) #)
320	(+ (+ (+ (* 2.49_X4 1.99_X3) #) #) #)
330	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) (+ (* 2.50_X5 1.91_X6) #))
340	(+ # (+ # (+ (* 2.61_X2 1.93_X1) (* 2.50_X5 1.91_X6))))
350	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
360	(+ (+ # #) (+ (* 2.49_X4 1.99_X3) #))
370	(+ (+ # (+ # #)) (* 2.50_X5 1.91_X6))
380	(+ (* 2.50_X5 1.91_X6) (+ # (* 2.50_X5 1.91_X6)))
390	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
400	(+ # (+ # (+ (* 2.50_X5 1.91_X6) (+ # #))))
410	(+ (+ (* 2.50_X5 1.91_X6) (+ (* 2.61_X2 1.93_X1) #)) (+ (* 2.50_X5 1.91_X6) #))
420	(+ (+ (+ # (* 2.50_X5 1.91_X6)) #) #)
430	(+ (+ (* 2.49_X4 1.99_X3) (* 2.49_X4 1.99_X3)) (+ # #))
440	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #)
450	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
460	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #)
470	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #)
480	(+ # (+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #))
490	(+ (+ (+ (* 2.50_X5 1.91_X6) #) (* 2.61_X2 1.93_X1)) (+ # (* 2.61_X2 1.93_X1)))
500	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))

Table 7.1: SGP Poly-10 Proportional Most Frequent Schemas

Generation	Schema
1	(- (- (- -7.49 17.04) 0.12_X7) (- 0.52_X4 (* 14.79 #)))
10	(* 0.29_X1 (/ 0.54_X2 (+ (- # #) 9.58)))
20	(+ (- 13.93 (* 17.23 (* 2.61_X2 1.93_X1))) #)
30	(+ (- 13.93 (* 17.23 (* 2.61_X2 1.93_X1))) #)
40	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
50	(+ # (+ (* # 1.93_X1) (* 2.61_X2 1.93_X1)))
60	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
70	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
80	(+ # (+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #))
90	(+ (+ (* 2.61_X2 1.93_X1) #) (+ # #))
100	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
110	(+ (+ # #) (+ (* 2.50_X5 1.91_X6) #))
120	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
130	(+ (+ # (* 2.61_X2 1.93_X1)) (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)))
140	(+ # (+ # (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1))))
150	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
160	(+ # (+ (+ # (* 2.61_X2 1.93_X1)) (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1))))
170	(+ (+ (+ # (* 2.49_X4 1.99_X3)) #) #)
180	(+ # (+ (* 2.50_X5 1.91_X6) (+ # #)))
190	(+ (+ (+ (* 2.61_X2 1.93_X1) #) #) (* 2.61_X2 1.93_X1))
200	(+ (+ # (* 2.50_X5 1.91_X6)) (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)))
210	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
220	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
230	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #)
240	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
250	(+ (+ (+ (* 2.61_X2 1.93_X1) #) #) #)
260	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #)
270	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) (+ (* 2.61_X2 1.93_X1) #))
280	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.61_X2 1.93_X1)))
290	(+ (+ (+ (* 2.61_X2 1.93_X1) #) #) #)
300	(+ # (+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) (* 2.49_X4 1.99_X3)))
310	(+ (+ (* 2.49_X4 1.99_X3) (+ (* 2.50_X5 1.91_X6) #)) #)
320	(+ (+ (+ (* 2.49_X4 1.99_X3) #) #) #)
330	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) (+ (* 2.50_X5 1.91_X6) #))
340	(+ # (+ # (+ (* 2.61_X2 1.93_X1) (* 2.50_X5 1.91_X6))))
350	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
360	(+ (+ # #) (+ (* 2.49_X4 1.99_X3) #))
370	(+ (+ # (+ # #)) (* 2.50_X5 1.91_X6))
380	(+ (* 2.50_X5 1.91_X6) (+ # (* 2.50_X5 1.91_X6)))
390	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))
400	(+ # (+ # (+ (* 2.50_X5 1.91_X6) (+ # #))))
410	(+ (+ (* 2.50_X5 1.91_X6) (+ (* 2.61_X2 1.93_X1) #)) (+ (* 2.50_X5 1.91_X6) #))
420	(+ (+ (+ # (* 2.50_X5 1.91_X6)) #) #)
430	(+ (+ (* 2.49_X4 1.99_X3) (* 2.49_X4 1.99_X3)) (+ # #))
440	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #)
450	(+ (+ (* 2.61_X2 1.93_X1) (* 2.61_X2 1.93_X1)) #)
460	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #)
470	(+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #)
480	(+ # (+ (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)) #))
490	(+ (+ (+ (* 2.50_X5 1.91_X6) #) (* 2.61_X2 1.93_X1)) (+ # (* 2.61_X2 1.93_X1)))
500	(+ # (+ (* 2.50_X5 1.91_X6) (* 2.50_X5 1.91_X6)))

Table 7.2: SGP Poly-10 Tournament Most Frequent Schemas

Generation	Schema
1	(/ 0.68_Y (/ (/ (/ # #) #) #))
10	(+ (/ 11.97 (* -0.06_X #)) (+ # #))
20	(+ (- -15.84 (/ 12.10 (* 1.81_Y 0.23_Y))) #)
30	(+ (- -15.84 (/ 12.10 (* 1.81_Y 0.23_Y))) #)
40	(+ (/ 11.97 (* -0.06_X (/ (+ 4.66 (* (* 1.41_X (* 0.71_Y 1.73_Y)) 1.48_X)) (- (* 0.25_X #) 0.72_X)))) #)
50	(+ # (/ 11.97 (+ 4.66 (* (* 1.41_X (* 1.81_Y 0.23_Y)) 1.48_X))))
60	(+ # (/ 11.97 (+ 4.66 (* # 1.48_X))))
70	(+ # (/ 11.97 (+ 4.66 (* (* 1.41_X #) 1.48_X))))
80	(+ # (/ 11.97 (+ 4.66 (* # 1.48_X))))
90	(+ # (/ 11.97 (+ 4.66 (* # 1.48_X))))
100	(+ # (/ 11.97 (+ 4.66 (* # 1.48_X))))
110	(+ (/ 11.97 (+ 4.66 #)) (/ 11.97 (+ 4.66 (* # #))))
120	(+ # (/ 11.97 (+ 4.66 (* # 1.48_X))))
130	(+ # (/ 11.97 (+ 4.66 (* # 1.48_X))))
140	(+ # (/ 10.44 (+ 4.68 (* # 1.48_X))))
150	(+ # (/ 10.44 (+ 4.68 (* (* # (* 1.87_Y 0.21_Y)) 1.48_X))))
160	(+ # (/ 12.04 (+ 4.62 (* (* 1.41_Y (* 1.84_X 0.23_Y)) 1.39_X))))
170	(+ # (/ 12.04 (+ 4.62 (* # 1.39_X))))
180	(+ # (/ 12.04 (+ 4.62 (* # 1.39_X))))
190	(+ # (/ 11.35 (+ 5.06 (* # 1.42_Y))))
200	(+ # (/ 11.04 (+ 4.30 (* (* 1.39_Y (* 2.01_Y 0.23_X)) 1.47_X))))
210	(+ (/ (/ 0.01_X #) #) (/ 11.04 (+ 4.30 #)))
220	(- (/ # #) (/ 8.95 (+ 3.82 (* (* 1.54_X (* 2.04_Y 0.23_X)) 1.47_Y))))
230	(+ (/ 0.00_Y #) (/ 10.63 (+ 4.31 #)))
240	(- # (/ 10.54 (+ # (* # 1.54_Y))))
250	(+ (/ 0.00_Y #) (/ 10.63 (+ 4.31 #)))
260	(+ (/ 0.01_X #) (/ 9.27 (+ 4.31 (* # 1.54_Y))))
270	(+ # (/ 9.27 (+ 5.35 (* # 1.57_X))))
280	(+ (/ 0.01_X (- # #)) (/ 9.27 #))
290	(+ # (/ 9.27 (+ 5.35 (* (* 2.05_Y #) 1.57_X))))
300	(+ # (/ 11.35 (+ 4.62 (* (* 1.43_Y (* 1.91_X 0.23_Y)) 1.39_X))))
310	(+ # (/ 11.35 (+ 4.62 (* (* 1.43_Y #) 1.39_X))))
320	(+ # (/ 11.35 (+ 4.62 (* (* 1.43_Y (* 1.91_X 0.23_Y)) 1.39_X))))
330	(+ # (/ 9.27 (+ 5.22 (* (* 1.52_Y (* 1.76_X 0.24_X)) 1.64_Y))))
340	(+ # (/ 9.27 (+ 5.35 (* (* 1.48_Y #) 1.63_Y))))
350	(+ # (/ 9.27 (+ 6.71 (* (* 1.52_Y (* 1.76_X 0.24_X)) 1.64_Y))))
360	(+ # (/ 11.30 (+ 4.73 (* (* 1.39_Y (* 1.30_X 0.25_Y)) 1.98_X))))
370	(+ (/ 0.00_Y #) (/ 10.32 (+ 6.77 #)))
380	(+ # (/ 9.27 (+ 5.92 (* # 1.50_X))))
390	(+ (/ 0.00_Y #) (/ 10.73 (+ 3.79 #)))
400	(+ (/ 10.01 #) (/ 10.01 (+ 4.92 (* (* 1.56_Y #) 1.52_X))))
410	(+ (/ 0.00_X #) (/ 10.01 (+ 4.92 #)))
420	(+ # (/ 9.76 (+ 3.96 (* # 1.46_X))))
430	(+ # (/ 10.08 (+ 4.39 (* (* 1.43_X #) 1.48_Y))))
440	(+ # (/ 10.08 (+ 4.39 (* (* 1.43_X (* 1.75_X 0.21_Y)) 1.48_Y))))
450	(+ (/ 0.00_Y #) (/ -14.36 (+ 4.72 #)))
460	(+ (/ 0.00_Y #) (/ -14.36 (+ 4.72 (* (* 1.38_Y #) 1.95_X))))
470	(+ (/ 0.01_Y #) (/ 10.10 (+ 3.13 #)))
480	(+ # (/ 10.02 (+ 4.54 (* # 1.48_Y))))
490	(+ (/ 0.01_X #) (/ 10.02 (+ 6.60 #)))
500	(+ (/ 0.01_Y (+ # #)) (/ 10.10 #))

Table 7.3: SGP Pagie-1 Proportional Most Frequent Schemas

Generation	Schema
1	(+ 1.00_X (+ (- -5.78 (+ (* # #) -12.74)) #))
10	(+ 1.00_X (+ # (* -0.23_Y (- (* (* (* 2.55_Y 0.62_X) 3.92_X) 16.32) #))))
20	(+ (- # (/ (- 7.52 -18.25) (* 0.79_Y 1.66_Y))) (* (/ -5.08 -20.36) (/ -5.08 -20.36)))
30	(+ (- # (/ # (* 0.79_Y 1.66_Y))) #)
40	(+ (- # (/ (* -18.62 -3.67) (* 0.79_Y 1.66_Y))) #)
50	(+ (- # (/ # (* 0.79_Y 1.66_Y))) (* # #))
60	(+ (- # (/ # (* 0.79_Y 1.66_Y))) #)
70	(+ (- (/ -10.66 (* -0.38_X -0.58_X)) #) #)
80	(+ # (+ # (* # (* 2.43_X 0.61_X))))
90	(+ (- # (/ (* -18.62 -3.67) (* 0.78_Y 1.66_Y))) #)
100	(+ # (* (* 0.79_Y 1.66_Y) (* 2.43_X 0.64_X)))
110	(+ # (* (* 0.85_Y #) (* 2.43_X 0.61_X)))
120	(+ (- # #) (* (* 0.79_Y 1.66_Y) #))
130	(+ (- # (/ (* -18.62 -3.67) (* 0.79_Y 1.66_Y))) #)
140	(+ # (* (- # -2.29) (* 2.43_X 0.62_X)))
150	(+ (- # (/ # #)) (* # #))
160	(+ # (* (- # -2.29) (* 2.43_X 0.62_X)))
170	(+ (- (- # #) (/ # #)) (* # (* 2.43_X 0.62_X)))
180	(+ (- # (/ (* -18.62 -3.67) (* 0.79_Y 1.66_Y))) #)
190	(+ # (* (* 2.43_X 0.62_X) (* 2.43_X 0.62_X)))
200	(+ # (* (- (* 0.78_Y 1.61_Y) -2.29) #))
210	(+ (- # (/ # (* 0.79_Y 1.66_Y))) #)
220	(+ (- # (/ (* -19.21 -3.67) (* 0.79_Y 1.66_Y))) #)
230	(+ (- # #) (* # (* 2.43_X 0.62_X)))
240	(+ (- # (/ (* -18.62 -3.67) (* 0.79_Y 1.66_Y))) #)
250	(+ (- (- (+ (/ -11.06 #) #) #) (/ # #)) #)
260	(+ (- # (/ (* -18.62 -3.69) (* 0.79_Y 1.66_Y))) #)
270	(+ # (* (- # -11.06) (* 2.43_X 0.62_X)))
280	(+ (- # (/ # (* 0.79_Y 1.66_Y))) #)
290	(+ (- (- (+ # #) #) #) #)
300	(+ (- (+ # -0.59_Y) (/ (* -18.62 -2.88) (- (* # -0.58_X) -0.06))) #)
310	(+ # (- (- (* -0.36_X -0.53_X) #) #))
320	(+ # (- (- # (/ # (- (* -0.36_Y -0.59_Y) -0.06))) #))
330	(+ (- # #) (- # (* # -0.58_X)))
340	(+ (- # (/ (* -18.62 -2.88) (- (* -0.58_X -0.58_X) -0.06))) #)
350	(+ (- # (/ (* -18.62 -2.88) #)) #)
360	(+ (- (/ -0.58_X #) (/ (* -18.62 -2.88) (- (* (* (* -0.36_X -0.58_X) -0.58_X) -0.58_X) -0.06))) #)
370	(+ (- # (/ (* -18.62 -2.88) #)) #)
380	(+ (- # (/ (* -18.62 -2.88) (- (* (* (* -0.36_X -0.53_X) -0.58_X) -0.58_X) -0.06))) #)
390	(+ # (- (- (* -0.36_X #) #) #))
400	(+ (- (* -18.62 -2.88) (/ (* -18.62 -2.88) #)) #)
410	(+ (- (- (* -0.36_Y -0.36_Y) -0.06) (/ (* -18.62 -2.88) (- (* (* (* -0.36_X -0.53_X) -0.58_X) -0.58_X) -0.06))) #)
420	(+ (- # (/ (* -18.62 -2.88) (- # -0.06))) #)
430	(+ (- (* -0.58_X -0.53_X) (/ # #)) #)
440	(+ (- # (/ (* -18.62 -2.88) (- # -0.06))) #)
450	(+ (- # (/ (* -18.62 -2.88) (- (* (* (* -0.36_X -0.53_X) -0.58_X) -0.58_X) -0.06))) #)
460	(+ (- (* -0.35_Y -0.35_Y) (/ (* -18.62 -2.88) (- # -0.06))) #)
470	(+ (- (* -0.35_Y -0.35_Y) (/ (* -18.62 -2.88) (- (* # -0.58_X) -0.06))) #)
480	(+ (- (* -18.62 -2.88) (/ # (- (* # -0.58_X) -0.06))) #)
490	(+ (- # (/ (* -18.62 -2.88) (- (* (* (* -0.36_X -0.53_X) -0.58_X) -0.58_X) -0.06))) #)
500	(+ (- # (/ (* -18.62 -2.88) (- # -0.06))) #)

Table 7.4: SGP Pagie-1 Tournament Most Frequent Schemas

Generation	Schema
1	(* (/ (/ # (+ # 1.12_X4)) (- 17.61 17.55)) #)
10	(- (* # -7.32) (- (+ 1.92_X10 #) 3.17))
20	(- (- # (/ -14.87 -19.35)) (- # (* 0.90_X8 1.48_X4)))
30	(- 15.71 (+ # (+ (* (+ -16.13 0.61_X7) 2.14_X4) -0.04_X2)))
40	(* (- (/ (- 0.47_X1 (* 1.18 #)) 11.40) #) 0.20_X4)
50	(* (* (+ 1.75_X1 (+ 1.42_X5 -0.29_X4)) 1.56_X4) #)
60	(- (* 18.35 (* -19.26 #)) (+ # -4.59))
70	(+ 3.76_X4 (- (+ (+ 1.66_X1 17.24) 1.39_X2) #))
80	(* (+ 1.75_X1 1.39_X2) (+ (+ (+ # (- # 1.71_X4)) #) 2.12_X7))
90	(- (+ 18.06 2.34_X2) (- # (+ # -7.59)))
100	(+ # (* (+ 1.73_X4 (+ # 0.47_X4)) 0.83_X4))
110	(+ # (* (- (* (+ 1.75_X1 1.65_X5) 18.10) 1.97_X8) 0.83_X4))
120	(+ # (+ (+ 1.73_X4 1.39_X2) (+ 1.73_X4 1.39_X2)))
130	(+ (+ (+ 3.76_X4 -0.29_X4) (+ 1.73_X4 1.39_X2)) #)
140	(+ (+ (+ # 6.42) (+ 1.75_X1 (+ 1.75_X1 1.68_X2))) 1.39_X2)
150	(+ # (+ 19.66 (- (+ 1.73_X4 1.39_X2) -7.48)))
160	(+ 1.82_X2 (+ # (+ (+ 0.15_X10 (* 1.72_X4 1.37_X4)) 2.79_X1)))
170	(+ (+ (+ 1.75_X1 1.39_X2) (+ 1.75_X1 3.76_X4)) #)
180	(+ (+ # (+ (+ 1.39_X2 #) 3.76_X4)) (* 1.63_X4 1.28_X3))
190	(+ # (+ 0.15_X10 (+ (* 1.75_X1 1.39_X2) #)))
200	(+ (+ 1.75_X2 3.73_X4) (+ 0.30_X2 (+ 19.13 #)))
210	(+ (+ (+ 0.69_X1 (+ 3.76_X4 1.39_X2)) (+ # -3.00)) (+ 1.75_X1 (+ # 1.39_X9)))
220	(+ (+ # (+ 1.72_X5 (+ 1.69_X4 (+ 1.65_X5 (+ 1.75_X1 3.76_X4))))) #)
230	(+ (+ # (/ 1.75_X1 -11.08)) (+ 1.75_X1 11.97))
240	(+ (+ 1.65_X5 #) (+ (+ 1.73_X4 1.39_X2) #))
250	(+ (+ 1.65_X5 2.20_X5) (+ # (+ (+ # 17.24) -8.91)))
260	(+ 1.65_X5 (+ (+ 1.70_X1 (+ -17.07 (+ 0.04 1.39_X2))) #))
270	(+ (+ # (+ 3.76_X4 (+ 1.73_X4 1.39_X2))) #)
280	(+ # (+ -0.08_X8 (+ (+ # 1.39_X2) 17.17)))
290	(+ 17.58 (+ # (+ (+ (+ 3.76_X4 1.39_X2) 1.75_X1) 1.39_X2)))
300	(+ (+ 1.68_X5 2.29_X7) (+ (+ -1.14 2.76_X4) #))
310	(+ # (+ (+ # 1.75_X1) (+ 3.76_X4 1.39_X2)))
320	(+ 3.82_X5 (+ 1.73_X4 (+ (+ (+ 1.97_X4 #) #) 17.24) #)))
330	(+ (+ (+ (+ 2.32_X5 2.60_X2) (+ 1.75_X1 (+ 1.41_X2 3.07_X4))) 0.34_X5) #)
340	(+ 3.61_X4 (+ # (+ (+ 0.52 12.36) 1.91_X2)))
350	(+ (+ (+ 1.40_X2 (+ # 5.26)) (+ (+ 3.72_X4 0.54_X8) (+ # 2.04_X5))) 1.39_X2)
360	(+ -6.92 (+ (+ -4.63 1.55_X5) (+ # (+ 1.77_X1 1.82_X2))))
370	(+ # (+ (+ # (+ 1.27_X2 2.04_X5)) (+ (+ -6.75 1.64_X5) 2.04_X2)))
380	(+ 1.39_X2 (+ (+ # (+ (+ 1.81_X4 1.85_X1) 3.23_X6)) #))
390	(+ (+ 1.74_X5 (+ (+ 3.67_X1 1.39_X2) -0.77_X3)) #)
400	(+ # (+ 1.79_X4 (+ # (+ 1.76_X5 0.93_X10))))
410	(+ (+ (+ 2.76_X4 #) (+ (+ 1.73_X4 0.76_X1) 1.87_X1)) #)
420	(+ (+ (+ # 2.43_X4) -8.20) (+ # 0.95_X7))
430	(+ (+ 1.59_X3 (+ (+ 1.65_X5 (+ 0.89_X6 #)) 2.81_X1)) #)
440	(+ # (+ (+ # -7.66) (+ -0.63_X2 2.00_X2)))
450	(+ (+ 2.03_X2 (+ (+ (+ 1.75_X1 1.73_X4) #) 1.75_X1)) #)
460	(+ (+ (+ 1.75_X1 (+ 2.76_X4 1.33_X1)) 0.95_X9) #)
470	(+ (+ (+ 1.75_X4 -18.99) (+ 3.76_X4 1.75_X4)) #)
480	(+ (+ (+ 1.86_X2 (/ -19.29 0.68)) 1.21_X2) #)
490	(+ (+ -16.35 (+ 1.41_X2 (+ 1.73_X4 #))) #)
500	(+ (+ 0.46_X10 #) (+ (+ # (+ 2.87_X4 3.50_X4)) 1.68_X5))

Table 7.5: SGP Friedman-2 Proportional Most Frequent Schemas

Generation	Schema
1	(- -14.27 (* (* (* (+ 0.60_X6 #) (/ # (* (+ 1.75 (+ 2.34_X9 10.20)) (- 14.22 1.54_X1)))) 2.10) (+ (- # 2.09) -10.59)))
10	(- (* 12.02 (- (* 1.13_X4 #) #)) #)
20	(+ (- (+ 8.78 (* (+ 1.95_X5 2.85) 1.90_X1)) 2.38_X6) #)
30	(+ (/ # -18.67) (+ (* (- 6.99 (* 16.87 (+ (- # 0.68_X2) 1.89_X2))) (+ 1.95_X5 2.85)) #))
40	(+ # (+ (* (- 6.99 (* 16.87 (+ (- (+ 1.25_X1 1.50_X4) 0.68_X2) 1.89_X2))) (+ 1.95_X5 4.00)) #))
50	(+ # (+ # (* 2.56_X1 (* # 1.84_X1))))
60	(+ # (* (+ 8.78 (* (+ 1.95_X5 2.85) 1.90_X1)) #))
70	(+ # (* (+ 8.78 (* (+ 1.95_X5 2.85) 1.90_X1)) #))
80	(+ (* 16.87 #) (* (+ 8.78 #) #))
90	(+ # (* (+ 8.78 (* (+ 1.95_X5 2.85) 1.90_X1)) (- (* # 0.65_X2) (* (+ 1.78_X2 (+ 2.48_X4 0.53_X5)) #))))
100	(+ (+ 1.78_X2 #) (* (+ 8.78 (* (+ 1.95_X5 2.85) 1.90_X1)) #))
110	(+ # (* (+ 8.78 (+ 1.50_X4 0.53_X5)) #))
120	(+ # (* (+ 8.78 #) (- (* 4.00 0.65_X2) #)))
130	(+ (+ 2.48_X4 0.53_X5) (* (+ 8.78 (+ # 2.85)) #))
140	(+ # (* (+ 8.78 (+ (+ 1.95_X5 2.85) 8.78)) #))
150	(+ (+ 1.50_X4 0.53_X5) (* (+ 8.78 (+ # 2.85)) (- # # #)))
160	(+ # (* (+ 8.78 0.53_X5) (- # # #)))
170	(+ (+ (+ (* (+ 1.50_X4 0.53_X5) 1.84_X1) 1.89_X2) #) #)
180	(+ (+ (* 1.89_X2 1.90_X1) (* 1.89_X2 1.90_X1)) #)
190	(- (* (+ (* 2.56_X1 1.81_X2) (+ 1.50_X4 0.53_X5)) (- # 14.65)) #)
200	(+ (+ (+ 1.50_X4 0.53_X5) (+ 1.50_X4 0.53_X5)) #)
210	(+ (+ (+ 8.78 0.53_X5) 0.53_X5) (- # # #))
220	(+ (+ 8.78 #) (- # (- # # #)))
230	(+ (+ # 0.53_X5) (- (+ 1.74_X2 #) (- (* (+ (* 2.56_X1 1.81_X2) (+ 1.50_X4 0.53_X5)) #) #)))
240	(+ (+ # 0.53_X5) (- # (- (* (+ # (+ 1.50_X4 0.53_X5)) (- (* (* 2.56_X1 1.81_X2) (* 1.89_X2 1.84_X1)) 14.65)) #)))
250	(+ # (- (+ 1.74_X2 #) (- (* (+ (* 2.56_X1 1.81_X2) #) (- (* (* 2.56_X1 1.81_X2) (* 1.89_X2 1.84_X1)) 14.65)) (* 1.78_X3 1.97_X3))))
260	(+ (+ (+ 1.50_X4 0.53_X5) (+ 1.50_X4 0.53_X5)) #)
270	(+ # (- # (- (* (+ (* 2.56_X1 1.81_X2) (+ 1.50_X4 0.53_X5)) (- (* (* 2.56_X1 1.81_X2) (* 1.89_X2 1.84_X1)) 14.65)) (* 1.78_X3 1.97_X3))))
280	(+ (+ (+ 1.50_X4 0.53_X5) (+ 1.50_X4 0.53_X5)) #)
290	(+ (+ (* 2.56_X1 1.81_X2) (* 2.56_X1 1.81_X2)) #)
300	(+ (+ 1.50_X4 0.53_X5) (- (* (+ 1.50_X4 0.53_X5) (+ 1.50_X4 0.53_X5)) #))
310	(+ (+ 1.50_X4 0.53_X5) (- (* (+ 1.50_X4 0.53_X5) (+ # #)) #))
320	(+ (+ (+ 1.50_X4 0.53_X5) (+ 1.50_X4 0.53_X5)) #)
330	(+ (* (+ 1.50_X4 0.53_X5) #) (- # (* # (- (* # (* 1.89_X2 1.84_X1)) 14.65))))
340	(+ # (- # (* (+ # (+ 1.50_X4 0.53_X5)) #)))
350	(+ (* (+ 1.50_X4 0.53_X5) #) (- # (* (+ (* 2.56_X1 1.81_X2) (+ 1.50_X4 0.53_X5)) (- (* (* 2.56_X1 1.81_X2) (* 2.56_X1 1.81_X2)) 14.65))))
360	(+ # (- (* (+ 1.50_X4 0.53_X5) (* 2.56_X1 1.81_X2)) #))
370	(+ (+ 1.50_X4 0.53_X5) (- (+ 0.53_X5 #) #))
380	(+ (* 1.78_X3 1.97_X3) (- (+ # (+ 1.50_X4 0.53_X5)) #))
390	(+ # (- # (- (* (+ (* 2.56_X1 1.81_X2) (+ 1.50_X4 0.53_X5)) (- (* (* 2.56_X1 1.81_X2) (* 1.89_X2 1.84_X1)) 14.65)) (+ (+ 1.50_X4 0.53_X5) #))))
400	(+ (* 1.89_X2 #) (+ # (- (* 1.89_X2 1.84_X1) (- # #))))
410	(+ # (- (* (+ 1.50_X4 -0.08_X5) (+ (* 2.56_X1 1.81_X2) (* 2.56_X1 1.81_X2))) #))
420	(+ (+ (+ 1.50_X4 0.53_X5) (+ 1.50_X4 0.53_X5)) #)
430	(+ (+ (+ 1.50_X4 0.53_X5) (* 2.56_X1 1.81_X2)) #)
440	(+ (+ 1.50_X4 0.53_X5) (+ (+ 1.50_X4 0.53_X5) #))
450	(+ (+ (+ 1.50_X4 0.53_X5) (+ 1.50_X4 0.53_X5)) #)
460	(+ # (- # (- (* (+ (+ 1.50_X4 0.53_X5) (* 2.56_X1 1.81_X2)) (- (* (* 2.56_X1 1.81_X2) (* 1.81_X2 1.84_X1)) 14.65)) #)))
470	(+ (+ (* 2.56_X1 1.81_X2) (* 2.56_X1 1.81_X2)) #)
480	(+ (+ # #) (+ (+ 1.50_X4 0.53_X5) #))
490	(+ (+ # (- (* # (* 1.78_X3 1.97_X3)) 1.97_X3)) #)
500	(+ # (- (- # 1.97_X3) (* (+ (+ 1.50_X4 0.53_X5) (* 2.56_X1 1.81_X2)) (- (* (* 2.56_X1 1.81_X2) (* 1.81_X2 #)) 14.65)))

Table 7.6: SGP Friedman-2 Proportional Most Frequent Schemas

Generation	Schema
1	(/ 0.76_x12 (+ (+ 15.63 (* 2.85_x8 (* (+ (+ -17.81 (/ (- (+ 12.29 0.91_x3) (+ 17.18 -19.38)) (* -0.80_x13 (+ 18.35 0.89_x11))))) (- # #)) 2.14_x23))) (+ -14.33 15.05)))
10	(- (/ -0.15_x12 #) (* (/ 1.32_x6 -1.06) -12.15))
20	(- (/ 1.24_x6 #) (- 1.12_x6 (+ 0.25_x21 #)))
30	(/ (- (+ 1.36_x10 (+ 1.68_x13 1.14_x1)) (+ (* -13.75 20.01) (- # -0.85_x6))) -17.90)
40	(+ 19.12 (* 0.71_x25 (/ (* # #) #)))
50	(- (+ (+ # (+ 1.32_x18 1.61_x12)) -0.42) #)
60	(- (/ (- (* (- 2.41_x6 1.58_x6) 9.28) (- # 1.49_x15)) 1.14_x1) (- -1.40 #))
70	(* (- 1.90_x18 (- 14.07 (- (- 2.49_x12 1.54_x24) #))) #)
80	(* (* (- 0.49_x3 1.14_x1) 0.53_x23) (/ # 1.14_x1))
90	(* # (* (* 2.34_x6 (- 1.41 2.11)) 0.53_x23))
100	(- (* (+ 0.57_x5 #) (/ 2.27_x6 1.14_x1)) #)
110	(- (* (- # (* # 0.53_x23)) (/ 2.27_x6 1.14_x1)) #)
120	(- # (/ # (- (* (* 16.29 1.31_x6) 0.08_x6) 0.76_x24)))
130	(* (/ (* (* -5.47 0.49_x6) #) 1.14_x1) #)
140	(+ 0.91_x7 (/ (- (/ 1.93_x6 #) (* # 0.53_x23)) 1.14_x1))
150	(/ (+ (- # -11.55) (* # 0.53_x23)) 1.14_x1)
160	(/ (+ (- (/ (- # 1.93_x6) 1.18_x9) -11.55) #) 1.14_x1)
170	(+ # (/ (* 15.89 (* 1.80_x23 2.24_x6)) 1.14_x1))
180	(+ # (/ (* 15.89 (* 1.80_x23 2.24_x6)) 1.14_x1))
190	(+ # (/ (* 15.89 (* 1.80_x23 2.24_x6)) 1.14_x1))
200	(+ (/ (/ 2.27_x6 #) #) (/ # 1.14_x1))
210	(- (+ 0.31_x19 #) (/ (/ (/ # (/ 2.27_x6 1.14_x1)) 1.14_x1) 2.27_x6))
220	(- (/ (- 2.27_x6 (* # #)) 1.14_x1) -13.63)
230	(/ (- # (* # (+ 1.11_x6 -9.71))) 1.14_x1)
240	(+ # (/ (+ (* 15.72 2.27_x6) (* 1.80_x23 2.24_x6)) 1.14_x1))
250	(* (/ (+ # (* 1.80_x23 2.24_x6)) 1.14_x1) 7.72)
260	(* (/ (- (+ # #) #) 1.14_x1) #)
270	(+ # (/ (+ # (* 1.80_x23 2.24_x6)) 1.14_x1))
280	(* (/ (+ # (* 1.80_x23 2.24_x6)) 1.14_x1) #)
290	(+ (/ (+ # (* 11.21 1.81_x23)) #) #)
300	(+ (+ (* 1.80_x23 2.24_x6) (* 1.80_x23 2.24_x6)) #)
310	(+ (+ # (/ (* 1.80_x23 2.24_x6) 1.14_x1)) (* -12.47 2.07_x13))
320	(+ (/ # 1.14_x1) (/ (* 2.27_x23 2.24_x6) 1.14_x1))
330	(+ (/ (* 1.80_x23 2.24_x6) #) (/ (* 1.80_x23 2.24_x6) 1.14_x1))
340	(+ (+ # (/ (* 1.80_x23 2.24_x6) 1.14_x1)) (* -12.47 2.07_x13))
350	(+ (/ (* (* 1.80_x23 2.24_x6) -6.49) 1.14_x1) #)
360	(+ # (+ (* 1.80_x23 2.24_x6) (* 1.80_x23 2.24_x6)))
370	(/ (+ (+ (* 1.79_x12 2.24_x6) #) #) 1.14_x1)
380	(/ (+ (+ (* 1.80_x23 2.24_x6) #) (* 1.80_x23 2.24_x6)) 1.14_x1)
390	(+ # (+ (* # 2.24_x6) (* 1.80_x23 2.24_x6)))
400	(+ # (/ (+ (* 1.80_x23 2.24_x6) #) 1.14_x1))
410	(/ (+ # (+ # (* 1.80_x23 2.24_x6))) 1.14_x1)
420	(/ (+ (+ (+ (* 1.80_x23 2.24_x6) (* 1.80_x23 2.24_x6)) #) #) 1.14_x1)
430	(/ (+ (+ # (* 1.80_x23 2.24_x6)) #) 1.14_x1)
440	(/ (+ (+ (- 1.43_x18 #) 18.18) #) 1.14_x1)
450	(/ (+ # (- (* 1.80_x23 2.24_x6) -15.63)) 1.14_x1)
460	(/ (+ (* 1.80_x23 2.24_x6) (* # 2.24_x6)) 1.14_x1)
470	(/ (+ (+ # (+ 0.55_x23 0.05)) (/ # 1.14_x1)) 1.14_x1)
480	(/ (+ (* 1.80_x23 2.20_x6) (/ # 1.14_x9)) 1.14_x1)
490	(/ (+ (* 2.24_x6 2.16_x23) (+ # #)) 1.14_x1)
500	(/ (+ (* 2.24_x6 2.16_x23) (+ # #)) 1.14_x1)

Table 7.7: SGP Tower Proportional Most Frequent Schemas

Generation	Schema
1	(+ # (* (+ (+ (- 1.88 (- # (/ 8.66 8.39))) (- (+ 6.30 -15.87) 1.13_x6)) #) (- 1.93 9.34)))
10	(+ (* -2.21 #) (* # (- 1.93 9.34)))
20	(* 12.53 (* (/ # (- -2.61 2.07_x16)) -0.50))
30	(/ (- (+ (* 1.71_x6 #) #) (+ # 0.95_x4)) 2.00_x14)
40	(+ -18.82 (- # (- # (+ (+ 1.39_x13 -0.26_x8) -2.61))))
50	(+ -18.82 (- # (- # (+ # # -2.61))))
60	(+ -18.82 (- (+ (* 1.71_x6 #) (/ # -0.06_x11)) (+ # -2.61)))
70	(+ -18.82 (- # (+ 1.39_x13 (/ 0.73_x19 -5.11))))
80	(+ -18.82 (- (+ # (/ # -0.06_x11)) #))
90	(+ -18.82 (- # (+ 1.39_x13 (/ # -5.11))))
100	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
110	(+ # (- # (- # (+ 1.39_x13 -0.27_x15))))
120	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (/ # -0.06_x11))
130	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
140	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
150	(+ (/ (* # 0.70_x1) -0.06_x11) (- # #))
160	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
170	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
180	(+ (/ # -0.06_x11) (/ (* 0.70_x1 0.70_x1) -0.06_x11))
190	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # (- # #)))
200	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
210	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (/ # -0.06_x11))
220	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (/ # -0.06_x11))
230	(+ (/ # -0.06_x11) (- # (- # #)))
240	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # (- # (+ 1.39_x13 -0.27_x15))))
250	(+ # (- # (- # (+ 1.39_x13 -0.27_x15))))
260	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
270	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (/ # -0.06_x11))
280	(+ # (- # (- # (+ 1.39_x13 -0.27_x15))))
290	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
300	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
310	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
320	(+ (/ (* 0.70_x1 #) -0.06_x11) (- # #))
330	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
340	(+ # (- # (- # (+ 1.39_x13 -0.27_x15))))
350	(+ # (- # (- # (+ 1.39_x13 -0.27_x15))))
360	(+ (/ (* 0.70_x1 0.70_x1) #) (- # #))
370	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (/ # -0.06_x11))
380	(+ (/ (* 0.70_x1 0.71_x1) -0.06_x11) (- # (- # (+ 1.39_x13 -0.27_x15))))
390	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
400	(+ (/ (* 0.70_x1 0.70_x1) -0.06_x11) (- # #))
410	(+ (/ (* 0.70_x1 0.71_x1) -0.06_x11) (- # #))
420	(+ (/ # -0.06_x11) (- (/ # #) (- # (+ 1.39_x13 -0.27_x15))))
430	(+ # (- # (- # (+ 1.39_x13 -0.27_x15))))
440	(+ (/ (* 0.70_x1 0.71_x1) -0.06_x11) (- # #))
450	(+ (/ (* 0.70_x1 0.71_x1) -0.06_x11) (- # #))
460	(+ (/ (* 0.70_x1 0.71_x1) -0.06_x11) (- # #))
470	(+ (/ (* 0.70_x1 0.71_x1) -0.06_x11) (- # #))
480	(+ # (- # (- (+ (* 1.71_x6 #) #) (+ 1.39_x13 -0.27_x15))))
490	(+ (/ # -0.06_x11) (- # (- # (+ 1.39_x13 -0.27_x15))))
500	(+ (/ (* 0.70_x1 0.71_x1) -0.06_x11) (- # #))

Table 7.8: SGP Tower Tournament Most Frequent Schemas

Generation	Schema
1	(* (- 1.36_X4 0.27_X3) (* (+ # #) #))
5	(* 1.27_X4 (+ (* (+ (- 17.44 2.32_X6) 0.70_X3) 1.16_X3) (* # #)))
10	(+ # (* 3.01_X3 (- 1.36_X4 (* 0.50 0.40_X4))))
15	(+ (* 1.70_X1 (+ 0.33_X7 2.08_X2)) (+ (* (* 0.86_X6 0.95_X5) (- (+ 0.33_X7 2.08_X2) #)) #))
20	(+ (* 1.70_X1 (+ 0.33_X9 2.08_X2)) (+ (* (* 0.86_X6 0.95_X5) (- 5.00 #)) (* 3.01_X3 #)))
25	(+ (* 1.70_X1 (+ (* 0.16_X9 1.67_X7) (+ (* 0.16_X9 1.67_X7) 2.08_X2))) #)
30	(+ (* 1.70_X1 (+ (* 1.39_X9 1.67_X7) 2.08_X2)) #)
35	(+ (* 1.70_X1 (+ (* 1.39_X9 1.67_X7) 2.08_X2)) (+ (* (* 0.86_X6 0.95_X5) (- 5.00 (* # #))) (* 3.01_X3 (- 1.36_X4 (* 0.27_X10 (* 0.85_X6 (+ # -4.26)))))))
40	(+ (* 1.70_X1 (+ (* 1.39_X9 1.67_X7) 2.08_X2)) #)
45	(+ (* 1.70_X1 (+ (* 1.39_X9 1.67_X7) 2.20_X2)) #)
50	(+ (* 1.75_X1 (+ (* 1.39_X9 1.67_X7) 2.26_X2)) (+ # (* 3.01_X3 (- 1.36_X4 (* 0.27_X10 (* 0.85_X6 #))))))
55	(+ (* 1.75_X1 (+ (* 1.39_X9 1.67_X7) 2.32_X2)) (+ (* (* 0.86_X6 0.95_X5) (- 5.00 #)) (* 3.01_X3 (- 1.36_X4 (* 0.27_X10 #)))))
60	(+ (* 1.75_X1 (+ (* 1.39_X9 1.67_X7) 2.32_X2)) (+ # (* 3.01_X3 (- 1.36_X4 (* 0.27_X10 (* 0.85_X6 (+ (+ # -1.68) -4.26)))))))
65	(+ (* 1.75_X1 (+ (* 1.39_X9 1.67_X7) 2.33_X2)) (+ (* (* 0.86_X6 0.95_X5) (- 4.98 (* (* # 1.67_X7) #))) (* 3.01_X3 (- 1.36_X4 (* 0.27_X10 (* 0.85_X6 (+ (+ # -1.68) -4.28)))))))
70	(+ (* 1.75_X1 (+ (* 1.39_X9 1.67_X7) 2.33_X2)) (+ (* (* 0.86_X6 0.95_X5) (- 4.98 #)) (* 3.01_X3 (- 1.36_X4 (* 0.27_X10 (* 0.85_X6 (+ (+ (* (* 0.09_X1 2.32_X2) 0.16_X9) -1.68) -4.28)))))))
75	(+ (* 1.75_X1 (+ (* 1.39_X9 1.67_X7) 2.34_X2)) (+ # (* 3.01_X3 (- 1.36_X4 #))))
80	(+ (* 1.75_X1 (+ (* 1.40_X9 1.67_X7) 2.34_X2)) #)
85	(+ (* 1.75_X1 (+ (* 1.40_X9 1.67_X7) 2.34_X2)) (+ (* (* 0.86_X6 0.95_X5) #) #))
90	(+ (* 1.75_X1 (+ (* 1.40_X9 1.67_X7) 2.34_X2)) (+ # #))
95	(+ (* 1.75_X1 (+ (* 1.40_X9 1.67_X7) 2.34_X2)) #)

Table 7.9: OSGP Poly-10 Proportional Most Frequent Schemas

Generation	Schema
1	(+ # (+ (/ 0.67_X6 (* 2.03_X10 (* (+ 0.10_X3 12.05) 1.53_X8))) (* 5.11 (/ (+ # 1.06_X3) 0.18_X5))))
2	(+ (/ (+ # (+ (/ 3.34 6.11) 0.04_X1)) #) (* 2.01_X6 1.41_X5))
3	(+ (/ # (/ (* (+ # -0.24_X9) #) 0.85_X8)) (* 2.01_X6 1.41_X5))
4	(+ (/ # (/ (* # #) 0.85_X8)) (* 2.01_X6 1.41_X5))
5	(* 0.77_X3 (* (/ 1.62_X4 (- # 5.47)) 16.66))
6	(* 0.77_X3 (* (/ 1.62_X4 (- # 5.47)) 16.66))
7	(* 0.77_X3 (* (/ 1.62_X4 (- # 5.47)) 16.66))
8	(+ (* 0.77_X3 (+ # 1.28_X4)) (* 2.01_X6 1.41_X5))
9	(+ (* 0.77_X3 (+ # 1.28_X4)) (* 2.01_X6 1.41_X5))
10	(+ (+ # (* 0.77_X3 (* (/ 1.62_X4 15.99) 16.66))) (* 2.01_X6 1.41_X5))
11	(+ (+ # (+ (* 1.70_X2 (* 1.78_X1 (+ 12.28 -11.71))) (* 0.77_X3 #))) (* 2.01_X6 1.41_X5))
12	(+ (+ # (+ (* 1.70_X2 (* 1.78_X1 (+ 12.28 -11.71))) #)) (* 2.01_X6 1.41_X5))
13	(+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 15.99 #)) (* 2.01_X6 1.41_X5)))
14	(+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 15.99 #)) (* 2.01_X6 1.41_X5)))
15	(+ (* 1.70_X2 2.02_X1) (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 15.99 #)) (* 2.01_X6 1.41_X5)))
16	(+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) #) (* 2.01_X6 1.41_X5)))
17	(+ (* 1.70_X2 2.02_X1) (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 15.99 #)) (* 2.01_X6 1.41_X5)))
18	(+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 15.99 #)) (* 2.01_X6 1.41_X5)))
19	(+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 15.99 #)) (* 2.01_X6 1.41_X5)))
20	(+ (- # (/ (* 1.70_X2 2.02_X1) (- 1.51_X6 -6.20))) #)
21	(+ (/ (+ (+ 12.81 12.81) 2.01_X6) (+ 12.81 #)) (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 15.99 (* 1.78_X1 #))) (+ (* 1.70_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
22	(+ (/ (+ (* 2.01_X6 1.41_X5) 12.81) (+ 12.81 -0.73_X6)) (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 15.99 #)) (+ (* 1.70_X2 1.78_X1) (* 2.01_X6 1.41_X5))))
23	(+ (/ (+ (+ 12.81 12.81) 2.01_X6) (+ 12.81 (- (* 1.70_X2 2.02_X1) #))) (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) #) (+ (* 1.70_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
24	(+ (/ (+ # 2.01_X6) #) (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 15.99 (* (* 1.78_X4 1.65) (* 1.70_X2 2.02_X1)))) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
25	(+ (/ (+ # 2.01_X6) (+ 12.81 #)) (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 15.99 (* (* 1.78_X4 1.65) (* 1.70_X2 2.02_X1)))) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
26	(+ (/ (+ (* 2.01_X6 1.41_X5) 2.01_X6) (+ 12.81 #)) #)
27	(+ (/ (+ (* 2.01_X6 1.41_X5) 2.01_X6) (+ 12.81 #)) #)
28	(+ (/ (+ (* 2.01_X6 1.41_X5) 2.01_X6) (+ 12.81 #)) #)
29	(+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) #) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
30	(+ (/ # (+ 12.81 #)) (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- # (* (* 1.78_X4 1.65) (* 2.01_X6 1.42_X10)))) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
31	(+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 16.22 (* (* 1.78_X4 1.65) (* 2.01_X6 1.42_X10)))) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
32	(+ (/ (+ # 2.01_X6) (+ 12.81 #)) (+ (/ (* 1.74_X3 #) (- 16.22 (* (* 1.78_X4 1.65) (* 2.01_X6 1.42_X10)))) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
33	(+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) #) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
34	(+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 16.22 (* (* 1.78_X4 1.99) (* 2.01_X6 1.42_X10)))) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
35	(+ (/ (+ (* (+ (* 1.70_X9 2.02_X1) 2.01_X6) 1.04_X7) 2.01_X6) (+ 12.81 (- # #))) #)
36	(+ # (+ # (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
37	(+ (/ (+ # 2.01_X6) (+ 12.81 (- # #))) (+ # (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
38	(+ # (+ # (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
39	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ (/ (* 1.74_X3 #) (- 16.22 #)) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
40	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ # (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))

Table 7.10: OSGP Poly-10 Gender Specific Most Frequent Schemas – Part 1

Generation	Schema
41	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ (/ (* 1.74_X3 (* 1.78_X4 15.99)) (- 16.22 #)) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5))))
42	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ # (+ (/ # (- 16.22 #)) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
43	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.65)) (- 16.22 #)) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
44	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.65)) (- 16.22 #)) (+ (* 1.28_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
45	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.65)) (- 16.22 #)) (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
46	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ (/ # (+ 12.81 (+ 1.65 1.41_X5))) (+ (/ (* 1.74_X3 (* 1.78_X4 15.65)) (- 16.22 #)) (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
47	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ (/ # #) (+ (/ (* 1.74_X3 (* 1.78_X4 15.65)) #) (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
48	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ (/ # (- 16.22 #)) (+ # (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
49	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.65)) (- 16.22 #)) (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
50	(+ (* (* 2.01_X6 1.42_X10) 1.36_X3) (+ (/ (+ (* (* 1.82_X9 (* 1.79_X1 15.99)) 1.04_X7) #) 15.65) #))
51	(+ (* (* 2.01_X6 1.15_X10) 1.36_X3) (+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.65)) #) (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
52	(+ (* (* 2.01_X6 1.42_X10) 1.04_X3) (+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.65)) (- 16.22 #)) (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
53	(+ (* (* 2.01_X6 1.42_X10) 1.04_X3) (+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.65)) 16.22) (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
54	(+ (* (* 2.01_X6 1.42_X10) 1.04_X3) (+ # (+ (/ (* 1.74_X3 (* 1.78_X4 15.65)) 16.22) (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
55	(+ (* (* 2.01_X6 1.42_X10) 1.04_X3) (+ # (+ # (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))
56	(+ (* (* 2.01_X6 1.42_X10) 1.04_X3) (+ # (+ # (+ (* 1.48_X2 2.02_X1) (* 2.01_X6 1.41_X5)))))

Table 7.11: OSGP Poly-10 Gender Specific Most Frequent Schemas – Part 2

Generation	Schema
1	(- (* (/ (- # (* (/ 16.50 -0.58_X) 5.63)) 1.50_X) #) -18.59)
2	(- (* (/ (- # (* (/ 16.50 -0.58_X) 5.63)) 1.50_X) (+ # # 16.76)) -18.59)
3	(- (* (/ (- # (* (/ 16.50 -0.58_X) 5.63)) 1.50_X) #) -18.59)
4	(* -4.27 (/ # (* (- # (/ -6.26 1.65_Y)) (- -0.63_Y 1.03_Y))))
5	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
6	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
7	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
8	(- 2.45 (/ -0.41_X (- (/ (- # (/ (/ -1.28 3.35_Y) 0.69_Y)) -0.71_X) 0.49_X)))
9	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
10	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
11	(- 2.45 (/ -0.41_X (- (/ (- # #) -0.71_X) 0.49_X)))
12	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
13	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
14	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
15	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
16	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
17	(- 2.45 (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
18	(- (- 2.77 (/ 2.45 #)) (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
19	(- # (/ -0.41_X (- (/ (- # (/ (/ -1.28 3.35_Y) 0.69_Y)) -0.71_X) 0.49_X)))
20	(- (- 2.77 #) (/ -0.41_X (- (/ # -0.71_X) 0.49_X)))
21	(- (- 2.77 (/ 2.45 (- (+ -1.28 14.06) (/ (/ -6.26 1.65_Y) 0.69_Y)))) #)
22	(- (- 2.77 (/ 2.45 (- 10.54 (/ # 0.69_Y)))) (/ -0.41_X (- # 0.49_X)))
23	(- (- 2.77 (/ 2.45 (- 10.54 (/ (/ -6.26 0.69_Y) 0.69_Y)))) (/ -0.41_X (- # 0.49_X)))
24	(- # (/ 2.31 (- (+ -1.34 13.03) (/ (/ -4.84 1.35_X) 0.69_X))))
25	(- # (/ 2.31 (- # (/ (/ -4.84 1.35_X) 0.69_X))))
26	(- # (/ # (- (- (+ -1.34 13.03) (/ (/ -4.84 1.35_X) 0.69_X)) (/ (/ -4.84 1.35_X) 0.69_X))))
27	(- # (/ 2.31 (- (+ -1.34 13.03) (/ (/ (/ # 0.69_X) 1.35_X) 0.69_X))))
28	(- (- 2.91 (/ 2.29 (- # #))) #)
29	(- (- 2.91 (/ 2.29 #)) (/ 2.31 (- (+ -1.34 13.03) (/ (/ (/ (/ -4.84 1.35_X) 0.69_X) 1.35_X) 0.69_X))))
30	(- # (/ 2.31 (- (+ -1.34 13.03) (/ (/ (/ (/ -4.84 0.69_X) 0.69_X) 1.35_X) 0.69_X))))
31	(- (- 2.91 #) (/ 2.31 (- (+ -1.34 13.03) (/ (/ (/ (/ -4.84 0.69_X) 0.69_X) 1.35_X) 0.69_X))))
32	(- # (/ 2.31 (- (+ -1.34 13.03) (/ (/ (/ (/ -4.84 0.69_X) 0.69_X) 1.35_X) 0.69_X))))
33	(- (- 2.91 (/ 2.29 #)) (/ 2.31 (- (+ -1.34 13.03) (/ (/ (/ (/ -4.84 0.69_X) 0.69_X) 1.35_X) 0.69_X))))
34	(- (- 2.91 (/ 2.29 (- (- (+ -1.34 13.03) (/ (/ (/ (/ -6.26 0.69_Y) 0.69_Y) 1.65_Y) 0.69_Y)) #))) #)
35	(- # (/ 2.31 (- (+ -1.34 13.03) (/ # 0.69_X))))
36	(- (- 2.91 (/ 2.29 (- (- (+ -1.34 13.03) (/ (/ (/ (/ -6.26 0.69_Y) 0.69_Y) 1.65_Y) 0.69_Y)) #))) (/ 2.31 #))
37	(- # (/ 2.31 (- (+ -1.34 13.03) #)))
38	(- # (/ 2.31 (- (+ -1.34 13.03) #)))

Table 7.12: OSGP Pagie-1 Proportional Most Frequent Schemas

Generation	Schema
1	(/ (* (+ (- # 4.09) -12.99) (* (+ (- (/ -3.88 1.69_X) (/ (* 0.00_Y 1.14_Y) (- -6.04 (- -1.67 6.37)))) 0.72_X) (- # #))) 0.34_X)
2	(/ (* (+ (- # 4.09) -12.99) #) 0.34_X)
3	(- (+ # (- 0.90 (* (* (* -18.77 19.97) (* -15.58 1.15_X)) 0.69_X))) -19.28)
4	(/ (+ # (+ 1.77_X 0.86_X)) (+ (* 12.42 (/ (/ 7.74 (* 1.12_Y 1.88_Y)) 1.81_X)) 0.33_X))
5	(+ -18.07 (/ 0.08_X (+ # (+ # (/ 19.80 0.17_X)))))
6	(+ -18.07 (/ 0.08_X (+ # (+ # (/ 19.80 0.17_X)))))
7	(+ -18.07 (/ 0.08_X (+ # (+ (* (* -10.93 -18.92) 0.59_X) (/ 19.80 0.17_X)))))
8	(/ (+ (/ # -13.07) #) (+ (* # (/ (/ 7.74 (* 1.12_Y 1.88_Y)) 1.81_X)) 0.33_X))
9	(/ # (+ (/ (/ 7.74 (* 1.12_Y 1.88_Y)) 1.81_X) 0.33_X))
10	(/ # (+ (/ (/ 7.74 (* 1.12_Y 1.88_Y)) 1.81_X) 0.33_X))
11	(/ # (+ (/ (/ 7.74 (* 1.12_Y 1.88_Y)) 1.81_X) 0.33_X))
12	(/ # (+ (/ (/ 7.74 (* 1.12_Y 1.88_Y)) 1.81_X) 0.33_X))
13	(/ # (+ (/ (/ 7.74 (* 1.12_Y 1.88_Y)) 1.81_X) 0.33_X))
14	(/ # (+ (/ (/ 7.74 (* 1.12_Y 1.88_Y)) 1.81_X) 0.33_X))
15	(- # (/ (+ -19.42 (/ -18.44 (- 15.77 (/ (* (+ (* 0.10_X 0.10_Y) -12.99) -18.68) (* 1.34_Y #))))) 1.67))
16	(- # (/ (+ -19.42 (/ -18.44 (- 15.77 (/ (* (+ # -12.99) -18.68) (* 1.34_Y (* 1.52_Y -14.64)))))) 1.67))
17	(- (+ (/ (- -5.51 7.05) (- 4.65 (- (* 1.50_X (* -9.60 1.50_X)) 16.43))) #) (/ (+ -19.42 (/ -18.44 #)) 1.67))
18	(- (+ (/ (- -5.51 (- 14.66 8.72)) (- 1.88_X (- (* 1.50_X (* -9.60 1.50_X)) 16.43))) #) (/ # 1.67))
19	(- (+ (/ (- -5.51 (- 14.66 8.72)) (- 1.81_X (- # 16.43))) #) (/ (+ 7.74 #) 1.67))
20	(- # (/ (+ -19.42 (/ -18.44 (- 15.77 (/ (* (/ (* (+ # -11.73) -18.68) (* 1.34_Y (* 1.52_Y -14.64))) -18.68) (* 1.34_Y #))))) 1.67))
21	(- # (/ (+ -19.42 (/ -18.44 (- 15.77 (/ (* -18.68 -18.68) (* 1.34_Y (* 1.52_Y -14.64)))))) 1.67))
22	(- # (/ (+ -19.42 (/ -18.44 (- 15.77 (/ (* (/ (* (+ -16.99 -11.73) -18.68) (* 1.34_Y #)) -18.68) (* 1.34_Y (* 1.52_Y -14.64)))))) 1.67))
23	(- (+ (/ (+ (* 0.10_X 0.10_Y) -12.99) (- 7.58 (- (* 1.48_X (* -9.60 1.50_X)) 7.74))) #) (/ # 1.67))
24	(- (+ # #) (/ (+ -19.42 #) 1.67))
25	(- # (/ (+ -19.42 (/ -18.44 #)) 1.67))
26	(- (+ (/ (- -5.51 12.81) (- 7.58 (- (* 1.39_X (* # 1.50_X)) 16.43))) #) (/ # 1.67))
27	(- # (/ (/ -18.44 (- 15.77 #)) 1.67))
28	(- (+ (/ (- -5.51 12.81) (- 7.58 (- (* 1.39_X (* (* # (* -9.60 1.41_X)) 1.50_X)) 16.43))) #) (/ # 1.67))
29	(- (+ (/ (- -5.51 12.81) (- 7.58 (- (* 1.39_X #) 16.43))) #) (/ (/ -18.44 #) 1.67))
30	(- (+ (/ (- -5.51 12.81) (- 7.58 (- (* 1.39_X #) 16.43))) #) (/ (/ -18.44 #) 1.67))
31	(- (+ (/ (- -5.51 12.81) (- 7.58 (- (* 1.39_X (* (* 1.48_X (* -5.51 1.50_X)) 1.50_X)) 16.43))) #) (/ # 1.67))
32	(- # (/ (+ -19.42 (/ -18.44 (- 14.35 (/ (* (/ (* -18.57 -18.68) #) -18.68) (* 1.34_Y (* 1.12_Y -14.64)))))) 1.67))
33	(- (+ (/ (- -5.51 12.81) (- 7.58 (- (* 1.39_X (* (* 1.48_X (* -5.51 1.50_X)) 1.50_X)) 16.43))) #) (/ # 1.67))
34	(- # (/ (+ -19.42 (/ -18.44 (- 14.35 #))) 1.67))
35	(- # (/ (/ -18.44 (- 15.77 (/ (* (/ (* (+ -18.92 -12.99) -18.68) (* 1.34_Y (* 1.47_Y -14.64))) -18.68) (* 1.34_Y (* 1.52_Y -11.73))))) 1.67))

Table 7.13: OSGP Pagie-1 Gender Specific Most Frequent Schemas

Generation	Schema
1	(* (+ # (* 6.12 (* 2.61_X4 15.36))) 2.94_X2)
2	(* (+ # (* 6.12 (* 2.61_X4 15.36))) 2.94_X2)
3	(* (+ # (* 6.12 (* 2.61_X4 15.36))) 2.94_X2)
4	(+ # (- 1.71_X10 (+ (- (- # (* 17.69 (- 0.57_X2 (* (+ 1.08_X3 (* -8.80 0.39_X4)) 1.08_X2)))) (+ # (* 1.60_X5 19.75))) 5.71)))
5	(- (* 0.90_X4 (- 2.56_X7 (+ -16.86 #))) (- # (- (/ (* 19.43 2.61_X2) 4.46) (+ (* (+ 1.28_X5 -18.18) 0.39_X1) #))))
6	(- (- (+ (- (* 7.73 -14.53) 1.25_X4) #) #) #)
7	(- # (+ -17.00 (+ (+ # (* 1.44_X1 5.85)) (* -19.33 (- -0.61_X2 0.69_X4)))))
8	(/ (+ (+ 0.86_X2 #) (+ # 1.02_X1)) 14.26)
9	(/ (+ (+ 0.86_X2 (+ (+ 1.28_X5 -18.59) (+ 2.21_X4 0.44_X2))) (+ # 1.02_X1)) 14.26)
10	(/ (+ (+ 0.86_X2 (+ (+ 1.28_X5 -18.59) (+ 2.21_X4 0.44_X2))) #) 14.26)
11	(/ (+ (+ 0.86_X2 (+ (+ 1.28_X5 -18.59) (+ 2.21_X4 0.44_X2))) #) 14.26)
12	(+ (/ -21.31 (+ 0.41_X9 (- (/ -10.44 1.77_X1) (- 7.15 (+ # 1.05_X4))))) (+ 0.86_X2 #))
13	(+ (/ -21.31 (+ 0.41_X9 #)) (+ 0.86_X2 (+ (+ 1.28_X5 #) (+ 2.21_X4 0.44_X2))))
14	(+ 0.84_X4 (+ (+ -16.86 (* 0.58_X5 0.60_X2)) (/ -18.47 (+ # (- 7.20 (- (* (* 1.44_X1 5.85) (+ # (* (* 19.69 1.25_X2) (/ 11.07 -17.00)))) 16.90))))))
15	(+ 0.84_X4 (+ (+ -16.86 (* 0.58_X5 0.60_X2)) (/ -18.47 #)))
16	(+ 0.84_X4 (+ (+ -16.86 (* 0.58_X5 0.60_X2)) (/ -18.47 #)))
17	(+ 0.84_X4 (+ (+ -16.86 (* 0.58_X5 0.57_X5)) #))
18	(+ 0.84_X4 (+ (+ -16.86 (* 0.58_X5 0.57_X5)) #))
19	(+ 0.84_X4 (+ (+ -16.86 (* 0.58_X5 0.57_X5)) #))
20	(+ 0.84_X4 (+ (+ -16.86 (* 0.58_X5 0.57_X5)) #))
21	(+ 0.84_X4 (+ (* 0.58_X5 0.57_X5) (/ -18.47 #)))
22	(+ 0.84_X4 (+ (* 0.58_X5 0.57_X5) (/ -18.47 #)))
23	(+ 0.84_X4 (+ (* 0.58_X5 0.57_X5) (/ -18.47 #)))
24	(+ 0.84_X4 (+ (* 0.58_X5 0.57_X5) (/ -18.47 #)))
25	(+ 0.84_X4 (+ # (/ -18.47 (+ # #))))
26	(+ 0.84_X4 (+ 0.37_X5 (/ -18.47 (+ # #))))
27	(+ 0.84_X4 (+ 0.41_X5 (/ -18.47 (- 7.20 #))))
28	(+ 0.84_X4 (+ 0.41_X5 (/ (+ -16.86 (* 1.44_X1 #)) (- 7.20 (- (* # (* # (* # 0.39_X1))) 16.90)))))
29	(+ 0.84_X4 (+ 0.41_X5 (/ (+ -16.86 #) (- 7.20 (- (* (* 1.44_X1 (* 1.44_X1 #)) (* # (* (* # #) 0.39_X1))) 16.90)))))
30	(+ 0.84_X4 (+ 0.41_X5 (/ (+ -16.86 #) (- 7.20 (- # 16.90)))))
31	(+ 0.84_X4 (+ 0.41_X5 (/ (+ -16.86 #) #)))
32	(+ 0.84_X4 (+ 0.41_X5 (/ (+ -16.86 (* 1.44_X1 #)) (- 7.20 (- # 16.90)))))
33	(+ 0.84_X4 (+ 0.41_X5 (/ (+ -16.86 (* 1.44_X1 #)) (- 7.20 #))))
34	(+ 0.84_X4 (+ 0.41_X5 (/ (+ -16.86 #) (- 7.20 (- (* (* 1.44_X1 (* (/ 0.39_X1 (* 0.60_X3 2.61)) (* # 1.25_X2))) #) 16.90)))))
35	(+ 0.84_X4 (+ 0.41_X5 (/ (+ -16.86 #) #)))
36	(+ 0.84_X4 (+ 0.41_X5 (/ (+ -16.86 (* 1.77_X1 (* (* 20.50 (* 0.60_X3 2.61)) 1.25_X2))) (- 7.20 (- (* (* 1.44_X1 (* (/ 0.39_X1 (* 0.60_X3 2.61)) #)) (* # (* # 0.39_X1))) 16.90)))))
37	(+ 0.84_X4 (+ 0.41_X5 (/ (+ -16.86 (* 1.77_X1 #)) #)))

Table 7.14: OSGP Friedman-2 Proportional Most Frequent Schemas

Generation	Schema
1	(+ (/ (- (- # (* 19.34 (- (* (+ 2.89_X4 1.64_X1) 10.03) 12.87))) (* 0.38_X8 -0.92_X6)) #) 0.66_X3)
2	(+ (/ (- (- (+ -3.06 #) (* 19.34 (- (* (+ 2.89_X4 1.64_X1) 10.03) 12.87))) #) #) 0.66_X3)
3	(+ (/ (- (- # (* 19.34 (- (* (+ 2.89_X4 1.64_X1) 10.03) 12.87))) (* 0.38_X8 -0.92_X6)) #) 0.66_X3)
4	(- (- # (* 19.34 (- (* # 10.03) 12.87))) #)
5	(+ 2.89_X4 (+ (/ (- # #) -13.33) #))
6	(/ (- (+ 2.89_X4 1.64_X1) (+ 0.10_X9 (/ 1.91_X2 (/ (- 19.85 (- (/ -0.64_X7 1.82_X1) (/ 0.45_X9 #))) -12.03)))) #)
7	(/ (- (+ 2.89_X4 1.64_X1) (+ 0.10_X9 (/ 1.91_X2 (/ (- 19.85 (- (/ -0.64_X7 1.82_X1) #)) -12.03)))) #)
8	(/ (- (+ 2.89_X4 1.64_X1) (+ 0.10_X9 (/ 1.91_X2 (/ (- 19.85 (- (/ -0.64_X7 1.82_X1) #)) -12.03)))) #)
9	(+ # (+ (/ (+ (* (* -12.68 (+ -7.33 3.25)) (+ (+ (+ 0.26_X6 1.11_X5) 1.30_X2) (+ 2.89_X4 1.64_X1))) 10.16) (- -11.12 (* 0.34_X5 0.64_X8))) #))
10	(+ (/ -16.28 #) (+ (/ (+ (* (* -12.68 (+ -7.33 3.25)) (+ (+ (+ 0.26_X6 1.11_X5) 1.30_X2) (+ 2.89_X4 1.64_X1))) 10.16) #) #))
11	(+ (/ -16.28 #) (+ (/ (+ (* (* -12.68 (+ -7.33 3.25)) (+ (+ (+ 0.26_X6 1.11_X5) 1.30_X2) (+ 2.89_X4 1.64_X1))) 10.16) (- -11.12 (* 0.34_X5 0.64_X8))) #))
12	(+ # (+ (/ (+ (* (* -12.68 (+ -7.33 3.25)) (+ (+ (+ 0.26_X6 1.11_X5) 1.30_X2) (+ 2.89_X4 1.64_X1))) 10.16) (- -11.12 (* 0.34_X5 0.64_X8))) (+ 2.89_X4 1.64_X1)))
13	(+ # (+ (/ (+ (* (* -12.68 (+ -7.33 3.25)) (+ (+ (+ -16.59 1.11_X5) 1.30_X2) (+ 2.89_X4 1.64_X1))) 10.16) (- -11.12 #)) (+ 2.89_X4 1.64_X1)))
14	(+ # (+ (/ (+ (* (+ 12.51 (+ 18.44 1.08_X2)) (+ (+ (+ 4.58 1.11_X5) 1.30_X2) (- 1.57_X1 -15.77))) -12.68) (- -11.12 19.34)) (- 0.77_X2 2.08_X4)))
15	(+ # (+ (/ (+ (* (+ 12.51 (+ 18.44 1.08_X2)) (+ (+ (+ 4.58 1.11_X5) 1.30_X2) (- 1.57_X1 -15.77))) -12.68) (- -11.12 19.34)) (- 0.77_X2 2.08_X4)))
16	(+ # (+ (/ (+ (* (+ 12.51 (+ 18.44 1.08_X2)) (+ (+ (+ 4.58 1.11_X5) 1.30_X2) (- 1.57_X1 -15.77))) -12.68) (- -11.12 19.34)) (- 0.77_X2 2.08_X4)))
17	(+ (/ -16.28 (/ (- (/ 6.92 (/ (+ (+ 2.89_X4 1.64_X1) (+ 2.89_X4 1.64_X1)) (/ 2.89_X4 1.64_X1))) (- -9.25 1.05_X8)) -0.40_X2)) (+ (+ (+ # 1.11_X5) 1.30_X2) (+ 2.89_X4 1.64_X1)))
18	(+ (/ -16.28 (/ (- (/ 6.92 (/ # (/ 2.89_X4 1.64_X1))) (- -9.25 1.05_X8)) -0.40_X2)) (+ (+ (+ # 1.11_X5) 1.30_X2) (+ 2.89_X4 1.64_X1)))
19	(+ # (+ (+ (+ # 1.11_X5) 1.30_X2) (+ 2.89_X4 1.64_X1)))
20	(+ (/ -16.28 (/ (- (/ 6.92 (/ (+ (+ 1.64_X1 1.64_X1) (+ 2.89_X4 1.64_X1)) (/ 2.89_X4 1.64_X1))) (- -9.25 1.05_X8)) -0.40_X2)) (+ (+ # 1.30_X2) (+ 2.89_X4 1.64_X1)))
21	(+ (/ -16.28 (/ (- (/ # #) (- -9.25 1.05_X8)) -0.40_X2)) (+ # (+ 2.89_X4 1.64_X1)))
22	(+ (/ -16.28 #) (+ (+ (+ # 1.11_X5) 1.30_X2) (+ 2.89_X4 1.64_X1)))
23	(+ (/ -16.28 (/ # -0.40_X2)) (+ (+ (+ (/ 6.92 (- # (/ # 1.09_X3))) 1.11_X5) 1.30_X2) (+ 2.89_X4 1.64_X1)))
24	(+ (/ 6.47 (+ (+ (* 1.31_X2 0.67_X1) -10.49) 11.36)) (- # (+ 2.95_X4 1.67_X5)))
25	(+ (/ 6.47 (+ (+ (* 1.31_X2 0.67_X1) -10.49) 11.36)) (- # (+ 2.95_X4 1.67_X5)))
26	(+ (/ 6.47 (+ (+ (* 1.31_X2 0.67_X1) -10.49) 11.36)) (- # (+ 2.95_X4 1.67_X5)))
27	(+ (/ 6.47 #) (- # (+ 2.95_X4 1.67_X5)))
28	(+ (/ 6.47 (+ (+ (* 1.31_X2 0.67_X1) #) 11.36)) (- (+ # (* 1.31_X2 0.67_X1)) (+ 2.95_X4 1.67_X5)))
29	(+ (/ 6.47 #) (- # (+ 2.95_X4 1.67_X5)))
30	(+ (/ 6.47 (+ (+ (* 1.31_X2 0.67_X1) (+ # -10.49)) 11.36)) (- (+ # #) (+ 2.95_X4 1.67_X5)))
31	(+ (/ 6.47 (+ (+ # (+ (* 1.31_X2 0.67_X1) -10.49)) 11.36)) (- (+ # #) (+ 2.95_X4 1.67_X5)))
32	(+ (/ 6.47 (+ (+ (* 1.31_X2 1.64_X1) -10.49) 11.36)) #)
33	(+ (/ 6.47 (+ (+ (* 1.31_X2 1.64_X1) -10.49) 11.36)) #)
34	(+ (/ 6.47 (+ (+ (* 1.31_X2 1.64_X1) -10.49) 11.36)) #)
35	(+ (/ 6.47 (+ (+ (* 1.31_X2 1.64_X1) -10.49) 11.36)) #)
36	(+ (/ 6.47 (+ (+ (* 1.31_X2 1.64_X1) -10.49) 11.36)) #)
37	(+ (/ 6.47 (+ (+ (* 1.31_X2 1.64_X1) -10.49) 11.36)) (- # (+ 2.95_X4 1.67_X5)))
38	(+ (/ 6.47 (+ (+ (* 1.31_X2 0.67_X1) (+ (* 1.31_X2 0.67_X1) -10.49)) 11.36)) (- # (+ 2.95_X4 1.67_X5)))
39	(+ (/ 6.47 (+ (* (* 1.31_X2 0.67_X1) (- -11.12 #)) 11.36)) (+ (/ 6.47 (+ (+ (* 1.31_X2 1.64_X1) -10.49) 11.36)) (- # (+ 2.95_X4 1.67_X5))))
40	(+ (/ 6.47 (+ (* (* 1.31_X2 0.67_X1) (- -11.12 (* # 0.67_X1))) 11.36)) (+ (/ 6.47 (+ (+ (* 1.31_X2 1.64_X1) -10.49) 11.36)) #))
41	(+ (/ 6.47 #) (+ (/ 6.47 (+ (+ (* 1.31_X2 1.64_X1) -10.49) 11.36)) (- # (+ 2.95_X4 1.67_X5))))

Table 7.15: OSGP Friedman-2 Gender Specific Most Frequent Schemas

Generation	Schema
1	(- -11.73 (+ (- -12.19 (/ (- # (- 1.53_x15 17.09)) 0.63_x15)) 1.54_x6))
2	(- -11.73 (+ (- -12.19 (/ (- # (- 1.53_x15 17.09)) 0.63_x15)) 1.54_x6))
3	(- (+ -6.98 (/ -0.15_x5 #)) (/ 2.52_x13 1.35_x6))
4	(- (+ -6.98 (/ -0.15_x5 #)) (/ 2.52_x13 1.35_x6))
5	(- (- # (/ (* 1.94_x1 #) 0.44_x6)) -5.52)
6	(- (- # (/ (* 1.94_x1 #) 0.44_x6)) -5.52)
7	(- (- # (/ (* 1.94_x1 #) 0.44_x6)) -5.52)
8	(/ (/ (- 1.05_x4 (* (+ -2.00 1.96_x23) (- (+ (* 1.27_x6 14.27) (+ 1.28_x24 0.29_x16)) 1.07_x16))) 1.96_x1) #)
9	(/ (/ (- 1.05_x4 (* (+ -2.00 1.96_x23) #)) 1.96_x1) #)
10	(* -15.08 (+ (- (- 2.31_x6 #) (/ 0.97_x22 1.70_x15)) (+ 3.16_x23 (* 19.91 (/ (+ 3.39_x24 (- # (- 1.89_x9 0.55_x1))) 1.04_x1))))))
11	(/ (/ (- # (* (+ -2.00 1.96_x23) (- (+ (+ -0.81 1.93_x6) (+ 1.28_x24 0.29_x16)) 1.07_x16))) 1.96_x1) (+ 7.74 #))
12	(/ (/ (- 1.05_x4 (* (+ -2.00 1.96_x23) (- (+ (+ -0.81 1.93_x6) (+ 1.28_x24 0.29_x16)) 1.07_x16))) 1.96_x1) #)
13	(/ (/ (- 1.05_x4 (* (+ -2.00 1.96_x23) (- (+ (+ -0.81 1.93_x6) (+ 1.28_x24 0.29_x16)) 1.07_x16))) 1.96_x1) #)
14	(/ (/ (- 1.05_x4 (* (+ -2.00 1.96_x23) (- (+ (* 1.27_x6 14.27) (+ 1.28_x24 0.29_x16)) 1.07_x16))) 1.96_x1) #)
15	(/ (/ (- 1.05_x4 (* (+ -2.00 1.96_x23) (- (+ (* 1.27_x6 14.27) #) 1.07_x16))) 1.96_x1) (+ (/ 2.24_x4 (/ -0.20_x6 0.28_x12)) (- # (- 1.89_x9 0.55_x1))))
16	(/ # (+ (/ 2.24_x4 (/ -0.20_x6 0.28_x12)) (- # (- 1.89_x9 0.55_x1))))
17	(/ (/ (- (- 3.48_x12 -0.64_x12) (* (+ -2.00 1.96_x23) (- (+ (* 1.27_x6 14.27) (* 1.54_x15 (/ 2.32_x1 15.71))) 1.07_x16))) 1.96_x1) #)
18	(/ (/ (- (- 3.48_x12 -0.64_x12) (* (+ -2.00 1.96_x23) (- (+ (* 1.27_x6 14.27) (* 1.54_x15 (/ 2.32_x1 15.71))) 1.07_x16))) 1.96_x1) #)
19	(/ (/ (- # (* (+ 0.55_x1 1.96_x23) (- (+ (* 1.27_x6 14.27) (* 1.54_x15 (/ 2.32_x1 15.71))) 1.07_x16))) 1.96_x1) #)
20	(/ (/ (- (- 3.48_x12 -0.64_x12) (* (+ -2.00 1.96_x23) (- (+ (* 1.27_x6 14.27) (* 1.54_x15 (/ 2.32_x1 15.71))) 1.96_x23))) 1.96_x1) #)
21	(/ (/ (- (- 3.48_x12 -0.64_x12) (* (+ -2.00 1.96_x23) (- (+ (* 1.27_x6 14.27) (* 1.54_x15 (/ 2.32_x1 15.71))) 1.96_x23))) 1.96_x1) #)
22	(/ (/ # 1.96_x1) (+ # (- (+ (/ 0.97_x22 1.70_x15) (- 0.53_x13 (* 12.62 4.83))) (/ -4.04 #))))
23	(/ (/ (- (- 3.48_x12 -0.64_x12) (* (+ 12.62 1.96_x23) (- (+ (* 1.27_x6 14.27) (* 1.54_x15 (/ 2.32_x1 15.71))) 1.96_x23))) 1.96_x1) #)
24	(/ (/ (- # (* (+ -2.00 1.96_x23) (- (+ (* 1.27_x6 14.27) (* 1.54_x15 (/ 2.32_x1 15.71))) 1.96_x23))) 1.96_x1) #)
25	(/ (/ (- # (- # (* (+ 14.27 1.96_x23) (- (+ (* 1.27_x6 12.56) (* 1.54_x15 (/ 2.32_x1 15.71))) 1.96_x23)))) 1.96_x1) (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 (* 12.62 4.83)) (/ -4.04 #))))
26	(/ (/ (- # #) 1.96_x1) (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 (* 12.62 4.83)) #)))
27	(/ (/ (- # (- # (* (+ 14.27 1.96_x23) (- (+ (* 1.27_x6 12.56) (* 1.54_x15 (/ 2.32_x1 15.71))) 1.96_x23)))) 1.96_x1) (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 (* 12.62 4.83)) (/ 0.83_x24 (* 1.27_x6 14.27)))))
28	(/ (/ (- (- 3.48_x12 -0.64_x12) (* (+ 14.27 1.96_x23) (- (+ # (* 1.54_x15 (/ 2.32_x1 15.71))) 1.96_x23))) 1.96_x1) #)
29	(/ (/ # 1.96_x1) (+ (/ 0.83_x24 (- 3.44_x22 -0.64_x12)) #))
30	(/ (/ (- # (* (+ 14.27 1.96_x23) (- (+ (* 1.27_x6 12.56) (* 1.54_x15 (/ 2.32_x1 15.71))) 1.96_x23))) 1.96_x1) #)
31	(/ # (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 (* 12.62 4.83)) (/ -4.04 1.37_x2))))
32	(/ (/ # 1.96_x1) (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 (* 12.62 4.83)) (/ -4.04 (- 1.04_x1 0.53_x6)))))
33	(/ # (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 (* 12.62 4.83)) (/ -4.04 (- 1.04_x1 0.53_x6)))))
34	(/ # (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 (* 12.62 4.83)) (/ -4.04 (- 1.04_x1 0.53_x6)))))
35	(/ (/ # 1.96_x1) (+ (/ 0.82_x8 (+ -10.48 -6.27)) #))
36	(/ (/ # 1.96_x1) (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 #) (/ -6.27 (- 1.04_x1 0.53_x6)))))
37	(/ # (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 (* 12.62 4.83)) (/ -6.27 (- 1.04_x1 0.53_x6)))))
38	(/ (/ # 1.96_x1) (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 #) (/ -6.27 (- 1.04_x1 0.53_x6)))))
39	(/ (/ # 1.96_x1) (+ (/ 0.82_x8 (+ -10.48 -6.27)) (- (- 0.53_x13 (* 12.62 4.83)) (/ -6.27 (- 1.04_x1 0.53_x6)))))
40	(/ (/ (- # (+ # (* (+ 0.22_x16 1.96_x23) (- (+ (* 1.27_x6 11.20) (* 1.54_x15 (/ 2.32_x1 18.06))) #)))) 1.96_x1) (+ (/ 0.82_x8 (- 1.04_x1 0.53_x6)) #))
41	(/ (/ (- # (+ # (* (+ 0.22_x16 1.96_x23) (- (+ (* 1.27_x6 11.20) (* 1.54_x15 (/ 2.32_x1 18.06))) 1.96_x23)))) 1.96_x1) (+ (/ 0.82_x8 (- 1.04_x1 0.53_x6)) (- (- 0.53_x13 #) (/ -4.04 (- 1.04_x1 0.53_x6)))))
42	(/ (/ # 1.96_x1) (+ (/ 0.82_x8 (- 1.04_x1 0.53_x6)) (- (- 0.53_x13 #) (/ -4.04 (- 1.04_x1 0.53_x6)))))
43	(/ (/ # 1.96_x1) (+ (/ 0.82_x8 (- 1.04_x1 0.53_x6)) #))
44	(/ # (+ (/ 0.82_x8 (- 1.04_x1 0.53_x6)) #))

Table 7.16: OSGP Tower Proportional Most Frequent Schemas

Generation	Schema
1	(- (/ -8.76 (+ (+ -13.37 -0.02_x21) #)) #)
2	(- (/ -8.76 (+ (+ -13.37 -0.02_x21) #)) #)
3	(* 2.22_x6 (+ (* (- 8.94 0.60_x3) 0.29_x6) #))
4	(/ (- 19.28 (* 0.22 (* 1.49_x6 -17.31)))) (- 0.73_x20 #))
5	(/ (- 19.28 (* 0.22 (* 1.49_x6 -17.31)))) (- 0.73_x20 (/ 0.81_x2 #)))
6	(* (* (+ -7.57 17.60) -0.32_x23) (/ # 1.05_x1))
7	(* (* (+ -7.57 17.60) -0.32_x23) (/ # 1.05_x1))
8	(* (* (+ -7.57 17.60) -0.32_x23) (/ # 1.05_x1))
9	(* (* (+ -7.57 17.60) -0.32_x23) (/ # 1.05_x1))
10	(* (* (+ -7.57 17.60) -0.32_x23) (/ # 1.05_x1))
11	(* (* (+ -7.57 17.60) -0.32_x23) (/ # 1.05_x1))
12	(* (* (+ -7.57 17.60) -0.32_x23) (/ (/ (/ (+ -0.60_x6 (- -16.57 -0.38)) (* 1.85_x16 (+ 19.17 -2.60))) #) 1.05_x1))
13	(* (* (+ (/ 1.65_x3 8.73) 17.60) -0.32_x23) (/ (/ (/ # (* -6.23 (+ 7.89 (+ 0.08_x12 -12.26))))) (* # (- -4.91 (+ 1.49_x4 17.82)))) 1.05_x1))
14	(* (* (+ (/ 1.29_x4 -6.66) 19.20) -0.32_x23) (/ # 1.05_x1))
15	(* (* (+ (/ 1.65_x3 8.73) 17.60) -0.32_x23) (/ (/ (/ (+ -0.60_x6 #) (* -6.23 (/ (- 1.04_x13 #) -16.53))) (* # (- -4.91 (+ 1.49_x4 17.82)))) 1.05_x1))
16	(* (* (+ (/ 1.65_x3 8.73) 17.60) -0.32_x23) (/ (/ (/ (+ -0.60_x6 #) (* -6.23 (/ (- 1.04_x13 #) -16.53))) (* # (- -4.91 (+ 1.49_x4 17.82)))) 1.05_x1))
17	(* (* (+ # 17.60) -0.32_x23) (/ (/ (/ (- 1.95_x6 (/ 3.32_x20 (+ 7.89 -16.72))) (* 2.22_x13 -14.09)) (* (- 0.43_x20 (/ (+ (+ 3.43_x6 3.33_x25) 1.30_x7) -0.32_x23)) (/ (/ (- 2.22_x9 7.27) -0.23_x6) 0.39_x15))) 1.05_x1))
18	(* (* # -0.32_x23) (/ (/ (/ (- 1.95_x6 (/ 3.32_x20 (+ 7.89 -16.72))) (* 2.22_x13 -14.09)) (* (- 0.43_x20 (/ (+ (+ 3.43_x6 3.33_x25) 1.30_x7) -0.32_x23)) (/ (/ (- 2.22_x9 7.27) -0.23_x6) 0.39_x15))) 1.05_x1))
19	(* (* (+ # 17.60) -0.32_x23) (/ (/ (/ (- 1.95_x6 (/ 3.32_x20 (+ 7.89 -16.72))) #) (* (- 0.43_x20 (/ (+ (+ 3.43_x6 3.33_x25) 1.30_x7) -0.32_x23)) (/ (/ (- 2.22_x9 7.27) -0.23_x6) 0.39_x15))) 1.05_x1))
20	(* (* (+ (/ # #) 17.60) -0.32_x23) (/ (/ (/ (- 1.95_x6 #) (* 2.22_x13 -14.09)) (* (- 0.43_x20 (/ (+ (+ 3.43_x6 3.33_x25) 1.30_x7) -0.32_x23)) (/ (/ (- 2.22_x9 7.27) -0.23_x6) 0.39_x15))) 1.05_x1))
21	(* (* 0.11_x15 -0.32_x23) (/ (/ (/ (+ -0.60_x6 (+ -0.60_x6 (- # 9.87))) (* -6.23 (+ 19.55 #)))) (* (- 1.44_x20 (+ -0.08_x8 0.95_x1)) (- 0.43_x20 (+ (/ 1.29_x4 -6.66) 19.20)))) 1.05_x1))
22	(* (* (+ -7.57 #) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (* 1.91_x20 #)) 1.05_x1))
23	(* # (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) #) 1.05_x1))
24	(* (* (+ -7.57 (- 0.43_x20 #)) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) #) 1.05_x1))
25	(* (* (+ -7.57 (- 0.43_x20 #)) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) #) 1.05_x1))
26	(* (* # -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) #) 1.05_x1))
27	(* # (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) #) 1.05_x1))
28	(* (* (+ -7.57 #) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 0.43_x20 #)) 1.05_x1))
29	(* (* (+ -7.57 (- 0.43_x20 (/ (- 2.22_x9 #) -0.23_x6))) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) #) 1.05_x1))
30	(* (* # -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (* -6.22 (* (- 1.04_x13 (/ 0.81_x2 1.05_x1)) 8.24))) 1.05_x1))
31	(* (* (+ (/ 0.81_x2 1.05_x1) (- 0.43_x20 #))) -0.32_x23) #)
32	(* (* (+ (/ 0.81_x2 1.05_x1) (- 0.43_x20 #))) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) #) 1.05_x1))
33	(* (* (+ -7.57 (- 0.43_x20 (- 13.46 (/ -9.98 (+ # #)))))) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (* 1.91_x20 (- -13.95 (- 13.46 #))))) 1.05_x1))
34	(* # (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (* -6.22 #)) 1.05_x1))
35	(* (* (+ (/ 0.81_x2 1.05_x1) #) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 #))) 1.05_x1))
36	(* (* (+ (/ 0.81_x2 1.05_x1) #) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 (/ (- 2.22_x9 7.27) #)))) 1.05_x1))
37	(* # (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 (/ (- 2.22_x9 7.27) (+ -0.60_x6 1.05_x1)))))) 1.05_x1))
38	(* (* # -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 (/ (- 2.22_x9 7.27) (+ -0.60_x6 1.05_x1)))))) 1.05_x1))
39	(* (* # -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 (/ # (+ -0.60_x6 1.05_x1)))))) 1.05_x1))
40	(* (* # -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 (/ (- 2.22_x9 7.27) (+ -0.60_x6 1.05_x1)))))) 1.05_x1))

Table 7.17: OSGP Tower Gender Specific Most Frequent Schemas – Part 1

Selector	Poly-10	Pagie-1	Friedman-2	Tower
Proportional	496014	501168	524916	551336
Gender specific	617276	731684	691742	822984

Table 7.19: OSGP Average Evaluated Solutions

Generation	Schema
41	(* (* # -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 (/ (- 2.22_x9 7.27) (+ -0.60_x6 1.05_x1)))))) 1.05_x1))
42	(* (* # -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 (/ (- 2.22_x9 7.27) (+ -0.60_x6 1.05_x1)))))) 1.05_x1))
43	(* # (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 (/ (- 2.22_x9 7.27) (+ -0.60_x6 1.05_x1)))))) 1.05_x1))
44	(* # (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 (/ (- 2.22_x9 7.27) (+ -0.60_x6 1.05_x1)))))) 1.05_x1))
45	(* (* # -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.26) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.35_x12 (/ (- 2.22_x9 7.27) (+ -0.60_x6 1.05_x1)))))) 1.05_x1))
46	(* (* (+ (+ 0.08_x12 -12.26) (+ (/ 0.81_x2 1.05_x1) #)) -0.32_x23) (/ # 1.05_x1))
47	(* (* (+ -7.57 (+ (/ 0.81_x2 1.05_x1) (- 0.43_x20 (/ (+ -0.60_x6 1.05_x1) 1.63_x2)))) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ # 0.39_x15)) (- 1.04_x13 #)) 1.05_x1))
48	(* (* # -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.22) -0.23_x6) 0.39_x15)) (- 1.04_x13 (- 0.33_x16 (/ (+ 2.22_x9 #) (+ -0.60_x6 1.05_x1)))))) 1.05_x1))
49	(* (* (+ (/ 0.81_x2 1.05_x1) (- 0.43_x20 (+ (- 5.90 #) 0.43_x10))) -0.32_x23) (/ # 1.05_x1))
50	(* (* (+ (/ 0.81_x2 1.05_x1) (- 0.43_x20 (+ # 0.43_x10))) -0.32_x23) (/ # 1.05_x1))
51	(* (* (+ (/ 0.81_x2 1.05_x1) (- 0.43_x20 #)) -0.32_x23) (/ (/ (/ (+ -0.60_x6 -0.32_x23) (/ (/ (+ 0.08_x12 -12.22) -0.23_x6) 0.39_x15)) (- 1.04_x13 #)) 1.05_x1))
52	(* (* (+ (/ 0.81_x2 1.05_x1) (- 0.43_x20 (+ (- 5.90 (+ 0.08_x14 #)) 0.43_x10))) -0.32_x23) (/ # 1.05_x1))

Table 7.18: OSGP Tower Gender Specific Most Frequent Schemas – Part 2

Replacement rule	Poly-10	Pagie-1	Friedman-2	Tower
$f(x) = x$	2597004	4202342	4730560	4724672
$f(x) = \tanh(x)$	2289008	4499724	4757191	4769932
$f(x) = \tanh(2x)$	2492327	4668455	4173848	4986492
$f(x) = \tanh(3x)$	2406219	4729180	4693742	4943163
$f(x) = \tanh(4x)$	2777475	3859863	4732257	4941368
$f(x) = 1 - \sqrt{1-x}$	2519187	4670874	4333620	4510939

(a) Minimum Phenotypic Similarity 90%

Replacement rule	Poly-10	Pagie-1	Friedman-2	Tower
$f(x) = x$	2643261	4623542	4429457	5009276
$f(x) = \tanh(x)$	2166972	4502617	4218980	4904499
$f(x) = \tanh(2x)$	2338860	4703514	4780838	4874290
$f(x) = \tanh(3x)$	2655113	4756972	4656273	4875607
$f(x) = \tanh(4x)$	2981839	4296971	4518030	4868244
$f(x) = 1 - \sqrt{1-x}$	1846627	3998111	4250781	4655143

(b) Minimum Phenotypic Similarity 95%

Table 7.20: OSGP-S Adaptive Replacement Average Solution Evaluations

Minimum Phenotypic Similarity	Poly-10	Pagie-1	Friedman-2	Tower
90%	3483716	4731112	5003719	5003015
95%	3202283	5004201	5004529	5003516

(a) Minimum Schema Length = 10

Minimum Phenotypic Similarity	Poly-10	Pagie-1	Friedman-2	Tower
90%	3544254	4857343	5004209	5004219
95%	3766646	4779941	5004358	5004258

(b) Minimum Schema Length = 25

Table 7.21: OSGP-S Fixed Replacement Average Solution Evaluations

Glossary of Biological Terms

allele	An allele is a concrete value out of a number of alternate values of the same gene or the same genetic locus.
colonial organism	A colonial organism could be described as a "collective entitiy" made up of many individual organisms of the same species living together, such as ants or bees.
DNA	DNA or deoxyribonucleic acid is a macromolecule which holds the genetic information that guides the development of biological organisms. Structurally the DNA consists of two strands of polynucleotides which are in turn made of nucleotides that can be of four types, depending on their nitrogen base: cytosine (C), guanine (G), adenine (A) or thymine (T). The nucleobases can bond with each other according to specific rules: A with T and C with G. Nucleotide sequences representing distinct units of information that encode for specific functions are called genes.
effective population size	Geneticist Sewall Wright introduced the concept of effective population size as "the number of breeding individuals in an idealised population that would show the same amount of dispersion of allele frequencies under random genetic drift or the same amount of inbreeding as the population under consideration". Simply put, the effective population size is the size of an ideal population that would lose heterozygosity at a rate equal to that of the observed population. Empirically, the effective population size can be estimated as the number of individuals in a population who contribute offspring to the next generation.
environment	In the context of GP, by environment we understand the fitness landscape on which the organisms have to compete for survival. As such, the environment may change for example when the training data for the algorithm changes.

epistasis

Epistasis refers to the situation where the effects of genes are influenced by other genes from different loci on the chromosome. The fact that genes are context-dependent through epistatic effects is of particular importance for concepts such as evolvability and genotype-phenoytpe maps and plays an influential role in the dynamics of evolution.

evolvability

Adaptability represents the potential of systems, processes, populations or organisms to adapt. Evolutionary adaptability or evolvability refers to an organism's capacity to generate heritable phenotypic variation. In the context of populations, evolvability refers to a population's potential to evolve, to adapt to changing environmental conditions. (Altenberg, 1994b) defines evolvability as the ability of populations to produce variants fitter than any yet existing. Altenberg considers evolvability to be an emergent property of evolutionary systems due to selection. (Hu and Banzhaf, 2010) define evolvability as the capability of a system to generate adaptive phenotypic variation and to transmit it via an evolutionary process.

fitness

An organisms' fitness indicates how well it is adapted to the environment. In GP, fitness is usually expressed as a measure of how well the estimated data fits the target data of the algorithm. For this purpose, normally a correlation measure (e.g.,Pearson R^2 coefficient) or an error measure (e.g., the mean squared error) is used.

gene

When talking about living organisms, a *gene* is defined as a molecular unit of heredity. Technically speaking, a gene is a DNA sequence which encodes information pertaining to a specific cell function. In the context of genetic programming, our working definition of a *gene* is "a heritable unit that may influence a trait". As with their biological counterparts, GP genes are context-dependent, subject to epistasis and can be characterized by the same properties such as robustness and evolvability.

gene flow

Gene flow represents the transfer of genes due to the movement of individuals between populations, and can be an important source of genetic variation.

genetic drift

Genetic or allelic drift is considered to be one of the main processes in evolution and refers to the variation in allelic frequencies in a population due to the random sampling of organisms. The chance element is represented by the probability that an individual survives and reproduces. The effects of genetic drift are inversely proportional to allele frequencies.

genetic linkage

Genetic linkage refers to the idea that linked genes that sitting together on the chromosome are likely to be inherited together. Geneticists can use linkage to find the location of a gene on a chromosome and can map gene distances by looking at how often different genes are inherited together.

genotype

The genotype can be taken to mean:

a The entire set of genes in an organism.

b A set of allelles that determines the expression of a particular characteristic or trait.

In this thesis, the first meaning will be favored. In the context of genetic programming all the genotype-altering operations actually operate on the entire collection of genes. Furthermore, in GP inheritance is in no way limited to particular structures or genes. Generally speaking, any basic gene or building block can be inherited.

heredity

Heredity is the passing of traits from ancestors to their descendants.

heterozygosity

If both alleles of a diploid organism are the same, the organism is homozygous at that locus. If they are different, the organism is heterozygous at that locus .

ideal population

An idealised population is a population which satisfies a number of simplifying assumptions. A common model, that of R.A. Fisher and Sewall Wright assumes ideal populations to be of constant size, with members that can mate and reproduce with any other member.

Lamarckism	Lamarckism or Lamarckian inheritance or soft inheritance is an idea developed by French biologist Jean-Baptiste Lamarck (1744-1829) who maintained that organisms can pass acquired traits to their offspring. Lamarck thought that during their lifetime individuals lose unnecessary traits and develop useful ones that can be inherited by their offspring. Lamarck's view on evolution is in contradiction with Darwin's theory of natural selection which explains adaptation as random variation acted on by selective pressure.
locus	In the field of genetics, a locus (plural *loci*) represents the specific location of a gene or DNA sequence on the chromosome.
macroevolution	In contrast to microevolution, macroevolution describes evolution at a bigger scale, studying changes that occur at or above the level of species.
macromolecule	A macromolecule is a large molecule composed of smaller subunits called monomers (smaller molecules that bind to each other to form polymers). DNA is the most important example of a macromolecule specialized in the encoding of information.
microevolution	Microevolution describes evolution at population level and refers to changes allele frequencies due to mutation, selection, gene flow and genetic drift.
orthogenesis	Orthogenesis is a hypothesis proposed by biologist Wilhelm Haacke in 1893 and later popularized by German zoologist Theodor Eimer. The orthogenetic hypothesis claims that evolution takes place continuously and "in a straight line", under the effects of internal and external factors. In other words it claims that variation is not random but directed towards fixed goals.

phenotype

Similarly to the definition of the genotype, the phenotype can have more than one meaning:

a The concrete appearance of an organism as a result of the interaction of its genotype and the environment.

b The expression of a particular trait, according to the individual's genetic makeup and environment.

In evolutionary search methods, the phenotype of an individual is given by a numerical value which indicates its fitness.

phenotypic plasticity

Phenotypic plasticity represents an organism's ability to change its phenotype in response to changes in the environment. According to (Kitano, 2004), phenotypic plasticity represents another aspect of robustness, as it enables organisms to robustly adapt to a changing environment.

polymorphism

Polymorphism represents the presence of different phenotypes in the same population of a species. Polymorphism is heritable and controlled by natural selection.

punctuated equilibria

Punctuated equilibria is a biological theory about evolution which claims that once formed, populations tend to stabilize and exhibit little evolutionary change, remaining in a state of evolutionary stasis (as observed in fossil records). The theory proposes that when subsequent significant evolutionary change occurs, it is generally restricted to branching speciation events.

takeover time

The time required by selection to fill up the population with copies of the best individual in the initial generation.

trait

A trait is a distinct variant of an organism's phenotypic character, that may be inherited, be environmentally determined or be a combination of the two. An example of a phenotypic trait is hair color: there are underlying genes that control the hair color, but the actual hair color, the part we see, is the phenotype.

Glossary of Computational Terms

big O notation	In mathematics, big O notation describes the limiting behavior of functions (when the argument tends to ∞ or to some value). The letter "O" signifies the order of the function, ignoring constants or lower order terms. In computer science, big O notation is used to classify algorithms by their resource usage (time or memory) as a function of input size, specifically describing the worst-case scenario.
computational class NP	NP refers to non-deterministic polynomial time. The computational class NP can be defined as: (a) The class of problems solvable by a non-deterministic Turing machine in polynomial time (b) The class of problems verifiable by a deterministic Turing machine in polynomial time Definitions (a) and (b) are equivalent. The class NP contains all the problems in P since one can verify any instance of the problem by solving it.
computational class P	The class of problems that can be solved by a deterministic Turing machine in polynomial time. (Cobham, 1965) asserts that computational problems can be feasibly computed if they belong to complexity class P.
feasible solution	In mathematics and computer science, a *feasible* or *candidate* solution for a given problem is a solution that satisfies all the constraints..
Markov's inequality	Markov's inequality offers information about the probability that the value of random variable X is "far" from its expectation. **Theorem**: For a non-negative random variable $X : \Omega \to \mathbb{R}$, where $X(s) \geq 0$ for all $s \in \Omega$, for any positive real number $a > 0$, $P(X \geq a) \leq \frac{E(X)}{a}$.

NP-complete	A decision problem is NP-complete when it is both in NP and NP-hard.
NP-hard	NP-hard (Non-deterministic Polynomial-time hard) is a class of problems that are "at least as hard as the hardest problems in NP". A problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H.
polynomial time	An algorithm is said to be of polynomial time if the amount of time taken to run as a function of the input size is upper bounded by a polynomial expression, i.e., $T(n) = O(n^k)$ for some constant k.
pseudo-polynomial time	An algorithm is said to be running in pseudo-polynomial time if it is polynomial in the *numeric value* of the input, but it is exponential in the *length* of the input – the number of bits required to represent it. For example, an algorithm which checks if a number n is prime by testing whether no number in $\{2, 3, \ldots, \sqrt{n}\}$ divides n evenly can take up to $\sqrt{n} - 1$ divisions which is sublinear in the *value* of n but exponential in the *size* of n (which is about $\log(n)$).
PTAS	A polynomial-time approximation scheme (PTAS) is a type of approximation algorithm for optimization problems (usually used to find approximate solutions to NP-hard problems) .
Turing machine	Hypothetical computation device that can manipulate symbols stored on an unbounded strip of tape, according to a table of rules. The Turing machine can read or write one symbol at a time and store its state in a special state register. Turing machines can be used to simulate the logic of any computer algorithm and to describe their behavior.
Turing machine (deterministic)	A Turing machine whose set of rules prescribes at most one action to be performed for any given situation.
Turing machine (non-deterministic)	A Turing machine whose set of rules prescribes more than one action for a given situation. This means that given a branching point, the choice made is non-deterministic.

Turing machine (probabilistic)	A Turing machine which chooses randomly between the available state transitions at each point, according to some probability distribution. When the transition probabilities are equal, the probabilistic Turing machine can also be described as a deterministic Turing machine with an added tape full of random bits called the random tape.

Bibliography

E.H.L. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization*. Discrete Mathematics and Optimization. Wiley, Chichester, UK, June 1997. ISBN 0-471-94822-5.

Michael Affenzeller and Stefan Wagner. *A Self-adaptive Model for Selective Pressure Handling within the Theory of Genetic Algorithms*, pages 384–393. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-45210-2. doi: 10.1007/978-3-540-45210-2_35.

Michael Affenzeller, Stephan Winkler, Stefan Wagner, and Andreas Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Chapman & Hall/CRC, 1st edition, 2009. ISBN 1584886293, 9781584886297.

P. Alberch. From genes to phenotype: dynamical systems and evolvability. *Genetica*, 84(1): 5–11, 1991. ISSN 0016-6707. doi: 10.1007/BF00123979.

D. L. Alderson and J. C. Doyle. Contrasting views of complexity and their implications for network-centric infrastructures. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 40(4):839–852, July 2010. ISSN 1083-4427. doi: 10.1109/TSMCA.2010.2048027.

Laurent Alonso and Rene Schott. *Random Generation of Trees: Random Generators in Computer Science*. Kluwer Academic Publishers, Norwell, MA, USA, 1995. ISBN 079239528X.

Lee Altenberg. Emergent phenomena in genetic programming. In Anthony V. Sebald and Lawrence J. Fogel, editors, *Evolutionary Programming — Proceedings of the Third Annual Conference*, pages 233–241, San Diego, CA, USA, 24-26 February 1994a. World Scientific Publishing. ISBN 981-02-1810-9.

Lee Altenberg. The evolution of evolvability in genetic programming. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 3, pages 47–74. MIT Press, 1994b.

Lee Altenberg. The schema theorem and price's theorem. In *FOUNDATIONS OF GENETIC ALGORITHMS*, pages 23–49. Morgan Kaufmann, 1995.

Lee Altenberg. Mathematics awaits: commentary on "genetic programming and emergence" by wolfgang banzhaf. *Genetic Programming and Evolvable Machines*, 15(1):87–89, 2014. ISSN 1389-2576. doi: 10.1007/s10710-013-9198-5.

Lauren W. Ancel and Walter Fontana. Plasticity, evolvability and modularity in rna. Working papers, Santa Fe Institute, 1999.

Peter J. Angeline. Subtree crossover: Building block engine or macromutation? In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 9–17, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

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

David Applegate, Robert Bixby, Vasek Chvátal, and William Cook. Tsp cuts which do not conform to the template paradigm. In *IN COMPUTATIONAL COMBINATORIAL OPTIMIZATION*, pages 261–303. Springer, 2001.

David Applegate, William Cook, and André Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS J. on Computing*, 15(1):82–92, January 2003. ISSN 1526-5528. doi: 10.1287/ijoc.15.1.82.15157.

Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, September 1998. ISSN 0004-5411. doi: 10.1145/290179.290180.

Anne Auger and Olivier Teytaud. Continuous lunches are free plus the design of optimal optimization algorithms. *Algorithmica*, 57(1):121–146, 2010. ISSN 0178-4617. doi: 10.1007/s00453-008-9244-5.

László Babai. Monte-Carlo algorithms in graph isomorphism testing. Technical report, Université de Montréal, 1979.

Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996. ISBN 0-19-509971-0.

Thomas Back, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, UK, 1st edition, 1997. ISBN 0750303921.

James E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, pages 14–21, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc. ISBN 0-8058-0158-8.

Sergio (Universitat Autònoma de Barcelona. Departament de Filologia Catalana) Balari and Guillermo Lorenzo González. Pere alberch's developmental morphospaces and the evolution of cognition. *Biological Theory*, 03(04):297–304, 2008.

Wolfgang Banzhaf. Genetic programming and emergence. *Genetic Programming and Evolvable Machines*, 15(1):63–73, 2014. ISSN 1389-2576. doi: 10.1007/s10710-013-9196-7.

L. Beadle and C.G. Johnson. Semantically driven crossover in genetic programming. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 111–116, June 2008. doi: 10.1109/CEC.2008.4630784.

Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. 1(1):3–52, May 2002. ISSN 1567-7818. doi: 10.1023/A:1015059928466.

Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms, 1995.

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003. ISSN 0360-0300. doi: 10.1145/937503.937505.

Walter Bohm and Andreas Geyer-Schulz. Exact uniform initialization for genetic programming. In Richard K. Belew and Michael Vose, editors, *Foundations of Genetic Algorithms IV*, pages 379–407, University of San Diego, CA, USA, 3–5 August 1996. Morgan Kaufmann. ISBN 1-55860-460-X.

Josh C. Bongard. A probabilistic functional crossover operator for genetic programming, 2010.

E.K. Burke, S. Gustafson, G. Kendall, and N. Krasnogor. Is increased diversity in genetic programming beneficial? an analysis of lineage selection. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 2, pages 1398–1405 Vol.2, Dec 2003. doi: 10.1109/CEC.2003.1299834.

Thomas Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *In Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 57–62. IEEE Press, 1994.

Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1991.

Thomas Bäck, Günter Rudolph, and Hans paul Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In *In Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 11–22, 1993.

Lynn Caporale. *Darwin In the Genome: Molecular Strategies in Biological Evolution*. McGraw-Hill, October 2002. ISBN 0071378227.

Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964. doi: 10.1287/opre.12.4.568.

Alan Cobham. The Intrinsic Computational Difficulty of Functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science, proceedings of the second International Congress, held in Jerusalem, 1964*, Amsterdam, 1965. North-Holland.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. doi: 10.1145/800157.805047.

David Corne and Joshua Knowles. Some multiobjective optimizers are better than others. In *IN IEEE CONGRESS ON EVOLUTIONARY COMPUTATION*, pages 2506–2512, 2003.

Peter A. Corning. The re-emergence of "emergence": A venerable concept in search of a theory. *COMPLEXITY*, 7(6):2002, 2002.

G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6): 791–812, 1958. doi: 10.1287/opre.6.6.791.

Jose Davila-Velderrain and Elena R Alvarez-Buylla. Bridging the genotype and the phenotype: Towards an epigenetic landscape approach to evolutionary systems biology. *bioRxiv*, 2014. doi: 10.1101/004218.

J. Arjan G. M. de Visser, Joachim Hermisson, Günter P. Wagner, Lauren Ancel Meyers, Homayoun Bagheri-Chaichian, Jeffrey L. Blanchard, Lin Chao, James M. Cheverud, Santiago F. Elena, Walter Fontana, Greg Gibson, Thomas F. Hansen, David Krakauer, Richard C. Lewontin, Charles Ofria, Sean H. Rice, George von Dassow, Andreas Wagner, and Michael C. Whitlock. Perspective: Evolution and detection of genetic robustness. *Evolution*, 57(9):1959–1972, 2003. ISSN 1558-5646.

Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B*, 26(1):29–41, 1996.

Stefan Droste, Thomas Jansen, and Ingo Wegener. Perhaps not a free lunch but at least a free appetizer, 1998.

Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+ 1) evolutionary algorithm. *Theor. Comput. Sci.*, 276(1-2):51–81, April 2002. ISSN 0304-3975. doi: 10.1016/S0304-3975(01)00182-7.

Marc Ebner, Mark Shackleton, and Rob Shipman. How neutral networks influence evolvability. *Complex.*, 7(2):19–33, November 2001. ISSN 1076-2787. doi: 10.1002/cplx.10021.

Roger Eckhardt. Stan Ulam, John von Neumann, and the Monte Carlo Method. *Los Alamos Science*, pages 131–143, 1987.

Anikó Ekárt and Sandor Z. Németh. A metric for genetic programs and fitness sharing. In *Proceedings of the European Conference on Genetic Programming*, pages 259–270, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-67339-3.

Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the tsp: Extended abstract. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1295–1304, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5.

Thomas A. Feo and Mauricio G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67 – 71, 1989. ISSN 0167-6377. doi: 10.1016/0167-6377(89)90002-3.

Candida Ferreira and U Gepsoft. What is gene expression programming, 2008.

David B. Fogel. *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling*. Ginn Press, 1991. ISBN 0536579431.

David B. Fogel and Lawrence J. Fogel. An introduction to evolutionary programming. In Jean-Marc Alliot, Evelyne Lutton, Edmund Ronald, Marc Schoenauer, and Dominique Snyers, editors, *Artificial Evolution*, volume 1063 of *Lecture Notes in Computer Science*, pages 21–33. Springer Berlin Heidelberg, 1996. ISBN 978-3-540-61108-0. doi: 10.1007/3-540-61108-8_28.

Lawrence J Fogel, Peter J Angeline, and David B Fogel. Approach to self-adaptation on finite state machines. In *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, volume 355. MIT Press, 1995.

L.J. Fogel. *On the Organization of Intellect*. University of California, Los Angeles - Engineering, 1964.

Walter Fontana and Peter Schuster. Continuity in evolution: On the nature of transitions. *Science*, 280(5368):1451–1455, 1998. doi: 10.1126/science.280.5368.1451.

Nicholas Geard, Janet Wiles, Jennifer Hallinan, Bradley Tonkes, and Ben Skellett. A comparison of neutral landscapes - nk, nkp and nkq, 2002.

Andreas Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, volume 3 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag, Heidelberg, 2nd revised edition, 1996.

Fred Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977. ISSN 1540-5915. doi: 10.1111/j.1540-5915.1977.tb01074.x.

Fred Glover. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 13(5):533–549, May 1986. ISSN 0305-0548. doi: 10.1016/0305-0548(86)90048-1.

Fred Glover. A template for scatter search and path relinking. pages 13–54. Springer, 1998.

David E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989, 1989.

David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. ISBN 0201157675.

David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. 1991.

Jeffrey Goldstein. Emergence as a construct: History and issues. *Emergence*, 1(1):49–72, 1999. doi: 10.1207/s15327000em0101_4.

Michaela Götz, Christoph Koch, and Wim Martens. Efficient algorithms for descendant-only tree pattern queries. *Inf. Syst.*, 34(7):602–623, November 2009. ISSN 0306-4379. doi: 10.1016/j.is.2009.03.010.

David Greenhalgh and Stephen Marshall. Convergence criteria for genetic algorithms. *SIAM Journal on Computing*, 30(1):269–282, 2000. doi: 10.1137/S009753979732565X.

John J. Grefenstette and James E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 20–27, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-006-3.

Peter Grunwald and Paul Vitányi. Shannon information and kolmogorov complexity. *arXiv preprint cs/0410002*, 2004.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9:159–195, 2001.

Nikolaus Hansen, Dirk V Arnold, and Anne Auger. Evolution strategies. 2013.

Richard F. Hartl. A global convergence proof for a class of genetic algorithms. Technical report, 1990.

Eric J Hayden, Evandro Ferrada, and Andreas Wagner. Cryptic genetic variation promotes rapid evolutionary adaptation in an rna enzyme. *Nature*, 474(7349):92–95, 2011.

Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM National Meeting*, ACM '61, pages 71.201–71.204, New York, NY, USA, 1961. ACM. doi: 10.1145/800029.808532.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

Paul Holmes and Peter J. Barclay. Functional languages on linear chromosomes. In *Proceedings of the 1st Annual Conference on Genetic Programming*, pages 427–427, Cambridge, MA, USA, 1996. MIT Press. ISBN 0-262-61127-9.

Holger Hoos and Thomas Sttzle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558608729.

Gregory S. Hornby. Alps: the age-layered population structure for reducing the problem of premature convergence. In *GECCO2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, volume 1, pages 815–822, ACM*, Seattle, Washington, USA., 2006. Press.

Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, April 1974. ISSN 0004-5411. doi: 10.1145/321812.321823.

Ting Hu and Wolfgang Banzhaf. *Evolvability and acceleration in evolutionary computation*. Memorial University, 2008.

Ting Hu and Wolfgang Banzhaf. Evolvability and speed of evolutionary algorithms in light of recent developments in biology, 2010.

Ting Hu and Wolfgang Banzhaf. Quantitative analysis of evolvability using vertex centralities in phenotype network. In Tobias Friedrich, editor, *GECCO '16: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, page fp503, Denver, USA, 20-24 July 2016a. ACM. Nominated for best paper.

Ting Hu and Wolfgang Banzhaf. Neutrality, robustness and evolvability in genetic programming. In Rick Riolo, William P. Worzel, and Mark Kotanchek, editors, *Genetic Programming Theory and Practice XIV*, Genetic and Evolutionary Computation. Springer, 2016b.

J. Huxley, editor. *Evolution: The Modern Synthesis*. Allen and Unwin, London, 1942.

Christian Igel and Marc Toussaint. Neutrality and self-adaptation. *Natural Computing*, 2: 2003, 2003a.

Christian Igel and Marc Toussaint. On classes of functions for which no free lunch results hold. *Inf. Process. Lett.*, 86(6):317–321, June 2003b. ISSN 0020-0190. doi: 10.1016/S0020-0190(03)00222-9.

David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study in local optimization, 1995.

Richard M. Karp. Reducibility among combinatorial problems. In RaymondE. Miller, JamesW. Thatcher, and JeanD. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972. ISBN 978-1-4684-2003-6. doi: 10.1007/978-1-4684-2001-2_9.

S Kauffman and S Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of theoretical biology*, 128(1):11, 1987.

Maarten Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 70–82, Essex, 14-16 April 2003. Springer-Verlag. ISBN 3-540-00971-X. doi: doi:10.1007/3-540-36599-0_7.

Hans Kellerer, Ulrich Pferschy, and MariaGrazia Speranza. An efficient approximation scheme for the subset-sum problem. In HonWai Leong, Hiroshi Imai, and Sanjay Jain, editors, *Algorithms and Computation*, volume 1350 of *Lecture Notes in Computer Science*, pages 394–403. Springer Berlin Heidelberg, 1997. ISBN 978-3-540-63890-2. doi: 10.1007/3-540-63890-3_42.

Hans Kellerer, Renata Mansini, Ulrich Pferschy, and Maria Grazia Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *Journal of Computer*

and System Sciences, 66(2):349 – 370, 2003. ISSN 0022-0000. doi: http://dx.doi.org/10.1016/S0022-0000(03)00006-0.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995. doi: 10.1109/ICNN.1995.488968.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, 1983.

Hiroaki Kitano. Biological robustness. *Nature Reviews Genetics*, 5(11):826–837, 2004.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5.

J.R. Koza. *Genetic Programming III: Darwinian Invention and Problem Solving*. A Bradford book. Morgan Kaufmann, 1999. ISBN 9781558605435.

W. B. Langdon. Size fair and homologous tree genetic programming crossovers. In *Genetic Programming and Evolvable Machines*, pages 95–119. Morgan Kaufmann, 1999.

W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. In *In*, pages 37–48. Springer-Verlag, 1998.

William B. Langdon, Tery Soule, Riccardo Poli, and James A. Foster. Advances in genetic programming. chapter The Evolution of Size and Shape, pages 163–190. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-19423-6.

J. Lee. *A First Course in Combinatorial Optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2004. ISBN 9780521010122.

G.H. Lewes. *Problems of Life and Mind*. Problems of Life and Mind. Trübner & Company, 1879.

S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, March-April 1973.

Sean Luke. Two fast tree-creation algorithms for genetic programming. *Evolutionary Computation, IEEE Transactions on*, 4(3):274–283, 2000.

Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

Sean Luke and Liviu Panait. A survey and comparison of tree generation algorithms. 2001.

Hammad Majeed and Conor Ryan. Using context-aware crossover to improve the performance of gp. In *In GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 847–854. ACM Press, 2006.

Olivier Martin, Steve W. Otto, and Edward W. Felten. Large-step markov chains for the tsp incorporating local search heuristics. *Operations Research Letters*, 11:219–224, 1992.

Rafael Martí, Manuel Laguna, and Fred Glover. Principles of scatter search. *European Journal of Operational Research*, 169(2):359 – 372, 2006. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2004.08.004. Feature Cluster on Scatter Search Methods for Optimization.

K Matsui. New selection method to improve the population diversity in genetic algorithms. In *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, volume 1, pages 625–630. IEEE, 1999.

Robert I Mckay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396, 2010.

Nicholas Freitag Mcphee and Justin Darwin Miller. Accurate replication in genetic programming. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95*, pages 303–309. Morgan Kaufmann, 1995.

Nicholas Metropolis and S. Ulam. The Monte Carlo method. 44(247):335–341, September 1949. ISSN 0162-1459 (print), 1537-274X (electronic). doi: http://dx.doi.org/10.2307/2280232.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953. ISSN 0021-9606. doi: 10.1063/1.1699114.

Brad L Miller and David E Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9(3):193–212, 1995.

Brad L. Miller, Brad L. Miller, David E. Goldberg, and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

Julian F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In Wolfgang Banzhaf, Jason Daida, Agoston E.

Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann. ISBN 1-55860-611-4.

Heinz Mühlenbein. Genetic algorithms, 1997.

Heinz Mühlenbein and Dirk Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm – i. continuous parameter optimization. *EVOLUTIONARY COMPUTATION*, 1:25–49, 1993.

John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Advances in genetic programming. chapter Explicitly Defined Introns and Destructive Crossover in Genetic Programming, pages 111–134. MIT Press, Cambridge, MA, USA, 1996. ISBN 0-262-01158-1.

Peter Nordin, Wolfgang Banzhaf, and Frank D. Francone. Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 12, pages 275–299. MIT Press, Cambridge, MA, USA, June 1999. ISBN 0-262-19423-6.

U. M. O'Reilly and F. Oppacher. The troubling aspects of a building block hypothesis for genetic programming. Working Paper 94-02-001, Santa Fe Institute, 1399 Hyde Park Road Santa Fe, New Mexico 87501-8943 USA, 1992.

Una-May O'Reilly. Using a distance metric on genetic programs to understand genetic operators, 1997.

IbrahimH. Osman and Gilbert Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996. ISSN 0254-5330. doi: 10.1007/BF02125421.

Massimo Pigliucci. Is evolvability evolvable? *Nature Reviews Genetics*, 9(1):75–82, January 2008. ISSN 1471-0056. doi: 10.1038/nrg2278.

Massimo Pigliucci. Genotype–phenotype mapping and the end of the 'genes as blueprint'metaphor. *Philosophical Transactions Royal Society B*, 365:557–566, 2010.

David Pisinger. An o(nr) algorithm for the subset-sum problem - and other balanced knapsack algorithms, 1995.

Riccardo Poli. Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming. Technical Report CSRP-96-14, University of Birmingham, School of Computer Science, August 1996. Presented at 3rd International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA'97.

Riccardo Poli. Hyperschema theory for gp with one-point crossover, building blocks, and some new results in ga theory. In *Genetic Programming, Proceedings of EuroGP 2000*, pages 15–16. Springer-Verlag, 2000.

Riccardo Poli. General schema theory for genetic programming with subtree-swapping crossover, genetic programming. In *In Proceedings of EuroGP, LNCS, (Milan):Springer-Verlag*, 2001a.

Riccardo Poli. Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines*, 2(2): 123–163, June 2001b. ISSN 1389-2576. doi: doi:10.1023/A:1011552313821.

Riccardo Poli and W. B. Langdon. Genetic programming with one-point crossover and point mutation. In *Soft Computing in Engineering Design and Manufacturing*, pages 180–189. Springer-Verlag, 1997.

Riccardo Poli and W. B. Langdon. Analysis of schema variance and short term extinction likelihoods. In *University of Wisconsin*, pages 284–292. Morgan Kaufmann, 1998a.

Riccardo Poli and W.B. Langdon. On the search properties of different crossover operators in genetic programming. In *University of Wisconsin*, pages 293–301. Morgan Kaufmann, 1998b.

Riccardo Poli and William B. Langdon. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252, 1998c. doi: doi:10.1162/evco.1998.6.3.253.

Riccardo Poli and Nicholas F. Mcphee. Covariant parsimony pressure for genetic programming, 2008.

Riccardo Poli and Nicholas Freitag McPhee. General schema theory for genetic programming with subtree-swapping crossover: Part I. *Evolutionary Computation*, 11(1):53–66, March 2003a. doi: doi:10.1162/106365603321829005.

Riccardo Poli and Nicholas Freitag McPhee. General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation*, 11(2):169–206, June 2003b. doi: doi:10.1162/106365603766646825.

Riccardo Poli, William B. Langdon, and Stephen Dignum. Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In *in GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary*, pages 1588–1595. Press, 2007a.

Riccardo Poli, William B Langdon, and Stephen Dignum. On the limiting distribution of program sizes in tree-based genetic programming. In *Genetic Programming*, pages 193–204. Springer, 2007b.

Riccardo Poli, Mario Graff, and Nicholas Freitag McPhee. Free lunches for function and program induction. In *Proceedings of the Tenth ACM SIGEVO Workshop on Foundations of Genetic Algorithms*, FOGA '09, pages 183–194, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-414-0. doi: 10.1145/1527125.1527148. URL http://doi.acm.org/10.1145/1527125.1527148.

G. Polya. *How to Solve It*. Princeton University Press, November 1971. ISBN 0691023565.

Nicholas J. Radcliffe. Equivalence class analysis of genetic algorithms, 1991.

I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, TU Berlin, 1971.

I. Rechenberg. *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Number 15 in Problemata. Frommann-Holzboog, Stuttgart-Bad Cannstatt, 1973.

Christian M. Reidys and Peter F. Stadler. Combinatorial landscapes. *SIAM Rev.*, 44(1):3–54, January 2002. ISSN 0036-1445. doi: 10.1137/S0036144501395952.

Emiliano Rodríguez-Mega, Alma Piñeyro-Nelson, Crisanto Gutierrez, Berenice García-Ponce, María De La Paz Sánchez, Estephania Zluhan-Martínez, Elena R Álvarez-Buylla, and Adriana Garay-Arroyo. Role of transcriptional regulation in the evolution of plant phenotype: A dynamic systems approach. *Developmental Dynamics*, 2015.

Justinian Rosca. Entropy-driven adaptive representation. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 23–32. Morgan Kaufmann, 1995.

Justinian P. Rosca. Analysis of complexity drift in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 286–294, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

D. Rosenkrantz, R. Stearns, and P. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977. doi: 10.1137/0206041.

G. Rudolph. Convergence analysis of canonical genetic algorithms. *Neural Networks, IEEE Transactions on*, 5(1):96–101, Jan 1994a. ISSN 1045-9227. doi: 10.1109/72.265964.

G. Rudolph. Convergence analysis of canonical genetic algorithms. *Neural Networks, IEEE Transactions on*, 5(1):96–101, 1994b.

Suzanne L. Rutherford. From genotype to phenotype: buffering mechanisms and the storage of genetic information. *BioEssays*, 22(12):1095–1105, 2000. ISSN 1521-1878.

Conor Ryan, J. J. Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 83–96, Paris, 14-15 April 1998. Springer-Verlag. ISBN 3-540-64360-5. doi: doi:10.1007/BFb0055930.

C. Schumacher, M. D. Vose, and L. D. Whitley. The no free lunch and problem description length. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 565–570. Morgan Kaufmann, 2001.

H. P. Schwefel. Numerische Optimierung von Computer-Modellen. Dissertation, 1974.

Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981. ISBN 0471099880.

Hans-Paul Paul Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1995. ISBN 0471571482.

Robert A. Skipper. The Persistence of the R.A. Fisher-Sewall Wright Controversy. *Biology and Philosophy*, 17(3):341–367, 2002.

T. Soule and J.A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 781–786, May 1998. doi: 10.1109/ICEC.1998.700151.

Lee Spector and Kilian Stoffel. Ontogenetic programming, 1996.

Lee Spector, W. B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors. *Advances in Genetic Programming 3*. MIT Press, Cambridge, MA, USA, June 1999. ISBN 0-262-19423-6.

Peter F. Stadler and Bärbel M.R. Stadler. Genotype-phenotype maps. *Biological Theory*, 1 (3):268–279, 2006. ISSN 1555-5542. doi: 10.1162/biot.2006.1.3.268.

Peter F. Stadler and Christopher R. Stephens. Landscapes and Effective Fitness, 2003.

Chris Stephens and Henri Waelbroeck. Schemata evolution and building blocks. *Evolutionary Computation*, 1999.

Christopher R Stephens and H Waelbroeck. Effective degrees of freedom in genetic algorithms. *Physical Review E*, 57(3):3251, 1998.

Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4): 341–359, 1997. ISSN 0925-5001. doi: 10.1023/A:1008202821328.

Thomas Stützle. *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms, and New Applications*. PhD thesis, TU Darmstadt, Computer Science Department, 1999.

EIICHI TANAKA and KEIKO TANAKA. The tree-to-tree editing problem. *International Journal of Pattern Recognition and Artificial Intelligence*, 02(02):221–240, 1988. doi: 10.1142/S0218001488000157.

Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2.

A. M. Turing. Computing machinery and intelligence, 1950. One of the most influential papers in the history of the cognitive sciences: http://cogsci.umn.edu/millennium/final.html.

NguyenQuang Uy, Michael O'Neill, NguyenXuan Hoai, Bob Mckay, and Edgar Galván-López. Semantic similarity based crossover in gp: The case for real-valued function regression. In Pierre Collet, Nicolas Monmarché, Pierrick Legrand, Marc Schoenauer, and Evelyne Lutton, editors, *Artifical Evolution*, volume 5975 of *Lecture Notes in Computer Science*, pages 170–181. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-14155-3. doi: 10.1007/978-3-642-14156-0_15.

Gabriel Valiente. An efficient bottom-up distance between trees. In *Proc. 8th Int. Symposium on String Processing and Information Retrieval*, pages 212–219. IEEE Computer Science Press, 2001.

V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, January 1985. ISSN 0022-3239. doi: 10.1007/bf00940812.

Michael D Voset and Gunar E Liepinsl. Punctuated equilibria in genetic search. *Complex systems*, 5:31–44, 1991.

Stefan Voss, Ibrahim H. Osman, and Catherine Roucairol, editors. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Norwell, MA, USA, 1999. ISBN 0792383699.

Gunter P. Wagner and Lee Altenberg. Complex Adaptations and the Evolution of Evolvability. *Evolution*, 1996a.

Günter P. Wagner and Lee Altenberg. Perspectives: Complex adaptations and the evolution of evolvability, 1996b.

Stefan Wagner. *Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*. Doctor technicae, Johannes Kepler University, Linz, Austria, May 7 2009.

Stefan Wagner, Gabriel Kronberger, Andreas Beham, Michael Kommenda, Andreas Scheibenpflug, Erik Pitzer, Stefan Vonolfen, Monika Kofler, Stephan Winkler, Viktoria Dorfer, and Michael Affenzeller. *Advanced Methods and Applications in Computational Intelligence*, volume 6 of *Topics in Intelligent Engineering and Informatics*, chapter Architecture and Design of the HeuristicLab Optimization Environment, pages 197–261. Springer, 2014.

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

Peter A Whigham et al. Grammatically-based genetic programming. Citeseer, 1995.

Darrell Whitley and Jean Paul Watson. Complexity theory and the no free lunch theorem, 2005.

Kay Wiese and Scott D. Goodwin. Keep-best reproduction: A selection strategy for genetic algorithms. In *Proceedings of the 1998 ACM Symposium on Applied Computing*, SAC '98, pages 343–348, New York, NY, USA, 1998. ACM. ISBN 0-89791-969-6. doi: 10.1145/330560.330837.

David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2011. ISBN 0521195276, 9780521195270.

Stephan Winkler, Michael Affenzeller, Bogdan Burlacu, Gabriel Kronberger, Michael Kommenda, and Philipp Fleck. Similarity-based analysis of population dynamics in genetic

programming performing symbolic regression. In Rick Riolo, William P. Worzel, and Mark Kotanchek, editors, *Genetic Programming Theory and Practice XIV*, Genetic and Evolutionary Computation. Springer, 2016.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997. ISSN 1089-778X. doi: 10.1109/4235.585893.

David H. Wolpert and William G. Macready. No free lunch theorems for search, 1995.

D.H. Wolpert and W.G. Macready. Coevolutionary free lunches. *Evolutionary Computation, IEEE Transactions on*, 9(6):721–735, Dec 2005. ISSN 1089-778X. doi: 10.1109/TEVC.2005.856205.

Man Leung Wong and Kwong Sak Leung. Evolutionary program induction directed by logic grammars. *Evol. Comput.*, 5(2):143–180, June 1997. ISSN 1063-6560. doi: 10.1162/evco.1997.5.2.143.

John R. Woodward and James R. Neil. *No Free Lunch, Program Induction and Combinatorial Problems*, pages 475–484. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-36599-0. doi: 10.1007/3-540-36599-0_45. URL http://dx.doi.org/10.1007/3-540-36599-0_45.

S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. *Proceedings of the Sixth International Congress on Genetics*, 1(6):356–366, 1932.

S Wright. "surfaces" of selective value. *Proceedings of the National Academy of Sciences*, 58 (1):165–172, 1967.

Stephen J. Wright. Continuous optimization (nonlinear and linear programming). In *Foundations of Computer-Aided Process Design*, 1999.

Huayang Xie. An analysis of selection in genetic programming. 2009.

Wei Yan and Christopher D. Clack. Behavioural GP diversity for dynamic environments: an application in hedge fund investment. In *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 2, pages 1817–1824, Seattle, Washington, USA, 8-12 July 2006. ACM Press. ISBN 1-59593-186-4. doi: doi:10.1145/1143997.1144290.

Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133 – 139, 1992. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/0020-0190(92)90136-J.

List of Figures

3.1 Crossover operators 42

4.1 Flowchart of the GP evolutionary process 48
4.2 Example symbolic expression tree encoding a mathematical formula 50
4.3 Example crossover 58
4.4 Sewall Wright's fitness landscape as originally shown in (Wright, 1932). Contours on the map connect points with equal fitness. 71
4.5 Hypothetical parameter space composed of six phenotypes (A, B, C, D, E, F) determined by the developmental interactions of two parameters x_1 and x_2. Image reproduced from (Alberch, 1991). 74
4.6 Genotype-phenotype map. Source: (Stadler and Stephens, 2003) 75
4.7 Variation–selection emergent loop in GP. Source: (Banzhaf, 2014) 78

5.1 The HeuristicLab algorithm model. Source: (Wagner et al., 2014) 82
5.2 Example bottom-up tree mapping. From (Valiente, 2001) 86
5.3 Intermediate crossover child saved as a vertex in the genealogy graph, where the vertex rank represents the generation. The intermediate rank value (0.5) means that the intermediate child is not part of the population at generation 1. 88
5.4 Example genealogy graph 90
5.5 Preorder arithmetics for subtree inclusion 92
5.6 Subtree weights (above each node) and sample counts (bold font, in brackets) 97
5.7 Dynamic replacement ratio rules 101

6.1 SGP Average Population Quality 105
6.2 SGP Average Selection Ratio 106
6.3 SGP Proportional Selector Average Parent Distribution 107
6.4 SGP Tournament Selector Average Parent Distribution 107
6.5 SGP Average Tree Length 108
6.6 SGP Average Crossover Fragment Length 108
6.7 SGP Average Mutation Fragment Length 109
6.8 Size distribution of individuals in the population and subtrees replaced by the crossover operator 110
6.9 SGP Average Genotype Similarity 111
6.10 SGP Average Phenotype Similarity 112

6.11 SGP Proportional Selector Average Crossover Improvement 113
6.12 SGP Tournament Selector Average Crossover Improvement 113
6.13 SGP Proportional Selector Average Mutation Improvement 114
6.14 SGP Tournament Selector Average Mutation Improvement 114
6.15 SGP Proportional Selector Schema Frequencies and Quality 116
6.16 SGP Tournament Selector Schema Frequencies and Quality 116
6.17 SGP Proportional Selector Schema Similarities 117
6.18 SGP Tournament Selector Schema Similarities 117
6.19 SGP Proportional Selector Cumulated Subtree Sample Count and Parent Contribution Ratio 119
6.20 SGP Tournament Selector Cumulated Subtree Sample Count and Parent Contribution Ratio 120
6.21 OSGP Proportional Selector Average Population Quality 124
6.22 OSGP Gender Specific Selector Average Population Quality 125
6.23 OSGP Proportional Selector Average Selection Ratio 126
6.24 OSGP Gender Specific Selector Average Selection Ratio 126
6.25 OSGP Proportional Selector Average Parent Distribution 127
6.26 OSGP Gender Specific Selector Average Parent Distribution 127
6.27 OSGP Proportional Selector Average Tree Length 128
6.28 OSGP Gender Specific Selector Average Tree Length 129
6.29 OSGP Proportional Selector Average Crossover Fragment Length 129
6.30 OSGP Gender Specific Selector Average Crossover Fragment Length 130
6.31 OSGP Proportional Selector Average Mutation Fragment Length 130
6.32 OSGP Gender Specific Selector Average Mutation Fragment Length 131
6.33 OSGP Proportional Selector Average Genotype Similarity 132
6.34 OSGP Gender Specific Selector Average Genotype Similarity 132
6.35 OSGP Proportional Selector Average Phenotype Similarity 133
6.36 OSGP Gender Specific Selector Average Phenotype Similarity 133
6.37 OSGP Proportional Selector Average Crossover Improvement 134
6.38 OSGP Gender Specific Selector Average Crossover Improvement 134
6.39 OSGP Proportional Selector Average Mutation Improvement 135
6.40 OSGP Gender Specific Selector Average Mutation Improvement 135
6.41 OSGP Proportional Selector Schema Frequencies and Quality 136
6.42 OSGP Gender Specific Selector Schema Frequencies and Quality 137
6.43 OSGP Proportional Selector Schema Similarities 137
6.44 OSGP Gender Specific Selector Schema Similarities 138
6.45 OSGP Proportional Selector Cumulated Subtree Sample Count and Parent Contribution Ratio 139
6.46 OSGP Gender Specific Selector Cumulated Subtree Sample Count and Parent Contribution Ratio 140

List of Tables

2.1 Algorithm complexity classes, in increasing complexity from top to bottom 13
2.2 Metaphors and metaheuristics 21

4.1 Linear and tree-based grammar guided GP systems 50

5.1 Tracing cases 93

6.1 Problem formulas, training and test partitions and best achievable quality. . 103
6.2 SGP Best Solution Qualities 104
6.3 SGP Average Solution Qualities 105
6.4 SGP Best Solution Contribution Ratio 118
6.5 SGP Best Solution Trace v. Ancestry Size 118
6.6 Correlation between the cumulated subtree sample count and the cumulated parent contribution ratio curves from Figs. 6.19 and 6.20 119
6.7 SGP Proportional Selector Most Sampled Subtrees 121
6.8 SGP Tournament Selector Most Sampled Subtrees 121
6.9 OSGP Best Solution Qualities 123
6.10 OSGP Average Solution Qualities 124
6.11 OSGP Best Solution Contribution Ratio 138
6.12 OSGP Best Solution Trace v. Ancestry Size 139
6.13 OSGP Proportional Selector Most Sampled Subtrees 140
6.14 OSGP Gender Specific Selector Most Sampled Subtrees 141
6.15 OSGP-S Adaptive Replacement Ratio Experiment Configuration 143
6.16 OSGP-S Adaptive Replacement Ratio, Minimum Phenotypic Similarity 90%. The results are expressed as $(\mu \pm \sigma)$ values of the training and test qualities in the first and second line, respectively. 144
6.17 OSGP-S Adaptive Replacement Ratio, Minimum Phenotypic Similarity 95%. The results are expressed as $(\mu \pm \sigma)$ values of the training and test qualities in the first and second line, respectively. 145
6.18 OSGP-S Fixed Replacement Ratio 90%. The results are expressed as $(\mu \pm \sigma)$ values of the training and test qualities in the first and second line, respectively. 145
6.19 OSGP Average Solution Qualities - 5M Evaluations. The results are expressed as $(\mu \pm \sigma)$ values of the training and test qualities in the first and second line, respectively. 147

7.1 SGP Poly-10 Proportional Most Frequent Schemas 156
7.2 SGP Poly-10 Tournament Most Frequent Schemas 157
7.3 SGP Pagie-1 Proportional Most Frequent Schemas 158
7.4 SGP Pagie-1 Tournament Most Frequent Schemas 159
7.5 SGP Friedman-2 Proportional Most Frequent Schemas 160
7.6 SGP Friedman-2 Proportional Most Frequent Schemas 161
7.7 SGP Tower Proportional Most Frequent Schemas 162
7.8 SGP Tower Tournament Most Frequent Schemas 163
7.9 OSGP Poly-10 Proportional Most Frequent Schemas 164
7.10 OSGP Poly-10 Gender Specific Most Frequent Schemas – Part 1 165
7.11 OSGP Poly-10 Gender Specific Most Frequent Schemas – Part 2 166
7.12 OSGP Pagie-1 Proportional Most Frequent Schemas 167
7.13 OSGP Pagie-1 Gender Specific Most Frequent Schemas 168
7.14 OSGP Friedman-2 Proportional Most Frequent Schemas 169
7.15 OSGP Friedman-2 Gender Specific Most Frequent Schemas 170
7.16 OSGP Tower Proportional Most Frequent Schemas 171
7.17 OSGP Tower Gender Specific Most Frequent Schemas – Part 1 172
7.19 OSGP Average Evaluated Solutions 173
7.18 OSGP Tower Gender Specific Most Frequent Schemas – Part 2 173
7.20 OSGP-S Adaptive Replacement Average Solution Evaluations 174
7.21 OSGP-S Fixed Replacement Average Solution Evaluations 174

Bogdan Burlacu

Dipl.-Ing.

Julius Raab Straße 10
4040, Linz, Austria
+43(0)67762141817
+43(0)50804-27134
FAX *+43(0)50804-27199*
bogdan.burlacu@fh-hagenberg.at

Personal Information

Date of birth	November 3, 1984
Nationality	Romanian
Civil Status	Not Married

Education

Nov 2011 – Sep 2017	**PhD Studies in Computer Science**, *Johannes Kepler University*, Linz, Austria, **Thesis title**: "Exact Tracing of Evolutionary Search Trajectories in Complex Hypothesis Spaces".
2004–2009	**Bachelor of Science**, *"Gheorge Asachi" Technical University*, Iași, Romania, **Main field of study**: Systems and Computer Engineering, **Specialization**: Automatic Control and Applied Informatics **Thesis title**: "Genetic Programming Techniques for the Design of Hybrid Neural Networks".
2000–2004	**Baccaulaureate Diploma**, *"Costache Negruzzi" College*, Iași, Romania.

Experience

Vocational

2008–2011 **System administrator and developer**, *SC Software Development Partnership SRL*, Iași, Romania.

Responsibilities ranged from networking and security (firewalls, virtual private networks, encrypted backup) to server monitoring, performance tuning, database management, batch data processing, configuration of web and mail servers in various setups, managing user accounts and contributing to software projects. In charge of 10 Linux servers. Extensive experience using Linux applications and tools.

Academic

Nov 2011–Sep 2017 **PhD Studies**, *Johannes Kepler University*, Linz, Austria.

Nov 2011–Present **Teaching and Research Assistant**, *University of Applied Sciences Upper Austria*, Hagenberg, Austria, Heuristics and Evolutionary Algorithms Laboratory.

Currently teaching "Data Fusion in Multi-sensor Systems" for Master students at the University. Actively involved in research and collaboration projects with industry partners in the areas of symbolic regression and optimization.

Developer, *HeuristicLab*, Open source software for heuristic optimization.

2009–2011 **Remote PhD Studies**, *"Gheorghe Asachi" Technical University, Department of Automatic Control and Applied Informatics*, Iași, Romania, Research topic: "Graph-based Genetic Programming".

Extracurricular

1992–Present **Chess player**.

For me, chess is more than just a sport. It is directly related to intellectual development and to art. Became a club player at age 8. Actively pursued a chess career up until 2007, participated in many national and international chess tournaments, played for a team in the Romanian League. Currently holding the title of Fide Master and playing in Austrian and German chess leagues.

Publications

Winkler, Stephan et al. “Similarity-based Analysis of Population Dynamics in Genetic Programming Performing Symbolic Regression”. In: *Genetic Programming Theory and Practice XIV*. Ed. by Rick Riolo, William P. Worzel, and Mark Kotanchek. Genetic and Evolutionary Computation. Springer, 2016.

Burlacu, Bogdan, Michael Affenzeller, and Michael Kommenda. “On the Effectiveness of Genetic Operations in Symbolic Regression”. In: *Proceedings of the International Conference on Computer Aided Systems Theory (EUROCAST 2015)*. Las Palmas, Gran Canaria, Spain, Feb. 2015, p. 8.

Burlacu, Bogdan, Michael Affenzeller, Stephan M. Winkler, et al. “Methods for Genealogy and Building Block Analysis in Genetic Programming”. In: *Computational Intelligence and Efficiency in Engineering Systems*. Ed. by Grzegorz Borowik et al. Vol. 595. Studies in Computational Intelligence. Springer, 2015, pp. 61–74.

Burlacu, Bogdan, Michael Kommenda, and Michael Affenzeller. “Building Blocks Identification Based on Subtree Sample Counts for Genetic Programming”. In: *2015 Asia-Pacific Conference on Computer Aided System Engineering (APCASE)*. July 2015, pp. 152–157. DOI: doi:10.1109/APCASE.2015.34.

Kommenda, Michael, Michael Affenzeller, Gabriel Kronberger, et al. “Multi-Population Genetic Programming with Data Migration for Symbolic Regression”. In: *Computational Intelligence and Efficiency in Engineering Systems*. Ed. by Grzegorz Borowik et al. Vol. 595. Studies in Computational Intelligence. Springer, 2015, pp. 75–87.

Kommenda, Michael, Gabriel Kronberger, et al. “Evolving Simple Symbolic Regression Models by Multi-objective Genetic Programming”. In: *Genetic Programming Theory and Practice XIII*. Ed. by Rick Riolo et al. Genetic and Evolutionary Computation. Ann Arbor, USA: Springer, May 2015.

Kommenda, M. et al. “Heat Treatment Process Parameter Estimation using Heuristic Optimization Algorithms”. In: *Proceedings of the 27th European Modeling and Simulation Symposium EMSS 2015*. Bergeggi, Italy, Sept. 2015, pp. 222–228.

Kommenda, Michael, Michael Affenzeller, Bogdan Burlacu, et al. “Genetic programming with data migration for symbolic regression”. In: *GECCO 2014 Workshop on Symbolic Regression and Modelling*. Ed. by Steven Gustafson and Ekaterina Vladislavleva. Vancouver, BC, Canada: ACM, Dec. 2014, pp. 1361–1366. DOI: doi:10.1145/2598394.2609857.

Winkler, Stephan M. et al. “Sliding Window Symbolic Regression for Detecting Changes of System Dynamics”. In: *Genetic Programming Theory and Practice XII*. Ed. by Rick Riolo, William P. Worzel, and Mark Kotanchek. Genetic and Evolutionary Computation. Ann Arbor, USA: Springer, Aug. 2014, pp. 91–107. DOI: doi:10.1007/978-3-319-16030-6_6.

Affenzeller, Michael et al. “Gaining Deeper Insights in Symbolic Regression”. In: *Genetic Programming Theory and Practice XI*. Ed. by Rick Riolo, Jason H. Moore, and Mark Kotanchek. Genetic and Evolutionary Computation. Ann Arbor, USA: Springer, Sept. 2013. Chap. 10, pp. 175–190. DOI: doi:10.1007/978-1-4939-0375-7_10.

Burlacu, Bogdan, Michael Affenzeller, and Michael Kommenda. “On the Evolutionary Behavior of Genetic Programming with Constants Optimization”. In: *Computer Aided Systems Theory, EUROCAST 2013*. Ed. by Roberto Moreno-Diaz, Franz Pichler, and Alexis Quesada-Arencibia.

Vol. 8111. Lecture Notes in Computer Science. 14th International Conference, Revised Selected Papers. Las Palmas de Gran Canaria, Spain: Springer, Feb. 2013, pp. 284–291. DOI: doi:10.1007/978-3-642-53856-8_36.

Burlacu, Bogdan, Michael Affenzeller, Michael Kommenda, Stephan Winkler, et al. "Visualization of genetic lineages and inheritance information in genetic programming". In: *GECCO '13 Companion: Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*. Ed. by Christian Blum et al. Amsterdam, The Netherlands: ACM, June 2013, pp. 1351–1358. DOI: doi:10.1145/2464576.2482714.

Burlacu, Bogdan, Michael Affenzeller, Michael Kommenda, Stephan M. Winkler, et al. "Evolution Tracking in Genetic Programming". In: *The 24th European Modeling and Simulation Symposium, EMSS 2012*. Ed. by Emilio Jimenez and Boris Sokolov. Vienna, Austria, Sept. 2012.

Scheibenpflug, Andreas et al. "On the Analysis, Classification and Prediction of Metaheuristic Algorithm Behavior For Combinatorial Optimization Problems". In: *Proceedings of the 24th European Modeling and Simulation Symposium EMSS 2012*. Vienna, Austria, Sept. 2012.

Ferariu, Lavinia and Bogdan Burlacu. "Multiobjective design of evolutionary hybrid neural networks". In: *17th International Conference on Automation and Computing (ICAC 2011)*. Huddersfield, UK, Oct. 2011, pp. 195–200.

– "Multiobjective genetic programming with adaptive clustering". In: *IEEE International Conference on Intelligent Computer Communication and Processing (ICCP 2011)*. Cluj-Napoca, Romania, 25-27 8 2011, pp. 27–32. DOI: doi:10.1109/ICCP.2011.6047840.
– "Multiobjective Graph Genetic Programming with Encapsulation Applied to Neural System Identification". In: *15th International Conference on System Theory, Control, and Computing (ICSTCC 2011)*. Sinaia, 14-16 10 2011.
– "Graph genetic programming for hybrid neural networks design". In: *International Joint Conference on Computational Cybernetics and Technical Informatics (ICCC-CONTI)*. May 2010, pp. 547–552. DOI: doi:10.1109/ICCCYB.2010.5491213.