Analyzing the Discovery and Use of Modules in Grammatical Evolution

John Mark Swafford

Thesis presented for the degree of

Ph.D. in Computer Science

to the

School of Computer Science and Informatics

College of Science

University College Dublin

Research Supervisors:

Dr. Michael O'Neill

Dr. Elizabeth Shotton

April 30, 2013

Abstract

Modularity, and the ability to exploit it, has been noted as an important open issue in the field of Genetic Programming (GP). This thesis examines modularity in the context of one area of GP, a grammar-based form of GP called Grammatical Evolution (GE). Very little work has been done in GE to examine how modularity can be incorporated and used to improve GE's search performance, leaving many open research questions. All of these open research questions can be considered as answering a portion of one the following two questions: "How should modules be made available to individuals during evolution?" and "How should modules be selected from the population?"

The first set of examinations provided by this thesis explores how modules should be incorporated into GE's grammar once they are discovered. Settings for the number of generations between grammar modifications, how many modules are added to the grammar, and the manner in which the grammar is modified are tested. The results of these tests show that certain parameter settings perform better than others, but no approach was significantly better than standard GE on three out of four of the benchmark problems used due to the disruption in the population caused by modifying GE's grammar. This lead to the creation of the genotype repair operator which ensures that grammar modifications do not change individuals' phenotypes. The inclusion of the genotype repair operator leads to an improvement in performance when adding modules to GE's grammar, but does not give a significant increase in performance on most of the benchmark problems tested.

Next, different methods for identifying modules are implemented, and their performance is compared against each other and standard GE. These comparisons considered numerous parameters for the module identification methods such as which individuals modules are taken from, how many fitness evaluations are used to estimate a module's contribution to an individual, and how much selection pressure should be used when selecting candidate modules. The results of these comparisons show that there is no single-best combination of

these parameters; however, approaches which used less fitness evaluations during the module identification step tended to outperform those which did not. Finally, an examination into the characteristics and usage of the modules discovered by the various module identification approaches is presented. This study showed that the best methods for identifying modules find modules within certain size and fitness diversity ranges. It also demonstrates the best approaches for identifying modules are able to more easily replace older modules with newer modules.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Publications Arising

1. M. O'Neill, J. M. Swafford, J. McDermott, J. Byrne, A. Brabazon, E. Shotton, C. McNally, and M. Hemberg. Shape grammars and grammatical evolution for evolutionary design. In G. Raidl, F. Rothlauf, and et al., editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1035–1042, Montreal, 8-12 July 2009. ACM

2. J. M. Swafford and M. O'Neill. An examination on the modularity of grammars in grammatical evolutionary design. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*. IEEE, Jul 2010

3. M. O'Neill, J. McDermott, J. M. Swafford, J. Byrne, E. Hemberg, E. Shotton, C. McNally, A. Brabazon, and M. Hemberg. Evolutionary design using grammatical evolution and shape grammars: Designing a shelter. *International Journal of Design Engineering*, 3, 2010

4. E. G. López, D. Fagan, E. Murphy, J. M. Swafford, A. Agapitos, M. O'Neill, and A. Brabazon. Comparing the performance of the evolvable pigrammatical evolution genotype-phenotype map to grammatical evolution in the dynamic Ms. Pac-Man environment. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*, pages 1–8. IEEE, 2010

5. J. McDermott, J. Byrne, J. M. Swafford, M. O'Neill, and A. Brabazon. Higher-order functions in aesthetic EC encodings. In *CEC 2010. Proceedings of the 12th IEEE Conference on Evolutionary Computation*, Jul 2010

6. E. G. López, J. M. Swafford, M. O'Neill, and A. Brabazon. Evolving a Ms. Pac-Man controller using grammatical evolution. In C. D. Chio, S. Cagnoni, C. Cotta,

M. Ebner, A. Ekárt, A. Esparcia-Alcázar, C. K. Goh, J. J. M. Guervós, F. Neri, M. Preuss, J. Togelius, and G. N. Yannakakis, editors, *EvoApplications (1)*, volume 6024 of *Lecture Notes in Computer Science*, pages 161–170. Springer, 2010

7. J. M. Swafford, M. O'Neill, M. Nicolau, and A. Brabazon. Exploring grammatical modification with modules in grammatical evolution. In S. Silva, J. A. Foster, M. Nicolau, P. Machado, and M. Giacobini, editors, *EuroGP*, volume 6621 of *Lecture Notes in Computer Science*, pages 310–321. Springer, 2011

8. J. M. Swafford, E. Hemberg, M. O'Neill, M. Nicolau, and A. Brabazon. A non-destructive grammar modification approach to modularity in grammatical evolution. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1411–1418, Dublin, Ireland, 2011. ACM

9. J. M. Swafford, M. Nicolau, E. Hemberg, M. O'Neill, and A. Brabazon. Comparing methods for module identification in grammatical evolution. In T. Soule and J. H. Moore, editors, *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012*, pages 823–830. ACM, 2012

10. J. M. Swafford, E. Hemberg, M. O'Neill, and A. Brabazon. Analyzing module usage in grammatical evolution. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Proceedings of the 12th Int. Conf. on Parallel Problem Solving From Nature*, volume 7491 of *Lecture Notes in Computer Science*. Springer, 2012

11. J. McDermott, J. Byrne, J. M. Swafford, M. Hemberg, C. McNally, E. Shotton, E. Hemberg, M. Fenton, and M. O'Neill. String-rewriting grammars for evolutionary architectural design. *Environment and Planning B: Planning and Design*, 39(4):713–731, 2012

Part I

Introduction and Background

Chapter 1

Introduction

This thesis examines the effects of incorporating and exploiting modularity in Grammatical Evolution (GE) [33, 114]. GE is one of many evolutionary algorithms (EAs), which can be broadly defined as population-based optimization algorithms. They use evolutionary operators inspired by those seen in biological evolution to generate, recombine, select, and mutate individuals to better solve a given problem. One popular EA is Genetic Programming (GP) [24, 73], which represents individuals as syntax trees. The aspect of GP that interests many researchers is its ability to evolve executable computer programs. GE is a logical extension to GP, as it is a grammar-based form of GP. In GE, individuals are represented as integer or binary arrays which go through a mapping process using a user-defined context-free grammar. The details of the GE algorithm are given in Chapter 2.

For many years, the idea of modularity has been studied in numerous different contexts. The concept of modularity in EAs originates from many areas of the biological world [11, 56, 84, 144, 163]. In the area of evolutionary algorithms (EAs) modularity has been shown to improve performance on appropriate problems [43, 51, 59, 74, 124]. Researchers in the field of GP have been particularly inspired by the benefits exhibited by modular biological systems (More rigorous coverage of research in this area is provided in Chapter 3). While

modularity has received much attention in the context of GP, it is still considered an open issue in the field [117]. GE, has not been the focus of nearly as many studies. Modularity in the context of GE refers to identifying beneficial sub-solutions from individuals. Ideally, these sub-solutions are able to be reused in the population to create better individuals. This thesis fills some of the gaps in terms of how modules should be discovered and used, as well as the effects of the incorporation of these modules in GE's population.

There are three main characteristics of modularity that make it appealing in the eyes of GP researchers. The first being that incorporating modularity into an EA facilitates *decomposing a large problem* in smaller, more easily solvable, sub-problems. The second characteristic is *hierarchy*. The advantage of hierarchy comes from decomposing a problem and solving the sub-problems on their own. EAs are able to use these partial solutions in conjunction with each other or with other primitive elements in an accretive manner to solve the whole problem. The third characteristic is the *reuse* of modules. Being able to identify useful components of a system and preserve them for further use can save an EA much work in terms of building a complete solution to a problem. However, the introduction of these features also adds more questions left to researchers to answer: how should a given problem be decomposed, which modules and primitives should be coupled together to create more complex modules, and how should modules be used during evolution?

In addition to the questions raised by problem decomposition, hierarchy, and use of modules in a modular system, there are many additional parameters that must be considered when extending an EA to incorporate some form of modularity. These parameters can be considered in one broad question: How should modularity be incorporated into an EA? In order to answer this question, most researchers develop methods to explicitly identify modules and make them available to the evolving solutions to a problem. A large body of work touches these areas (See Chapter 3), but, as one would expect, there is no single, best performing approach to modularity in GP or GE. The following chapters of this thesis

examine how modules should be made available to individuals during evolution (Part II) and how modules should be identified (Part III).

1.1 Problem Definition

This thesis presents an analysis of various methods for identifying modules and making them available for use during evolutionary runs of GE. There are many design choices a user must make when developing such methods, and each of these decisions comes with certain consequences. The most obvious of these choices is how a module should be defined. In the experiments carried out for this thesis, a module is defined as "an encapsulated sub-derivation tree containing more than one primitive or preexisting module, or a combination of primitives and modules." Another decision that must be made is which individuals in the population should be allowed to contribute modules. Limiting the individuals searched may restrict the quantity and quality of modules that can be found. On the other hand, identifying modules from the entire population may be computationally expensive and reveal many neutral or detrimental modules. Such decisions have the potential to drastically alter GE's performance and must be carefully considered. This section outlines the most important of these design choices and how they related to GE.

1.1.1 Identifying Beneficial Modules

One of the most crucial aspects of modular evolutionary systems are how the modules are identified. Garibay and Wu [45] show that encapsulating modules and incorporating them into the evolving population increases the total search space. In the same work [45], they also point out that, using a genetic algorithm (GA) [61], encapsulating a bad module increases the average hamming distance to a solution while encapsulating a good module decreases the hamming distance to a solution. Even though the research of Garibay and

Wu [45] focuses on GAs, this lesson can be applied to other EAs, such as GE. How modules are identified is arguably the most influential factor in determining if good or bad modules will be introduced into the population. Questions that a user must ask themselves while designing methods for identifying modules include, but are not limited to:

- From which individuals should modules be identified?
- Should candidate modules be evaluated based on a fitness-based measure, a usage-based measure, or a composite of the two?
- How should the fitness and/or usage be measured?
- What is an appropriate weighting for the different components in a composite measure?

Despite the large quantity of work in modularity and GP (See Chapter 3), there is no definitive answer to these questions.

1.1.2 Using Modules During Evolution

After a selection of modules has been identified, they must be made accessible by the population of an EA. Similar to identifying modules, there are many possibilities for incorporating modules into individuals during evolution with their own set of effects. Some of the questions developers must answer when determining how modules should be incorporated into an evolving population include:

- How many modules should be made available to the population at once?
- Should new modules be discovered and added at a constant time step or when a feature of the population (fitness diversity, phenotypic diversity, change in fitness over time, etc.) reaches a specified value?

- In what manner should new modules be made available to individuals?

Like the problem of how to identify modules (Section 1.1.1), there is no definitive answer to these questions.

1.2 Aim of Thesis

This thesis aims to tackle some of the most important questions regarding how modules should be identified and used, and analyze how different parameters for these functionalities enhance or detract from the performance of GE.

1.2.1 Research Questions

In order to accomplish the aim of this thesis, the following questions are asked:

Module Identification

Research Question 1: *Should modules be identified using a fitness-based or a usage-based approach?*

Research Question 2: *How much computational power is reasonable to use in identifying modules?*

Research Question 3: *What are the characteristics of the modules themselves that makes them useful?*

Research Question 4: *Are modules discovered by various identification methods used differently by evolution?*

Module Usage

Research Question 5: *How should modules be made available to GE's population during evolution?*

Research Question 6: *Does updating the set of modules available to individuals more or less frequently change the performance of GE?*

Research Question 7: *Does the number of modules made available to GE during evolution affect GE's performance?*

1.2.2 Objectives of Thesis

To answer the research questions posed in Section 1.2.1, multiple objectives are presented:

1. a survey of past research covering modularity and GP and GE;
2. implementing operations for identifying modules and incorporating them into an evolutionary run using GEVA [109]
3. understand how the frequency and manner in which modules are added to GE's grammar alters performance
4. understand the benefits and drawbacks of various methods for identifying modules
5. analyzing the characteristics of the modules discovered by different methods of identification
6. examining if modules are used (in)frequently or by (un)fit individuals during evolution.

1.3 Contributions

Many of the contributions provided by this thesis have been published. These publications are enumerated on Page viii. The main contributions of this work are listed by order of appearance in this thesis here:

Literature Review: An extensive review of approaches to modularity in GP is conducted in Chapter 3. This review covers definitions and applications of modules and modularity, (Section 3.2) and categorizes the various methods based on the approach incorporating modularity (Section 3.3). These methods are also categorized in tables outlining how they identify (Table 3.1) and make modules available to individuals (Table 3.2) during evolution.

Development of new module identification methods: Previous work in modularity in GP has presented numerous methods for module identification (Section 3.3). This thesis presents two novel methods for identifying modules in GE, which are also applicable to GP (One is first introduced in Section 4.1, but more detailed information about these algorithms is give in Chapter 6). These methods describe two ways to estimate the fitness contribution of a module, are parameterizable, and require no additional domain knowledge of the user, outside the original fitness function specification.

Comparison of module identification approaches: This thesis also offers a fair comparison of how various module identification operators compare with each other in terms of performance in GE (Chapter 6). Many parameters are examined and suggestions for which ones are reasonable and why is also discussed.

Analysis of module characteristics: While there has been much previous work showing how incorporating modularity into GP or GE alters the search performance of these algorithms, comparatively little has been done to show where these changes, or lack

thereof, come from. Another contribution of this thesis is the analysis of the characteristics of the modules which are discovered (Sections 7.2 - 7.3.2). This analysis compares the characteristics of modules discovered by the various module identification approaches in order to see which features of modules are correlated with the better performing approaches.

Analysis of module usage during evolution: Similar to the analysis of module characteristics, very little research examines how often modules are used and the fitness of individuals that use these modules. The final contribution of this thesis is the analysis of how many individuals in the population use a given module and the average fitness of individuals using a module (Section 7.4).

Analysis of modifying GE's grammar: In Chapter 4, variations on adding modules to GE's grammar are examined. The manner in which GE's grammar is modified (Section 4.4.1) and how frequently it is modified (Section 4.4.2) may be detrimental to GE's performance. The work presented in Chapter 4 gives suggestions for how to modify GE's grammar in a manner which causes the least amount of disruption to GE's performance.

Genotype repair operator: The process of modifying GE's grammar was shown to have a negative impact on GE's performance in Chapter 4. A genotype repair operator has been developed to remedy this effect (Chapter 5). As an added bonus, the genotype repair operator is not exclusively tied to adding modules to GE's grammar. It may also be used in conjunction with any operator with changes GE's grammar in order to prevent individuals' genotype-to-phenotype mapping from changing.

Publication of the work presented in this thesis: Each of the experimental chapters in this thesis is based on a corresponding publication. These works, in order of publication, are listed here:

- J. M. Swafford, M. O'Neill, M. Nicolau, and A. Brabazon. Exploring grammatical

modification with modules in grammatical evolution. In S. Silva, J. A. Foster, M. Nicolau, P. Machado, and M. Giacobini, editors, *EuroGP*, volume 6621 of *Lecture Notes in Computer Science*, pages 310–321. Springer, 2011 (Chapter 4),

- J. M. Swafford, E. Hemberg, M. O'Neill, M. Nicolau, and A. Brabazon. A non-destructive grammar modification approach to modularity in grammatical evolution. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1411–1418, Dublin, Ireland, 2011. ACM (Chapter 5),

- J. M. Swafford, M. Nicolau, E. Hemberg, M. O'Neill, and A. Brabazon. Comparing methods for module identification in grammatical evolution. In T. Soule and J. H. Moore, editors, *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012*, pages 823–830. ACM, 2012 (Chapter 6),

- J. M. Swafford, E. Hemberg, M. O'Neill, and A. Brabazon. Analyzing module usage in grammatical evolution. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Proceedings of the 12th Int. Conf. on Parallel Problem Solving From Nature*, volume 7491 of *Lecture Notes in Computer Science*. Springer, 2012 (Chapter 7).

1.4 Assumptions

Due to broad nature of problems and areas of study to which modularity can be applied or is inherently present, a number of assumptions must be considered. Firstly, there are numerous definitions and descriptions of modularity and what constitutes a module. In this thesis, a module is defined as **an encapsulated sub-derivation tree from a GE**

individual. There are many variations on how modules can be represented and what limits can be placed on module size and content. These variations will be discussed further in Chapter 3, but only standard GE, GE with Automatically Defined Functions (ADFs)[59, 113] and the approaches for identifying modules defined in Chapters 4 and 6 are examined. The addition of ADFs is a standard approach to modularity in GE and GP and is a sensible benchmark for gauging the performance of new approaches for identifying modules in GE and GP. The experimental results presented in Chapters 4 - 7 lay a foundation for possible future studies on modularity in GE by covering one possible module representation and numerous parameters which must be considered for any module identification operation.

With any EA, there are many parameters which can be tuned to improve its search performance. For the work presented in Chapters 4-7 only a single set of parameters (crossover rate, mutation rate, elite size, etc.) was used for the standard GE algorithm. No attempt was made to optimize these parameters. There are also many parameters which are introduced by the incorporation of modules into GE. An attempt was made to explore as many of these parameters as possible, however, there are still many permutations of these parameters which were left unexplored due to time restrictions. The settings tested in the experimental chapters of this thesis are considered sensible. For each parameter a small sweep of values was performed. Many of the values examined in these sweeps showed minimal, or no, change in performance when compared to other values. In order to present the reader with a more reasonable amount of data only parameters that exhibited differences from each other are discussed. Additional improvements in performance could be gained by optimizing these parameters.

There is a vast amount of literature covering modularity in a number of disciplines, e.g., biology and product design. Even though the idea of modularity in those areas is somewhat relevant to modularity in GP and GE, a great deal of this literature is omitted or mentioned only briefly. This was done to prevent the unnecessary bloat of the literature

review presented in Chapter 3.

1.5 Thesis Structure

The goal of this thesis is to examine methods for identifying modules and how those modules can be incorporated into an evolving population in GE. Through the studies presented in this work, insight into appropriate methods for incorporating modularity into GE is gained. The research carried out for this thesis is divided into four parts.

The first part of this thesis is Part I, which consists of three chapters. Chapter 1 briefly introduces modularity and the problems associated with integrating modularity into an EA. It also outlines the aims, research questions, objectives, contributions, and assumptions of this thesis. Next, Chapter 2 discusses the GE algorithm in terms of how individuals are represented, how a typical set of GE operators function, and some of the application areas and implementations of GE. The final chapter in Part I is Chapter 3. It presents some of the definitions of modularity and modules, a selection of application areas of modularity, and a literature review of the many approaches to modularity in GP research.

Part II of this thesis is split into two chapters. Chapter 4 introduces new methods for identifying modules. It then compares various methods adding modules to GE's grammar during an evolutionary run. It also compares variations for how frequently modules are added to GE's grammar and how many modules are added to GE's grammar per grammar modification step. The second chapter of Part II is Chapter 5. This chapter introduces the genotype repair operator which is used to ensure the addition of modules into GE's grammar does not present any adverse effects in terms of the fitness of the population.

Next, Part III is comprised of two chapters covering the definition and analysis of additional methods for identifying modules in GE and an analysis of the characteristics and features of the modules themselves. First, Chapter 6 defines four methods for identifying

modules in GE. It then covers various parameters and variations of these operators in an attempt to improve their performance and understand what features are most desirable in a module identification method. Chapter 7 goes on to analyze the features of the modules themselves. It reports how different module identification methods find modules of various size, content, and meaning. It also explains how modules discovered by different module identification methods are used more or less frequently and in better or worse individuals.

Finally, the last part of this thesis, Part IV, contains a single chapter. Chapter 8 presents a summary of the work carried out and the conclusions taken from the experiments and analysis performed in Parts II and III. It also describes a number of questions raised from this research and opportunities for future work in module identification and usage.

Chapter 2

The Grammatical Evolution Algorithm

Before delving into the experimental work of this thesis, the inner workings of the GE algorithm must be understood. This chapter details how GE creates solutions to problems and how these solutions are evolved over time. It also illustrates some of the differences between GP and GE.

2.1 Grammatical Evolution Overview

GE [33, 114] is a form of grammar-guided genetic programming [92]. Its functionality is similar to standard genetic programming, and in some cases is even identical. Like GP (and other EAs), it creates a population of individuals which is evolved in an attempt to solve a given problem by information exchanging, variation, selection, evaluation, and replacement operations. An example of this evolutionary cycle can be seen in Figure 2.1.

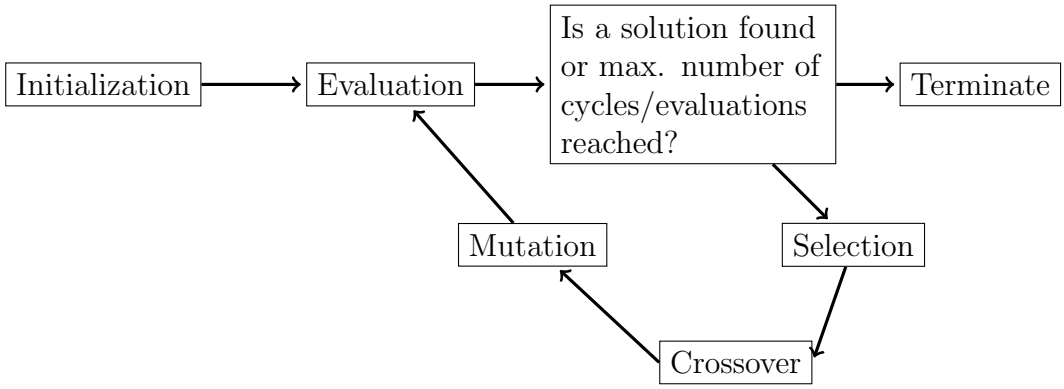


Figure 2.1: The evolutionary cycle

2.1.1 Differences Between GE and GP

What sets GE apart from standard GP is the linear representation of its individuals. While standard GP represents individuals as syntax trees created by user-defined function and terminal sets, GE represents individuals as linear arrays of binary numbers or integers[1], also known as chromosomes. A typical GE implementation uses a context-free grammar (CFG), usually in Backus-Naur Form (BNF), to map individuals' chromosomes to a phenotype which may be evaluated. To illustrate the differences between individuals in GE and GP Figure 2.2 shows examples of how individuals of individuals from both. However, it is also possible to represent individuals in GE as tree structures as well. This is explained in Section 2.2.

2.2 GE's Genotype-to-Phenotype Mapping

One of the large differences between GE and a standard implementation of GP is the mapping process that GE uses to create each individual's phenotype. Recalling the example individuals from Figure 2.2, note that the GP individual encodes its phenotypic information directly into its syntax tree, but the GE individual only contains integers. The mapping

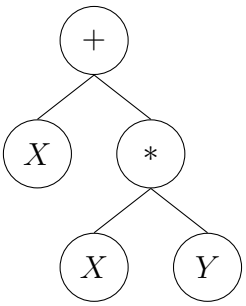
[1]The implementation used for all the experimental work in this thesis uses integer arrays.

$F = +, -, *, /$
$T = X, Y$

(a) GP Function and Terminal Set

<exp> ::= <op><exp><exp> | <var>
<var> ::= X | Y
<op> ::= $+$ | $-$ | $*$ | $/$

(b) GE Grammar in BNF



(c) GP Individual

2	4	9	24	42	78	31	52	63	11

(d) GE Individual

Figure 2.2: The difference between GP and GE individuals. Figures 2.2(a) and 2.2(c) show the function set and a possible tree for a GP individual. Figures 2.2(b) and 2.2(d) show a possible grammar and genotype for a GE individual. Both individuals create the expression $X + X * Y$ (or $+X * XY$ which is the prefix notation of the former).

process creates an individual's phenotype by combining the grammar and individual's integer array. An example of the mapping process will now be given using the grammar and genotype in Figures 2.2(b) and 2.2(d). First, the grammar's start symbol must be identified, which must always be a non-terminal and in this case is <exp>, and is used as the current phenotype of the individual. Next, the first element (elements in GE's chromosome are often referred to as codons) in the chromosome, the number 2, is taken. To determine how the phenotype is expanded, the following equation is used:

$$\text{codon value} \ \ \textit{mod} \ \ \text{number of productions} = \text{production index}.$$

The rule starting with <exp> is

<exp> ::= <op><exp><exp> | <var>,

which has 2 productions, and $2\%2 = 0$. This means the production in the 0^{th} index will be used to replace the current non-terminal. So, <exp> will be replaced with <op><exp><exp>.

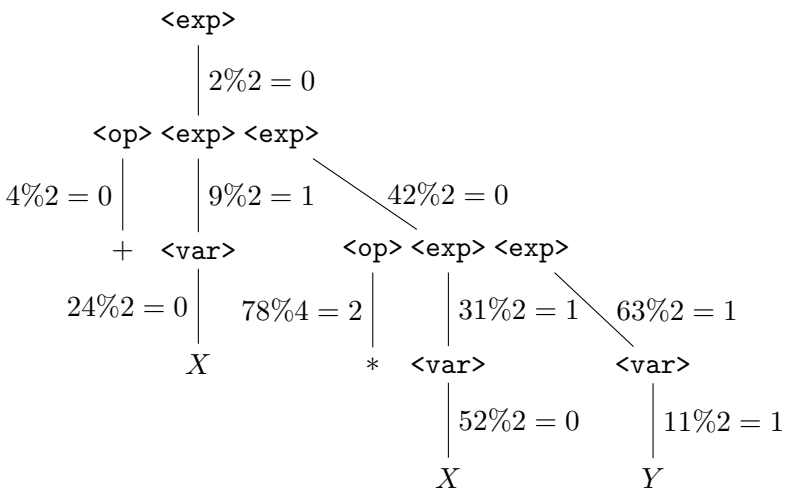


Figure 2.3: An example derivation tree using the grammar and chromosome from Figures 2.2(b) and 2.2(d). The labels on each edge show how the following node was chosen.

GE's mapping process always uses the left-most non-terminal, making `<op>` the current non-terminal to expand. The `<op>` non-terminal has 4 possible productions. So the next integer in the chromosome is used in the following rule: $4\%2 = 0$. Once again, the 0^{th} production is used to replace `<op>` making the individual's phenotype +`<exp><exp>`. The mapping process will continue in this manner until there are no more codons left in the chromosome or until there are no more non-terminal symbols left in the phenotype. Using the example chromosome and grammar, the individual maps out to the following phenotype: $+\ X\ *\ X\ Y$ which is simply the prefix notation of $X\ +\ X\ *\ Y$. Using this mapping process, it is also possible to create a tree structure (GE individuals represented as trees are often called derivation trees). The derivation tree of the individual derived above can be seen in Figure 2.3. In the example here, all the codons in the chromosome are used, but a situation may arise where not all the codons are used or that more codons are required than exist in the chromosome.

Wrapping and Invalid Individuals

During the mapping process, it is possible that every codon in the chromosome will not be used. Such codons are simply ignored. Crossover and/or mutation events may later create the need for the additional codons, and any extra codons at the end of an individual can help satisfy this need.

If more codons are required than exist, there are two options. The first is marking the individual as invalid and assigning it the worst possible fitness value. The second option is to allow *wrapping* of the chromosome. When wrapping is enabled, the user first specifies a maximum number of wrapping events. Then, when the mapping process hits the end of a chromosome but requires more codons, it wraps back to the beginning of the chromosome and continues with the first codon. If the maximum number of wraps is reached, the individual is marked as invalid and given the worst possible fitness.

When individuals are flagged as invalid, a number of possibilities arise for how to handle them. One is simply doing nothing and waiting for evolution to replace the individual due to its poor fitness. Another is replacing the invalid individual with one of its parents. A third possibility is replacing the invalid individual with a newly created individual. A final approach to handling invalid individuals would be to implement some form of a repair algorithm to modify invalid individuals' genotypes such that they are no longer invalid. In the experiments carried out in this thesis, replace invalid individuals with new individuals created using the same initialization method as the initial population.

2.3 GE Control Flow

The control flow of GE is similar to that of standard GP. First, a population of individuals is initialized and evaluated. Next, the evolutionary loop begins. Individuals are selected to swap information with each other and some individuals may also be mutated. The

modified individuals are then evaluated and given a fitness value. If some termination criteria has not been reached (usually an optimal solution, a given number of evolutionary cycles, or a given number of fitness evaluations) the population is updated with some of the newly created and evaluated individuals. Then, the evolutionary cycle moves to the selection step. An example of this can be found in Figure 2.1. The details of the phases of GE's evolutionary cycle are given below.

1. **Initialization** is the means by which new individuals are created. This may be done by randomly generating chromosomes or by generating chromosomes which map to individuals with derivation trees of certain depths. Initialization algorithms may be borrowed from GA and/or GP approaches.

2. **Selection** is the method of picking which individuals will be allowed to exchange information with one another. Most common are the roulette wheel and tournament selection methods. With the roulette wheel, individuals are selected proportionally based on their fitness. Tournament selection first needs a tournament size. It then randomly picks individuals equal to the tournament size. Out of the individuals picked, the one with the best fitness is allowed to participate in a crossover event.

3. **Crossover** allows individuals to swap information with each other. Because individuals in GE are integer arrays, but can also be represented as derivation trees, GA and GP methods of crossover may be used. An example of GE's standard single point crossover can be found in Figure 2.4.

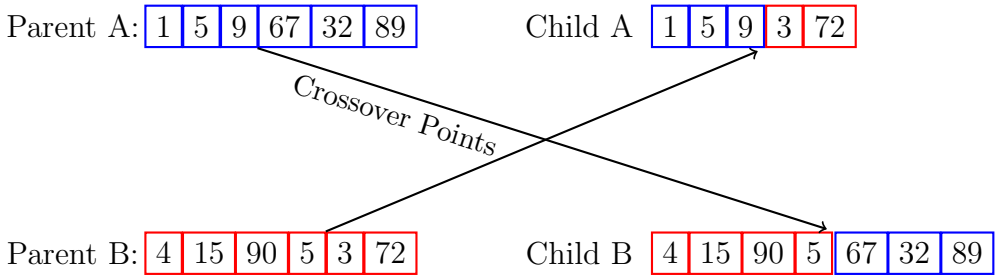


Figure 2.4: Single point crossover in GE

4. **Mutation** allows for random modifications to individuals. Like crossover, these operations can be borrowed from both GAs and GP. Most commonly used are the integer-flip (from GAs) and subtree (from GP) mutations. An example of GE's standard integer flip mutation can be found in Figure 2.5.

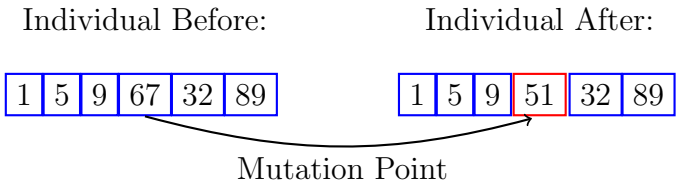


Figure 2.5: Integer-flip mutation in GE

5. **Replacement** is the mechanism for incorporating newly created and/or modified individuals back into the population. GE uses either the generational or steady state approach. Under the generational approach, the entire population is replaced with individuals created via crossover and mutation. Using the steady state approach, only individuals which improve on the population's fitness are allowed into the population.

2.4 Examinations and Extensions

Since its development, there have been many examinations into how GE is able to achieve its level of performance. There have also been numerous extensions which have been

implemented to enhance the performance of GE. Some of the first research in this area explored features of individuals such as genotype lengths, the amount of invalid individuals, and how many individuals had "wrapping" genotypes [112]. The functionality and performance of GE's standard one-point crossover as well additional crossover operators has been examined [54, 115, 120, 122]. Further investigations into GE involve the grammar used for the genotype-to-phenotype mapping process. Nicolau [103] examines how the complexity of the grammar used by GE can be reduced and may lead to improvements in performance. Similarly, Harper [53] examines different combinations of grammars and initialization schemes, showing that using the "wrong" combination of grammar and initialization method lead to a decrease in GE's performance. More research related to the genotype-to-phenotype mapping process is that of Fagan et al. [40, 41]. In their work, they investigate variations of the genotype-to-phenotype mapping process. These investigations show that the πGE genotype-to-phenotype mapping process presents advantages over the standard GE genotype-to-phenotype map. Another vein of research carried out by Fagan et al. [38, 39] shows that allowing evolution to alter the mutation rate of individuals may be beneficial when compared to GE's standard static mutation rate. More recent work looking at the behavior of GE is that of O'Neill et al. [111], who examine the behavior of GE in dynamic environments. They show that using modular varying goals in GE may lead to a "speed up" in evolution. In addition to the examinations of GE's performance, a number of extensions to the GE algorithm have been implemented. One of the first of these extensions is meta-grammar GE (mgGE) [121, 58]. This form of GE uses two grammars. First, a universal grammar specifies the productions for a second, solution grammar. The solution grammar is then used by the population to create programs which are evaluated by the fitness function. An additional extension of GE is tree-adjunct GE (TAGE) [97, 98, 99, 100]. This form of GE uses a tree-adjunct grammar instead of the standard CFG to create individuals and generally exhibits an increase in performance over

Table 2.1: Implementations of GE

Software Title	Language	URL
ECJ - Evolutionary Computation in Java	Java	http://cs.gmu.edu/~eclab/projects/ecj/
GEVA - GE in jaVA	Java	http://ncra.ucd.ie/Site/GEVA.html
JCLEC - Java Class Library for Evolutionary Computation	Java	http://jclec.sourceforge.net/
jGE - Java GE	Java	http://pages.bangor.ac.uk/~eep201/jge/
libGE	C++	http://bds.ul.ie/libGE/
ponyGE	Python	http://code.google.com/p/ponyge/
PyNeurGen - Python Neural Genetics Hybrid	Python	http://pyneurgen.sourceforge.net/
DRP - Directed Ruby Programming	Ruby	http://drp.rubyforge.org/
GERET - GE Ruby Exploratory Toolkit	Ruby	http://geret.org

standard GE.

2.5 Applications of GE

This section briefly describes some attempts at using GE to solve problems from a number of disciplines. The variety of problems on which GE has been used speaks for its versatility in problem solving. One popular trend is applying GE to problems in the finance domain [13]. Biological problems have also been tackled by GE. It has been used to capture protein structure by Escuela et al.[37] and also to classify fetal heart rate monitoring data by Georgoulas [47]. GE has also been used to aid in plagiarism detection [21].

The area of design has also been explored. Hemberg and O'Reilly [60] use GE to evolve digital surfaces and O'Neill et al. [110] use GE to evolve shelters. Fractal curves have also been evolved using GE [119]. Also along the lines of design, Murphy uses GE to optimize horse gait animation [101]. Evolutionary music has also seen some attention from GE researchers. Reddin et al. [130] evolved elevator music. Shao et al. [146] compose and evolve music using by making GE interactive.

Drchal and Šnorek [36] use GE to optimize both weights and topologies of neural

networks. Cullen [27] uses a hybrid of GE and Evolutionary Meta Compilation[26] to successfully evolve digital circuits. Şen and Clark [25] and Wilson and Kaur [172] use GE to evolve intrusion detection on two different types of systems.

The domain of video game controllers has also been examined by GE practitioners. Galván et al. use GE to evolve Java code to play the Ms. Pac-Man game [85, 86]. It has also been used to evolve behavior trees to play one variant of the Super Mario game by Perez et al.[125].

Source code for GE may be obtained from a number of places. Table 2.1 gives some implementations of GE and their relevant information.

2.6 Summary

This chapter outlines the GE algorithm and how it most notably differs from standard GP. More specifically, it covers how individuals are represented and modified by evolutionary operators in GE. It later covers some of the problem areas to which GE has been applied and various implementations of GE. In the following chapter a history of approaches to modularity in GP and GE is presented.

Chapter 3

A Background of Modularity in Genetic Programming

Before moving to the experimental chapters of this thesis, an understanding of how modularity can be defined is needed. It is also important to have an understanding of previous work in modularity and GP. This chapter covers these topics by first reviewing the origins (Section 3.1) and definitions of modularity and modules give by previous researchers in Section 3.2. Section 3.2.1 also covers different application areas of modularity. Next, Section 3.3 presents a survey of approaches to modularity in GP. Finally, Section 3.4 summarizes the gaps in GP modularity literature and how the remainder of this thesis addresses those gaps.

3.1 Origins of Modularity

Before examining how modules and modularity can be defined, this section gives a brief history of how modularity came into use by EC practitioners. The pioneering work of Wallace [169] and Darwin [28] propose that species in nature are able to evolve and adapt

to their environment through natural selection. Borrowing from these principles, it is also possible to evolve computer programs to solve problems [61, 73, 48]. However, the natural evolution discussed by Darwin [28] and Wallace [169] took place over many years, which is an unrealistic time frame for solving many computer science problems. Wagner [162] discusses that in order to more efficiently harness the power of evolution, computer scientists have become more interested in what aspects of evolution encourages a faster adaptation in members of the evolving population. How quickly the population is able to adapt to solve a given problem can be thought of as "evolvability." Altenberg [6] defines an Evolvability Theorem which can be summarized as *the probability a population generates individuals fitter than the current best fit individual*[1]. This can also be thought of as individuals improving in an accretive or stepwise manner. In order for this type of adaptation to take place, individuals must be able to improve without, or only minimally, sacrificing previously discovered, beneficial information. The concept of modularity comes into play here. By enabling some form of modularity, it is possible to protect parts of individuals' genotypes that encode a specific functionality from disruption due to mutation or crossover operations.

[1]Wagner [162] gives a more detailed definition of Altenberg's Evolvability Theorem: the probability that a population generates individuals fitter than any existing is

$$F\left(w_{max}\right)^{=} a\left\{R\left(w_{max}\right)+\bar{\beta}_{u}\left(w_{max}\right)+Cov\left[\beta_{jk}\left(w_{max}\right),\frac{w_{j}w_{k}}{w^{2}}\right]\right\}$$

where a is the maximal rate at which new genotypes are generated by mutation and/or recombination. $R\left(w_{max}\right)$ is the probability that a random sampling of genotypes yields a genotype with larger than the currently best, $\bar{\beta}_{u}\left(w_{max}\right)$ is the average search bias, the probability that the mutation and/or recombination of the given genotype produces better genotypes, and $Cov\left[\beta_{jk}\left(w_{max}\right),\frac{w_{j}w_{k}}{w^{2}}\right]$ is the covariance between the current fitness of genotypes and the probability to produce betters ones by mutation and/or recombination.

3.2 Defining Modules and Modularity

Modularity is an abstract concept that involves a problem or object being *decomposable* into elements which are able to operate independently. However, in the real world, the decomposition of problems into completely independent sub-problems is unrealistic in most cases. However any problem is divided, the different components will interact with each other at least to a small degree. It is also possible that some problems are unable to be subdivided and decomposed into more manageable pieces. Because of this, Simon [148] suggests that problems should instead be referred to as *nearly decomposable*. He goes on to give two examples of this decomposition in the form of parables about two watchmakers and opening a lock on a safe. The following is a paraphrasing of Simon's [148] two watchmaker parable.

> Hora and Tempus are two watchmakers who craft exquisite watches and constantly receive phone calls with orders for more watches. Hora profits greatly and his business succeeds, however Tempus is unable to keep up with the demand for his watches and his business fails. Both watchmakers use 1000 parts in each watch. The difference between the watchmakers is their watchmaking technique. The method Tempus uses to build his watches requires every piece to be in place before he can set a watch down. If interrupted, the watch falls apart and he must start assembling from the beginning. If his phone is constantly ringing, he is frequently interrupted and must restart building his watches. Hora developed a method where his watch may be assembled in a hierarchical manner. At the lowest level of the hierarchy are sub-assemblies of parts comprised of 10 parts each. Once 10 of these sub-assemblies have been completed, the can be combined into a larger sub-assembly, and 10 of the latter sub-assemblies can be combined to create the entire watch. Before and sub-assembly is completed,

> if parts or put down they will fall apart. However, a completed sub-assembly will remain together if left alone. If Hora is interrupted while working on a sub-assembly of a watch, only the progress on that sub-assembly is lost.

The moral this parable is that the more decomposable or "modular" approach to building watches is preferable because it allows for a type of checkpointing while building a watch. Hora puts together small amounts of pieces to create self-contained assemblies of the watch. They do not rely on any of the other (sub)assemblies to hold their form. By creating a number of the initial sub-assemblies and treating each one as a single element instead of a group of elements, Hora is able to solve small pieces of the watch making problem at a time. After solving the smaller pieces of the problem, he can use the combined pieces to create larger assemblies. Now, a summary of Simon's [148] safe lock example is given.

> Imagine a safe with 10 dials where each dial has 100 settings. This means there are 100^{10} different combinations for the safe. Assuming one would on average need to attempt half of the possible combinations before finding the solution, 50 billion billion combinations would need to be attempted. However, if the safe had a defect and each dial made a clicking noise when it was set to the correct combination, the number of combinations to search would be greatly reduced to $10 \times 50 = 500$. This reduces the problem from being practically impossible to being fairly easily solved.

In the safe lock example, Simon shows how it may be possible to modify a problem such that it can be decomposed into more easily solvable sub-problems. By modifying the safe in the example, he shows that decomposing the problem into 10 sub-problems the combinatorial complexity of the problem as a whole decreases. Once a single sub-problem is solved, it can be left alone, and focus can be shifted to other, unsolved sub-problems. This type of approach to problem solving can be referred to as "selection by components" [148],

where each "component" can be considered a module.

While Simon [148] explains modularity in a more general sense, there has also been much effort given to the definition of modularity in EAs. It is common for researchers [29, 30, 31, 32, 134, 135, 136, 137, 139, 140] to consider finding modules in GP the same as identifying building blocks.[2] Using this definition, modules must be low-order schemata of above-average fitness. However, it is possible to discover modules that are neutral or even detrimental to the population. Because a building block must improve a solution's performance, it is important to understand that a module is not necessarily a building block. Modules may indeed be harmful or neutral, which violates the definition of a building block. Garibay et al. [42, 43, 44, 45] show that the search space increases when identifying modules, regardless of the quality of the modules discovered. They show that the increased search space may be overcome if the discovered modules are sufficiently useful.

After this discussion of how modules may be defined, it is important to recognize that modules may be considered as one or both of the following:

1. Modules may be an entity with more internal connections than external connections. By thinking of modules in this way the human brain (or the brain of any other creature) can be considered a module. The brain is extremely interconnected, especially when compared to the number of connections it makes to the rest of the body.

2. Or, modules may be an entity which is repeatedly used in a larger entity. To help illustrate this view of modules, consider the floors of a skyscraper. If each floor of a skyscraper is a module, it is easy to see how repeatedly placing identical floors on top of each other is able to create the building as a whole.

It is important to note that these understandings of modules are not exclusive of each other. The human feet illustrate this. Human feet each have many components such as

[2]More information on building blocks and schema can be found in the work of Holland [61] and Goldberg [48].

bones, tendons, muscles, and, in the majority of cases, five toes per foot. The high level connections between these components allow the foot to be considered a module because it has more internal connections than external. In addition to being highly interconnected, humans generally have two feet which are mirrors of each other. In the case of human feet, two mirror-identical, highly interconnected elements are used as part of a larger entity, the body. In Section 3.3, numerous approaches for incorporating modules into EC systems are presented. It is important to realize that the modules discussed in these approaches may be viewed under either or both of the classifications mentioned here. It should also be emphasized that modules in the context of GE are encapsulated sub-derivation trees containing more than one primitive or preexisting module, or a combination of primitives and modules.

3.2.1 Applications of Modularity

Modularity is not a concept that is important only in GP systems. It has been shown to be useful in a variety of areas of computer science. This section outlines some of these areas and popular work in those areas.

Software Engineering

One of the most obvious applications of modularity for a computer scientist is in programming. It is commonly understood that any large software engineering project should be broken into separate packages, files, classes, and functions. Even smaller programming undertakings should at least be divided into separate functions. Similar to modularity in biological systems, modularity in programming preserves strong linkages between functionally similar pieces of code. Meyer [94] lists a number of reasons why software should be implemented in a modular fashion. Of those, some of the most familiar to GP practitioners will be *reusability* and *robustness*. These two features of modular software design enforce

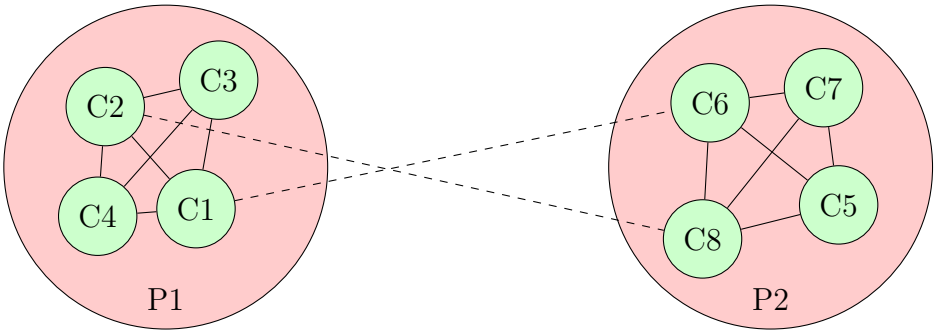


Figure 3.1: This figure shows two software packages (P1 and P2) each with a number of classes (C1-C8) inside them. The classes within each respective package are used by each other (denoted by solid lines). It is also possible for a class to interact with classes from a different package (denoted by a dashed line).

the benefit of code reuse that is much-discussed with examining modularity in GP. An example of a modular software system can be seen in Figure 3.1.

Neural Networks

Modularity has also been shown to be beneficial in the realm of artificial neural networks (ANNs) [141]. Some related work in modular neural networks has been done by Gruau [49] who uses a GA to create sub-networks which may be used as modules in a larger neural network. In later work, Gruau [50] they alter the GA to use a form of Automatically Defined Functions (ADFs) which encode sub-neural networks that can be reused by other neural networks. Modularity has also been examined more recently in the HyperNEAT [154] algorithm. Clune et al. [22] show that adding modularity to HyperNEAT improves its performance on a simplified version of the Retina Problem, but even with modularity it is still unable to perform well on the complete Retina Problem [161]. Verbancsics and Stanley [161] also extend HyperNEAT to create modular neural networks but are able to significantly increase the performance of HyperNEAT on the Retina Problem.

Robot Behavior

Another area that has seen the benefits of incorporating modularity is robotics. Early work Calabretta et al.[18, 19, 20] use a combination of a GA, a neural network, and modules to evolve a robot controller which clears garbage from an arena. Other work training robots comes from Olmer et al. [104] and Saunders et al. [143] who decompose their respective problems and into more easily solvable sub-tasks.

Design

Evolutionary design researchers have also shown that modularity is beneficial when using EC methods for their various design tasks [82, 83]. Hornby [64] shows that enabling modularity, hierarchy, and regularity improve the scalability of evolutionary design systems. Hornby et al. [63] also show that using a representation which enables modularity, hierarchy, and regularity is able to evolve more scalable designs for 3D robots. The importance of modularity is also discussed in the context of designing commercial products (lamps in this particular example) [65].

3.3 Modularity in Genetic Programming

Modularity has been a popular topic for GP research since the early work of Koza's Automatically Defined Functions (ADFs) [74] and Angeline and Pollack's Genetic Library Builder (GLiB) [9]. In GP, incorporating modularity into an existing system can be accomplished in numerous ways. Modules may be extracted from solutions already discovered by evolution or may evolve on their own. How this is done depends on the aims of the developer and the representation they choose for their GP system. Methods like this typically use modularity as an afterthought or an extension of a GP system. Finding modules in this way can be considered taking a "top-down" approach to modularity. However, it

is also possible to represent solutions by encoding them in an explicitly modular fashion. Contrary to the "top-down" approach, enabling modularity in this way can be thought of as a "bottom-up" approach. Figure 3.2 illustrate how primitives, modules, and solutions can be created in the top-down and bottom-up approaches. This section presents how researchers have approached modularity using these kinds of methods. Due to the fact that there are so many methods and slight variations of these methods, Tables 3.1 and 3.2 list these approaches and how they discover and make modules available to evolution.

It is important to note that the approaches listed in Sections 3.3.1 - 3.3.3 all focus on discovering modules or generating individuals in a modular fashion, but there are other veins of modularity research in GP which are more theoretical or do not focus on the discovery of modules. One such work is that of Woodward [173], where he shows that the size of a solution in GP is independent of the function set when modularity is permitted. The significance of this is that modules are able to remove the bias of the function set. Another such body of work comes from Krawiec and Wieloch [78, 77], who examine functional modularity. They use the term *functional modularity* to describe modules whose semantics fulfill requirements of some subgoal of a problem. Based on their results, Krawiec and Wieloch propose that functional modularity may provide better scalability on certain problem classes. In more recent work Krawiec [75] explores the semantics and modularity of programming tasks. He shows that there exists large variations of semantic characteristics among programming tasks and that this variation affects the modularity of the problem. He also concludes that many real-world programming tasks may be inherently modular, not just benchmark problems constructed with a modular design in mind.

3.3.1 Automatically Defined Modules

This section presents some of the earliest and most popular methods for automatically defining modules in GP. The approaches discussed here incorporate modules into GP sys-

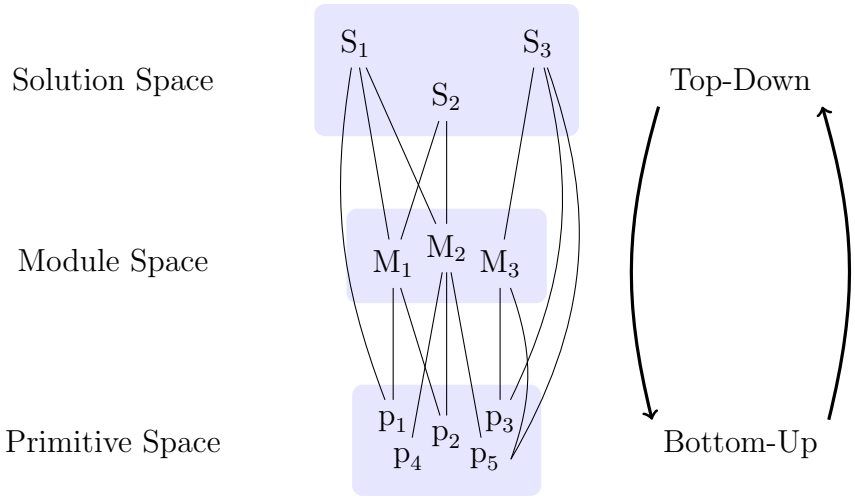
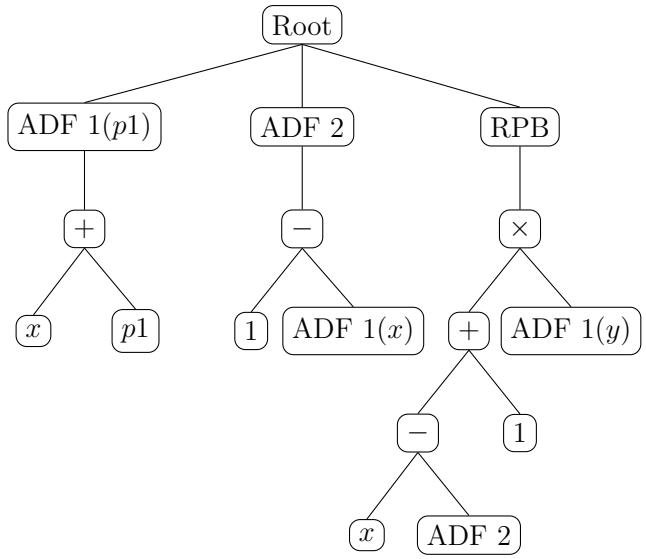


Figure 3.2: This figure demonstrates the top-down and bottom-up nature of how modules may be identified or created. Modules may be created by combining primitives from a problem which can then, in turn be used in solutions. Modules may also be identified by looking at solutions and determining which primitives function together as a module.

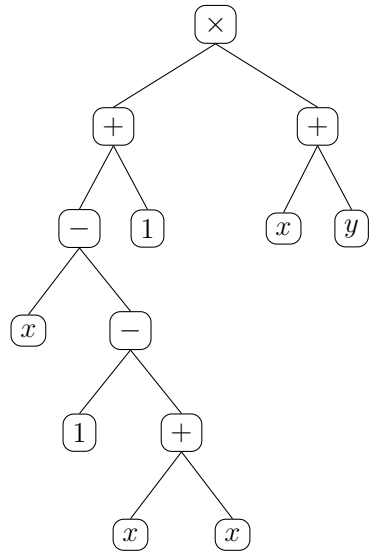
tems largely based on Koza's ADFs [74]. Many of these approaches use pre-defined numbers of modules which can be evolved by crossover and mutation operations. Some of the work in this section begins to examine more explicit methods of module identification, but the majority of those methods are presented in Section 3.3.2.

Automatically Defined Functions

One of the earliest and most influential examples of modularity in GP are Koza's ADFs [74]. In standard GP, the representation of an individual changes from a single tree to a tree divided into a results producing branch (RPB) and branches defining ADFs. ADFs are simply parameterizable subtrees created from the original GP function and terminal set. The function and terminal sets for the RPB are extended by adding calls for any ADFs. It is also possible for a user to allow ADFs to call each other, but care should be taken to ensure that ADF calls do not create an infinite loop. An example of a GP individual with ADFs and the same individual without ADFs is given in Figure 3.3. They can also



(a) A GP individual with ADFs



(b) The GP individual from Figure 3.3(a) with ADFs converted into a standard GP individual

Figure 3.3: This figure shows an example GP individual using ADFs (Figure 3.3(a)) and how that individual would look without ADFs (Figure 3.3(b)). In Figure 3.3(a), ADF 1 takes a single parameter, $p1$, and returns the result of adding that parameter to x. ADF 2 takes no parameters and calls ADF 1 with the parameter x. The RPB uses a number of functions and terminal symbols as well as both ADFs.

be evolved along with individuals using crossover and mutation operations. The benefit of ADFs is that they facilitate the reuse of information and functions. When ADFs are able to call each other, new information can be discovered in a hierarchical fashion. However, mutation and crossover operations may alter an ADF in a way that reduces the fitness of an individual or individuals. In his much-cited book discussing ADFs, Koza [74] shows how GP using ADFs is able to outperform standard GP on a number of benchmark problems of sufficient difficulty.

As ADFs are one of the oldest forms of modularity in GP, they have been analyzed, extended, and applied to a number of different benchmark problems. Koza [72] goes on to extend ADFs with numerous architecture altering operations to create, delete, and duplicate ADF arguments or ADFs themselves. He shows how his architecture altering

operations are able to solve a wide variety of benchmark problems. An early vein of work by Bruce [15, 16] applies GP with ADFs to the problem of generating object-oriented programs. Bruce [17] later went on to use ADFs with a stack-based form of GP [126] on the Lawn Mower problem.[3] He showed that ADFs are able to significantly improve the performance of his implementation of stack-based GP. Jassadapakorn [66] also experiments with ADFs, but instead of allowing a fixed number of ADFs, individuals can have anywhere from $0 - 16$ ADFs. Brock [14] uses ADFs across multiple runs. In his work, Brock [14] attempts to solve Even Parity problems of increasing difficulty. He begins by attempting the Even 4 Parity problem. When an individual solves it, that individual's ADFs are saved and added to the function set of the RPBs and ADFs of another run to solve a more difficult Even Parity problem. He shows that saving ADFs of good individuals and seeding them into new evolutionary runs greatly reduces the number of fitness evaluations needed to solve the Even Parity problems compared to GP with standard ADFs. O'Reilly [118] examines ADFs by applying GP with ADFs to Even Parity Problems. She extends a simulated annealing [131] algorithm by adding ADF functionality. Then, she compares the results of using standard GP, GP with ADFs, simulated annealing, and simulated annealing with ADFs. The results of this work showed that using simulated annealing with ADFs was superior as Even Parity problem instances became more difficult. One common trait about research outlined here is that there is no standard functionality to evolve the ADFs themselves. For most of the work mentioned, the basic mutation and crossover operations are used. Operators to alter ADFs are sometimes defined, but when and how the ADFs are changed is random. Because ADFs have the potential to be very influential in individuals, simply allowing evolution to change them via mutation and crossover events may be sub-optimal. It may be possible to develop a method to optimize ADFs and further improve GP's search potential.

[3]Descriptions of the Lawn Mower, Even Parity, Santa Fe Ant Trail, Symbolic Regression, and other common GP benchmark problems can be found in Koza [74].

One body of work by Ahluwalia and Fogarty [5] tries to optimize ADFs before they are used by individuals. They accomplish this by creating a sub-population for each of two ADFs available to the main individuals. The populations for the main individuals and ADFs are only allowed to exchange information with members of their own population. References to ADFs are pointed to the best ADF at each generation. Later, Ahluwalia et al. [4] experiment with various ADF selection strategies showing that selecting ADFs based on the fitness assigned to them during the evolution of the sub-population and linking ADFs both an individual in the main population and to another ADF are the best performing methods. Ahluwalia and Bull [2] go on to coevolve ADFs which are used as preprocessors for a k-nearest neighbor algorithm. Ahluwalia and Bull [1] further explored ADFs in GP by creating what they termed evolution-defined functions (EDFs). The EDF approach allows an unlimited amount of ADFs to be created using compress and expand operations based on Angeline and Pollack's [9] Genetic Library Builder (GLiB) (GLiB is described in more detail in Section 3.3.2). Ahluwalia and Bull [1, 3] show that EDFs outperform a number of other approaches on a 20-bit Multiplexer problem a selection of classification problems.

In a similar fashion to the EDF approach, Jonyer and Himes [67] developed Subdue ADFs (SADFs). The SADF method sends a percentage of the most fit individuals to the Subdue system [23]. The most frequently occurring subtrees are returned, and one of these subtrees is converted into an ADF. Once an ADF is created, its definition is added to some of the worst performing individuals and all occurrences of the subtree from which the ADF was created are replaced with references to the ADF. Although they do not present any comparisons of performance against other GP approaches, Jonyer and Himes give insight into the how the parameters of their SADF approach should be considered.

Higher-Order Functions and Lambda Abstraction

In other work, Yu [174] adds additional functionality to CFG GP by enabling higher-order functions and lambda abstraction. By doing this, individuals are able to generate and use recursive programs with no danger of not terminating. This approach was tested on a variety of Even Parity problems with the goal of evolving solutions that are capable of solving any size of Even Parity problem. Yu [174] accomplishes this by evolving solutions for the Even 2 Parity and Even 3 Parity instances and showing that there are solutions generated by these instances which can solve any size of Even Parity problem. She also shows that GP with higher-order functions and lambda abstractions are more efficient at find solutions than standard GP and GP with ADFs.

ADFs in GE

In addition to the ADFs implemented in GP, ADFs have also been implemented in GE. O'Neill and Ryan [113] first implemented ADFs in GE by adding ADF definitions into GE's grammar. Implementing ADFs in this manner allows the genotypes of the individuals to determine each ADF definition and usage in their respective individuals. They show that GE with ADFs is able to solve more instances of the Santa Fe Ant Trail problem in fewer generations than standard GE. Hemberg et al. [59, 57] similarly extend GE by incorporating ADF definitions into GE's grammar. However, contrary to the single ADF O'Neill and Ryan [113] encode into GE's grammar, Hemberg et al.[59, 57] allow for potentially unlimited ADFs to be defined by individuals' genotypes. They show that, once again, GE using ADFs is able to significantly outperform standard GE on the Santa Fe Ant Trail, as well as the Los Altos and San Mateo Ant Trail problems. On the other hand, Hemberg et al. [59, 57] also show that GE with ADFs showed no significant improvement in performance on a number of Symbolic Regression instances. Harper and Blair [55, 52] also extend GE with what they call Dynamically Defined Functions (DDFs). DDFs have

a similar function to Hemberg et al.'s [59, 57] GE with ADFs in that DDFs allow GE individuals' genotypes to determine the quantity and definition of ADFs present in each individual. In his exploration of DDFs [55, 52], Harper uses Koza's Minesweeper problem [74] and shows that GE with DDFs is able to solve more instances of the problem than GE with ADFs and standard GE. Rodrigues and Pozo [133] also used a grammar-guided form of GP combined with ADFs and show that using ADFs gives a performance increase over standard grammar-guided GP on Even 3, 4, and 5 Parity problems.

Macros and Tags

Two additional forms of modularity that are similar to ADFs have also been developed. The first of these to be discussed are automatically defined macros (ADMs) [149]. Spector [149] defines a macro as "operator that performs source code transformations." In the setup described by Spector [149], macros are used in place of ADFs on instances of the Lawn Mower, Dirt-Sensing, Obstacle-Avoiding Robot, and Wumpus World problems. The second form of modularity is the use of tags [62]. In GP, tags are a mechanism for naming, or assigning an address to, a block of genetic code. Once a portion of code has been tagged, the entire block may be used by calling that tag. Spector et al. [152] use tags with the PushGP [150, 153] system, as well as standard, tree-based GP [151]. When using PushGP, Spector et al. [152] show that tags are able to significantly improve PushGP's performance. They also show that the same improvements in performance are not always seen with tags in tree-based GP [151].

3.3.2 Explicitly Discovered Modules

ADFs have been a popular vein of modularity research in GP, but there are also many other ways to incorporate modules into various GP systems. The methods presented in this section add modules to their respective forms of GP by developing operators specifically

to handle the identification, usage, and/or deletion of modules.

Tree-based Modularity

One of the first and most cited methods for identifying modules comes from Angeline and Pollack's GLiB [9, 8]. In this approach, a roulette selection is used to pick an individual from the population to contribute a module. Then a random subtree from this individual is compressed to a user-defined depth, and a new function is created using the compressed subtree. Any branches of the subtree below the given depth are turned into parameters. Angeline and Pollack [10] also experiment with GLiB by evolving Tic Tac Toe and Tower of Hanoi players, but they state that GLiB does not add any benefits in terms of performance to these problems. Kinnear [71], however, compares two GP with ADFs to GLiB and finds no significant differences in performance.

A similar approach is Rosca and Ballard's adaptive representation [134, 135, 136]. Rosca and Ballard [134, 135, 136] create modules out of small subtrees that have been created in the current generation by crossover and mutation operations. Once modules are discovered, the function set used to create individuals is extended with the modules. Then, a user-defined number of the least fit individuals are replaced with randomly generated individuals from the newly extended function set. Rosca and Ballard [136, 135, 134] show that their AR method finds less complex solutions to a number of Even Parity problems in less generations than standard GP and GP with ADFs. In later work, Rosca and Ballard [134, 137, 139, 140] extend the AR framework and create the Adaptive Representation through Learning (ARL) approach. The ARL method differs from AR in that ARL only makes modules from individuals that have a better fitness than their least fit parent. From these individuals, only the most activated subtrees within a user-defined depth limit are used as modules. Once these subtrees have been identified, a random selection of their terminal symbols are converted to parameters. In order to maintain a reasonable number of modules, the average

fitness of individuals using each module over a fixed number of generations is kept for each module. When needed, modules with the worst fitness values are replaced by newly created modules. Rosca and Ballard show that ARL outperforms a random, simulated annealing, GP, and GP with ADFs on evolving Pac-Man controllers [137, 139, 140].

A later form of module encapsulation is tested by Roberts et al. [132]. They perform a number of independent GP runs and use individuals from the best run for subtree encapsulation. Each of the subtrees created during evolution are evaluated on their own. The fitness returned by this evaluation and the subtree is stored in a subtree database. After a user-defined number of runs are completed, the subtree database from the best run is analyzed and subtrees which have the same fitness are grouped together. A user-specified amount of the most frequent subtrees from the fitness groups are encapsulated into terminal symbols and added to the terminal set used in later GP runs. This process is repeated until the termination criteria is reached. Roberts et al.[132] report that on a target detection task, their approach for subtree encapsulation finds better solutions faster than standard GP and is often able to reduce the false alarm rate by up to 75%.

Majeed and Ryan [88] also propose another method for identifying modules. They identify modules by calculating their contribution to the fitness of an individual by replacing the subtree with the identity function of the module's parent node [89]. This is done in the final generation of a run and only individuals with fitness better than the mean fitness of the population are used to contribute modules. Only a limited number of modules can be kept, so Majeed and Ryan [88] define a fitness function for modules based on their previously calculated contribution, how many times they appear in the population, and how many times instances of that module were taken from individuals. If there are more modules in the repository than the size limit, modules with the worst fitness are removed until the repository is within its size limit. Once the module repository has been created, a new GP run is started and modules are mutated into individuals. Majeed and Ryan [88] show

that mutating modules into individuals based on their fitness and randomly outperforms standard GP on the $x^4 + x^3 + x^2 + x$ Symbolic Regression problem.

Grammar-based Modularity

Another method for identifying modules is Whigham's work on inductive grammar bias [170, 171]. In this work, Whigham uses a grammar-based form of GP to evolve solutions to the 6 multiplexer problem. In his grammar bias method the most fit individual in the population is used to find a useful sub-derivation tree. From that individual, one the deepest terminal symbol is chosen and propagated up the tree to the next production with at least one additional terminal or non-terminal symbol at the same level. Next, the altered production is added to the CFG used to generate individuals. Whigham [170, 171] also allows entire sub-derivation trees to be encapsulated and added to the grammar. After the grammar has been modified, productions are picked based on how often they are used in the population. The results of this work show that biasing the grammar led to the probability of a run finding a solution to increase from 22% to 66%. While is it never stated as such, Whigham's work [170, 171] can be thought of as discovering modules in grammar-based GP and incorporating them into the evolving population through the grammar.

More work in modularity and grammar-based GP has been carried out by Georgiou and Teahan [46]. They define one method for discovering modules in GE called Constituent GE (CGE). CGE discovers modules by randomly created small genotypic elements and evaluating them on a user-defined sub-set of the fitness function used during the actual evolution of the main population. This is similar to the work of Roberts et al. [132] and Majeed and Ryan [88] in that a preliminary run is performed to identify module before they are actually used to evolve complete solutions to the given problem. The phenotypes of the most fit genotypes are added to a grammar that will be used for the evolution of the population. Georgiou and Teahan [46] show that CGE is more effective at solving the

popular Santa Fe Ant Trail benchmark problem as well as the Los Altos Hill [73] and the Hampton Court Maze [160] problems than standard GE using two different grammars.

Linear GP Modularity

More work identifying modules has been carried out by Li et al. [79]. In their work, modules are discovered for the linear form of GP, Prefix Gene Expression Programming (P-GEP) [80]. In this work, Li et al. [79] look for low-complexity, frequently occurring, sub-structures in the elite individuals in the population when a new best-of-population individual is found. A user-defined limit is used to control how many of the frequently occurring sub-structures are compressed and made available to the population as modules. The elite individuals are then updated by replacing occurrences of the expanded modules with the compressed versions. Modules in individuals which are no longer among the most frequently used are expanded to their original form. Li et al. [79] show that their P-GEP with modules performs at a similar level to standard P-GEP. In further work, Li et al. [81] extend their work by introducing "loose modules." These modules only encapsulate genes that encode functions in P-GEP and leave any parameters to these functions up to evolution. The mechanism for determining which loose modules are kept or expanded is the same as in Li et al. [79]. Li et al. [81] show that for Koza'a two box problem [74], P-GEP with loose modules performs better than GP, GP with ADFs, and standard P-GEP.

Parent et al. [123] also incorporate modularity into a form of linear GP, which they call compressed GA (cGA). Before any evolutionary operations are performed on the population, modules are identified by compressing adjacent elements of the genotype into a single element. The most frequent of these compressed elements are stored in a dictionary. Next, a percentage of the population is selected by roulette selection, and if any of the expanded forms of modules in the dictionary exist in the selected individuals, they are replaced by the compressed form of the appropriate module. Once individuals are compressed, selec-

tion, crossover, and mutation operations are performed on the population. Before they are evaluated, each compressed individual is decompressed, replacing all the compressed genotypic elements with their original forms. The cGA's performance is compared to two common GA problems, OneMax and SEQ, where cGA outperformed a GA on OneMax but performed poorly on the SEQ problem. The cGA was also used on three GP problems (Even 5 Parity, $x^4 + x^3 + x^2 + x$ Symbolic Regression, and $cos(x) + sin(x)$ Symbolic Regression) and a data compression problem. On the GP problems, Parent et al. [123] report that cGA performs better then standard GA on all the GP problems examined.

Graph-based GP Modularity

The final approach in this section adds modularity to the popular form of graph-based GP, Cartesian GP (CGP) [95]. Walker and Miller [164] do so by adding a compress operation to encapsulate sub-graphs from the most fit individual at each generation into modules. These modules are then added to a module list, making them available to all the individuals in the population. Modules may also be removed from the module list by an expand operator. This extension of CGP is called embedded CGP (ECGP). Discovery and deletion of modules in this manner was inspired by Angeline [9, 8]. ECGP has been shown to solve numerous problem more efficiently than GP, GP with ADFs, and standard CGP [167, 168, 166, 165]. Kaufman and Platzner [68] experiment with module creation techniques for ECGP based on the age of nodes (old nodes are made into modules) and the groups of nodes that form cones. Their results show that age-based and conical modules reduce the computational effort needed to solve a variety of Even Parity, Multiplexer, and classifiers for electromyographic signals. A description of the electromyographic signals classification problem is given in Kaufman and Platzner [68].

3.3.3 Modular Representation

The previous two sections (Sections 3.3.1 and 3.3.2) present a variety of methods that have been implemented to discover and use modules in GP. These methods require users to explore a variety of methods and parameters for how modules should be identified, deleted, altered, and made available to individuals during evolution. The trade-off for the work that goes into determining these parameters is that these approaches to modularity offer the benefit of being compatible with many GP representations. This section presents alternative methods for introducing modularity into a GP system: designing a GP system which is inherently modular.

Run-Transferable Libraries

The first modular representation to discuss comes from Run Transferable Libraries (RTLs) [69, 70, 142]. When using RTLs, an individual is represented as a tree, but instead of a typical GP representation where each node in the tree contains a function or terminal symbol, tree nodes contain a floating point value (a tag) and an arity value. In a separate library, tree segments are stored based on their arity. When a tree is determining which functions and terminals to use, nodes look in the section of the tree segment library that corresponds with the arity value. The tag is used to determine which function or terminal is assigned to the tree node. The tags of tree segments in libraries can also be mutated by adding random numbers to them. Tree segments in the library may also be crossed over. Using this setup, a run of GP is performed. Over the course of this run, the library of tree segments is updated based on the frequency of use of each segment over time. During an initial run, the library is trained. When that run has finished, the trained library is used on a new problem. Keijzer [69, 70, 142], show that RTLs are able to solve Even 4 and 5 Parity and Lawn Mower problems more efficiently than GP with ADFs.

Hierarchical GP

Another modular GP representation is Banzhaf et al.'s hierarchical GP (hGP) [12]. In hGP, an individual has modules local to only that individual (like ADFs). Modules are hierarchical in the sense that they are comprised of a mix of standard GP nodes and additional modules, which in turn are comprised of GP nodes and even more modules. The maximum depth of the hierarchy of modules is user-defined. Banzhaf et al. [12] employ a modified version of the ARL [137, 140, 139, 134] method for identifying modules. In their setup, Banzhaf et al. [12] also reduce the likelihood of modules being altered by crossover and mutation operations the deeper they are in the module hierarchy. They show that hGP outperforms a four Symbolic Regression problems and the Even 7 Parity problem.

Dynamic Library GP

An approach similar to hGP [12] is Dynamic Lattice GP (DL^{GP}) [129]. Individuals in DL^{GP} are comprised of subroutines from a hierarchy of layers each with a number of sub-populations. Sub-populations in each layer evolve independent of each other and are able to use subroutines from any sub-population at a lower level in the hierarchy. The individuals in each population use the standard GP crossover and mutation operations but are also given additional mutation operators. DL^{GP} uses a weighted function measuring how subroutines are used in main individuals and fit those individuals are to determine a fitness value for each subroutine. Racine et al. [129] also discuss possible selection schemes for evolving the various populations. No experimental results are presented in this work.

Shared Grammar Evolution

Another modular representation is the shared-grammar approach introduced by Luerssen and Powers [87]. In their work, Luerssen and Powers [87] define a new approach called Shared Grammar Evolution (SGE). In SGE, individuals are defined by a unique, deter-

ministic grammar called an i-grammar. Each i-grammar is specified using productions from a p-grammar which is created from the union all the productions in each individual's i-grammar and the productions specified in a user-defined population axiom. A mutation operation is employed to create new productions in the i-grammars in the population. After individuals have been mutated, they are evaluated and the productions present only in the worst individuals are removed from the p-grammar and/or population axiom. Any newly created productions which are not removed due to their presence in only the worst individuals are added to the p-grammar for further use in the population. Luerssen and Powers [87] show that SGE outperforms standard GP on the binomial-3 ($(x+1)^3$) Symbolic Regression and Santa Fe Ant Trail problems. They also state that SGE outperforms Cartesian GP, and standard GE on the Santa Fe Ant Trail. However, on the 6-Multiplexer problem SGE is not as competitive and is outperformed by standard GP, GP using tree-adjoining grammars, and GP using a CFG.

Table 3.1: This table outlines the approaches to modularity discussed in Section 3.3 and how they implement methods for identifying modules.

Approach	**Module Acceptance Criteria**	**Where Modules are Selected**	**When Modules are Selected**	**Evolvable/ Parameterized**	**Module Size**
GLiB [9, 8, 10, 7]	Random	Roulette	Every generation	Yes (branches exceed depth)	Random within depth
AR [136, 134]	Standard fitness function	Subtrees created from crossover and mutation	Every generation	Both	Random within depth
ARL [140, 134, 140, 138, 137]	Most activated subtrees	Most fit offspring with fitness better than its least fit parent	Every generation	Both	Random in depth (3–5)
ADFs [74]	NA	NA	NA	Yes	Same as RPB
ADFX [66]	NA	NA	NA	Yes	Same as RPB
Ahluwalia [5, 4, 2]	NA	NA	NA	Both	Same depth as individuals

Module identification methods continued ...					
EDFs [1, 3]	NA	NA	NA	Both	Same depth as individuals
DDFs [55]	NA	NA	NA	Yes	Same depth as RPB
ADMs [149]	NA	NA	NA	Both	Same as RPB
SADF [67]	Frequency based	Most fit individuals	Every generation	Parameterized	Unlimited
Tags [152, 151]	None	Any individual	Anytime	Evolvable	Unlimited
Majeed [88]	Hybrid utility/ frequency function	Individuals with fitness greater than pop. mean	End of an initial run	No	None
RTLs [69, 70, 142]	Library	NA	Fixed step	Evolvable	None
Roberts et al. [132]	Frequency of semantics	Everywhere	Between GP runs	No	Unlimited
ECGP [166]	Random	Anywhere	Anytime	Both	Unlimited
Whigham [170]	Based on depth	Most fit individual	Anytime	Parameterized	Grammar-bound
Rodrigues [133]	NA	NA	NA	Both	Same as RPB
Hemberg [57, 59, 113]	NA	NA	NA	Evolvable	Same depth as RPB
Li [79]	Frequency-based	Elites (top 4 individuals)	Anytime	No	Complexity of 1
hGP [12]	Replace module with random subtrees	Most fit offspring with fitness better than its least fit parent	Every generation	Both	Random in depth (3–5)
ADATE [105, 108, 106, 107]	Pop. fitness based	Created from primitives/ existing functions	Anytime	Yes	No
DL^{GP} [129]	Weighted function of usage and fitness statistics	NA	NA	Both	NA
cGA [123]	Unique substring of given length	Roulette selection of individuals	every generation	No	User-defined length
SGE [87]	NA	NA	NA	Evolvable	Unlimited
Yu [174]	NA	NA	NA	Both	Unlimited

Table 3.2: This table outlines the approaches to modularity discussed in Section 3.3 and how they use/make available to the population modules they've found.

Approach	**Module Representation**	**How Modules Are Incorporated into the Population**	**How Modules Are Stored**	**Maximum Number of Modules**	**Module Replacement**
GLiB [9, 8, 10, 7]	LISP function	Seeding/ crossover	Library/ added to primitive set	Unlimited	Removed when not used in pop.
AR [136, 134]	LISP function	Seeding/ crossover	Function set	Unlimited	None
ARL [140, 134, 140, 138, 137]	LISP function	Seeding/ crossover	Function set	Unlimited	None
ADFs [74]	LISP function	Initialized	Local to individual	Multiple of pop. size	NA
ADFX [66]	LISP function	Initialized	Local to individual	0–16 per individual	NA
Ahluwalia [5, 4, 2]	LISP function	Initialized, fitness, and random selection	Per-ADF sub-populations	2	Fitness-based
EDFs [1, 3]	LISP function	Initialized and fitness selection	Per-EDF sub-populations	2/unlimited	Compress/ expand operators
DDFs [55]	Function defined by mapping	Initialized	Local to individual	Unlimited	NA
ADMs [149]	LISP function	Initialized	Local to individual	Multiple of pop. size	NA
SADF [67]	Parameterized function	Seeded	Library	Unlimited	None
Tags [152]	Instructions are tagged	Call to tagged instructions	In tag data structure	Unlimited	Untag Instruction
Majeed and Ryan [88]	Subtree	Mutation	Module data structure	Population size	None
RTLs [69, 70, 142]	Subtree	Library elements make individuals	Library	Pop. size (500)	Frequency-based
Roberts et al.[132]	Encapsulated subtree	Initialization, crossover, mutation	Function set	User-defined value	Frequency of semantics
ECGP [166]	Compressed linear genotype	Mutation	Library	Unlimited	Random expand operator
Whigham [170]	CFG production	Seeding/ mutation	In CFG	Unlimited	None

Module usage methods continued ...					
Rodrigues [133]	Function defined by mapping	Initialized	Local to individual	Function of pop. size	NA
Hemberg [59, 57] and O'Neill [113]	ADF definitions are defined in a CFG and by evolution	ADFs can be crossed over and mutated	In CFG	Function of pop. size	NA
Li [79]	Compressed phenotype	Seeded	Module table	5	Frequency-based
hGP [12]	Encapsulated function	Seeded, Crossover, Mutation	Local to individuals	1 per individual	Based on parent's fitness
ADATE [105, 108, 106, 107]	ML function	Seeding/ crossover	Stored in individuals	Unlimited	Based on utility in pop.
DL^{GP} [129]	subtree	Individuals are build from them	In sub-populations	Function of hierarchy depth, number of pops., and pop. sizes	Based on fitness
cGA [123]	Compressed genotype elements	Randomly seeded into a fraction of the population	Module dictionary	User-defined	Modules are decom-pressed at the end of a generation
SGE [87]	Productions in a grammar	Individuals are made using the grammar with modules	In a grammar	Unlimited	Productions used exclusively in worst individuals are removed
Yu [174]	Lambda abstraction/ Higher-order function	Initialized	Local to individual	Unlimited	NA

3.4 Research Gaps

While enabling and exploiting modularity in GP has been the subject of much research, there are still many openings to be filled by future research. In particular, the study of modularity in GE has been sparsely visited. Focusing on only GE-related work, the majority of it has been the addition of a form of ADF by modifying GE's grammar [113, 59, 55]. One piece of research has been carried out that modifies GE's grammar with explicitly discovered modules [46], but this is only done before the actual evolution of

the population. There are numerous gaps in the research around exploring methods for modifying GE's grammar with modules during evolution. This thesis fills some of these gaps by exploring different methods and parameters for incorporating modules into GE's grammar and the effects these methods have on the fitness of the population.

Returning to the broader scope of GP, there are still many openings in modularity related research. Many different approaches for discovering modules have been developed over the years, some with more success than others. Despite all of this existing work, there has been minimal work examining the features of modules that make them valuable or how they are used by the population. This thesis covers some of these openings by defining new module identification operators, exploring some of the most influential parameters that impact their performance, and investigating the characteristics of modules discovered by these operators and how the modules are used by the population during evolution.

3.5 Summary

This chapter reviews all the relevant research of modularity in GP. From the work presented above, it can be seen that modularity has been identified as an important issue since shortly after GP gained popularity. There have been numerous attempts at developing modular GP systems or incorporating modularity into a pre-existing GP system. Modular approaches have been developed for standard GP and many additional forms of GP, such as linear, graph-based, and grammar-based. However, little work has been undertaken in terms of incorporating modularity into GE, aside from the addition of GP-style ADFs. The remainder of this thesis first examines how modules can be incorporated into GE and then how different methods for identifying modules in GE. It should also be restated (originally stated in Section 1.1) that in the following chapters of this thesis, a module is defined as "an encapsulated sub-derivation tree containing more than one primitive or preexisting

module, or a combination of primitives and modules."

Part II

Grammar Modification: Making Modules Available to the Population

Chapter 4

Initial Module Identification and Grammar Modification

This chapter describes the initial method used to identify and add modules to GE's grammar. It examines two methods for adding modules to the grammar, and a variety of parameters concerning how many modules should be kept and how often they should be identified. The effects of adding modules to the grammar are also explored. Recalling the research questions outlined in Section 1.2.1, this chapter answers questions 1.2.1 - 1.2.1. The work presented in this chapter more deeply examines work published by Swafford et al. [159].

There have been numerous implementations for identifying and using modules, as described in Chapter 3. A commonality between each of these is that after modules have been identified, they are made available to the individuals in the evolving population. There are many approaches that can be taken to accomplish this, but the most appropriate depends on how modules are stored after their discovery and how it is intended that they should be used by the population. For example, most approaches to modularity discover modules during an evolutionary run in anticipation that they may be used to enhance performance

of that run. One exception to this is Keijzer et al.'s Run Transferable Libraries [69], which use modules discovered in completed evolutionary runs to seed subsequent trials. The work undertaken in this thesis is implemented under the former, expecting modules to be valuable during a single run. Many canonical GP approaches simply add modules as extra components of the representation used for creating individuals [34, 134, 170]. When modules are added to the function set of a GP system, they will only be introduced into the population by mutation events or a special operator that seeds individuals with the newly discovered modules. Using GE's standard Integer Flip mutation and Single Point crossover, it is possible to incorporate modules into the population without any additional operations. After examining various methods previously used to make modules available to the population, simply adding them to GE's grammar appears to be a sensible approach. This is because it is both easy to implement and past research in modularity for grammar-based GP adds modules to the function set from which solutions are created [170]. The remainder of this chapter focuses on how modules can be added to GE's grammar and the different effects these changes to the grammar have on the population.

4.1 Discovering Modules

The first step in altering GE's grammar with modules is identifying the modules themselves. The focus of this chapter is not how modules are discovered, but rather, how the incorporation of them into GE's grammar changes different aspects of the population. For this reason, only one method for identifying modules is used in this chapter. The approach for finding modules used in this chapter is novel and was implemented for the purpose of this research. It was strongly inspired by the work of Majeed and Ryan [88, 89]. Additional methods for finding modules are the focus of Chapter 6

When finding suitable modules, a parent individual for the module must be picked

first. Once a suitable parent has been selected, a portion of that individual must be chosen as a candidate module and evaluated to determine if it should be made available to the population. The algorithm for this process is given in Algorithm 4.1. The experiments presented here use an iterative approach where each individual, I, in the population is given the opportunity to contribute a module.[1] The next step in the module identification process is picking a candidate module from the parent and evaluating it. To do this, a random node on the parent individual's derivation tree is picked as a candidate module, λ. A copy of I, I' is made and λ is then replaced by n (a user-picked integer) randomly generated sub-derivation trees of the same depth as λ. After each replacement, the parent individual, I', is re-evaluated and the difference between the fitness of I and the fitness of I' is calculated and recorded. If ρ of the recorded differences are greater than 0 for $\rho \in \mathbb{N}$ and $0 < \rho \leq n$ (ρ is user-defined), then λ is saved. The mean of the stored updated fitness values is used as the module's fitness. At this point it should be noted that the fitness of modules is maximized, as opposed to the fitness of individuals, which is minimized. If an existing module has the same sub-derivation tree as λ, λ is discarded.

Once every individual has been given the chance to contribute a module, the identified modules are sorted by their fitness values. The best $\kappa, \kappa \in \mathbb{N}$ and $0 < \kappa \leq$ population size modules are kept, where κ is also user-defined. The interval at which modules are identified, τ, is a user-specified integer representing how many generations must pass before modules are identified and added to GE's grammar. The values for κ and τ can be very important. The κ value sets a limit as to how many modules may be kept in total. This effectively sets an upper limit on how much the grammar may be changed at one time. The τ value is similar in that it sets a limit for how often the grammar is able to change.

[1] A roulette selection for the parents was tested in preliminary experiments, but the results showed no difference in performance.

Algorithm 4.1 The mutation module identification algorithm

```
M = new_list()
for Individual I in Population do
  I_f = get_fitness(I)
  λ = get_random_sub_derivation_tree(I)
  λ_d = get_depth(λ)
  i_λ = index_of_sub_derivation_tree_in_individual(I, λ)
  fitness_diffs = new_array_of_size(n)

  i = 1
  while i ≤ n do
    I' = copy_of(I)
    rand_i = create_random_sub_derivation_tree_with_depth(λ_d)
    replace_sub_derivation_tree_at_index(I', rand_i, i_λ)
    I'_f = get_fitness(I')
    fitness_diffs[i] = I_f − I'_f
    i++

  better_count = 0
  for Δf in fitness_diffs do
    if Δf < 0 then
      better_count++

  keep_candidate = true
  if better_count < round(ρ × n) then
    keep_candidate = false

  for Module μ in M do
    if get_sub_derivation_tree(μ) == λ then
      keep_candidate = false

  if keep_candidate then
    λ_f = mean(fitness_diffs)
    M.save_as_module(λ)
```

4.2 Grammar Modification by Modules

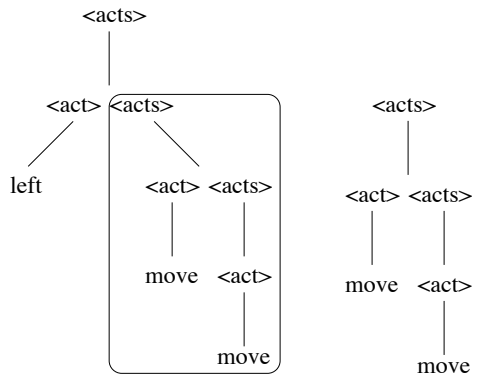
After modules have been identified, they must be made available for the population to use. This is done by simply adding the modules to the appropriate rules in the grammar. Two methods have been developed to do this.

The first of these methods takes discovered modules and adds them as productions directly to rules in the grammar depending on which rules' left-hand-symbols match the modules' root nodes. The following example explains this process in more detail. First, consider the simple grammar in Figure 4.1, an individual producible by that grammar (Figure 4.2(a)), and a module selected from that individual (Figure 4.2(b)). This module is incorporated into the grammar by taking the module's phenotype (`move move`) and making it a production of the rule matching the module's root symbol. Since the module's root symbol is `<acts>`, the module will be added to the rule: `<acts> ::= <act> | <act> <acts>`. The updated rule is now: `<acts> ::= <act> | <act> <acts> | <mod_0>`, and a new rule is created: `<mod_0> ::= move move`. For the purpose of better illustrating the differences between modifying the grammar in this manner and an additional approach that will be described later, a second module is also added to the `<acts>` rule: `<mod_1>`. An extra rule is also added to the grammar for this module `<mod_1> ::= move left move`.

The complete grammar with this modification is shown in Figure 4.3. Given the nature of the phenotype to genotype mapping in GE, it is quite obvious that modifying the grammar in this manner has the potential to be extremely destructive. When a new production is added to any rule in the grammar, it is likely that individuals using that rule will no longer map to the same phenotype. This probability that a rule retains its original mapping decreases even further as more modules are added to that rule. Consider an individual that uses the grammar in Figure 4.1 and mapping of said individual begins with the `<acts>` non-terminal and picks the `<act>` non-terminal. If a single module is added

```
<acts> ::= <act>
         | <act> <acts>
<act>  ::= move
         | left
         | right
```

Figure 4.1: A sample grammar for the Lawn mower problem



(a) A sample individual (b) A sample module

Figure 4.2: A sample individual producible by the grammar in Figure 4.1 and a sample module from that individual.

```
<acts>  ::= <act>
          | <act> <acts>
          | <mod_0>
          | <mod_1>
<act>   ::= move
          | left
          | right
<mod_0> ::= move move
<mod_1> ::= move left move
```

Figure 4.3: A sample grammar with an added module. The module has been added directly to the rule corresponding to the symbol of its root node

```
<acts>         ::= <act>
                 | <act> <acts>
                 | <acts_mod_lib>
<act>          ::= move
                 | left
                 | right
<acts_mod_lib> ::= <mod_0>
                 | <mod_1>
<mod_0>        ::= move move
<mod_1>        ::= move left move
```

Figure 4.4: The same grammar as in Figure 4.3 except using module libraries to add the module to the grammar

to the `<acts>` rule in the grammar, there is only a $\frac{1}{3}$ chance that the `<act>` non-terminal will be picked during subsequent mappings. As more modules are added to this rule, the chances of the mapping producing the same phenotype will be greatly reduced.

An alternative method was also used to modify the grammars. In this approach, module library non-terminals were added to the grammar instead of adding the module directly. Again, consider the initial grammar (Figure 4.1), individual (Figure 4.2(a)), and module (Figure 4.2(b)). Using the module library method, a new non-terminal will be added to

the rule matching the module's root node:

`<acts> ::= <act> | <act> <acts> | <acts_mod_lib>.`

Next, a module library rule is created to hold the module production:

`<acts_mod_lib> ::= <mod_0> | <mod_1>.` If multiple modules are discovered with the same root node symbol, they are all added to the same module library rule. Additional rules are created for the actual module phenotypes: `<mod_0> ::= move move` and `<mod_1> ::= move left move`. The grammar modified in this way can be seen in Figure 4.4. The original grammar when using and not using modules libraries is always the same. For example, on the Lawn Mower problem the grammar in Figure 4.1 would be the base grammar before modules are added, with or without module libraries. By using the module library non-terminals, the addition and subtraction of modules is localized to those library non-terminals. This should have the effect of less disruption to the individuals when the grammar is being modified and they are remapped. Figures 4.1, 4.2, 4.3, and 4.4 show how the grammar used in the Lawn Mower problem may be modified both using and not using modules libraries. Because changing GE's grammar means each individual's genotype-to-phenotype mapping may have changed, each individual must be remapped to use the new grammar. An example of this is given in Figure 4.5. The repercussions of this remapping will be discussed in Section 4.4.

4.3 Experimental Design

To understand how the grammar modification methods impact search in GE, the experimental setup compares the following:

1. the differences between modifying the GE's grammar using (**UL**) and not using (**NL**) module libraries;

2. the effects of modifying the grammar more or less frequently ($\boldsymbol{\tau 5}$, $\boldsymbol{\tau 10}$, and $\boldsymbol{\tau 20}$);

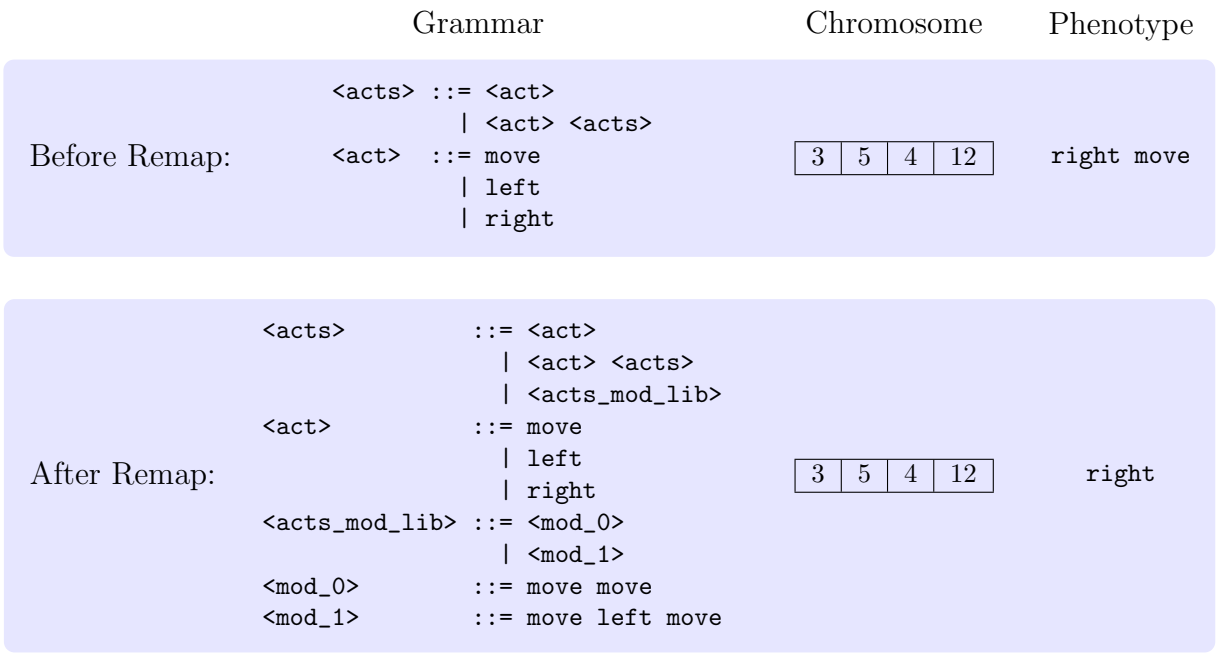


Figure 4.5: This figure shows how modifying GE's grammar may alter the phenotype of an individual

3. and the performance of these approaches against standard GE.

Table 4.1 shows the standard parameters for all evolutionary runs unless otherwise noted. For the following experiments, modules are not allowed to contain a single terminal or a single module. Modules must be comprised of multiple terminal symbols, a combination of terminals and pre-existing modules, or multiple modules. Figure 4.6 illustrates how the module identification, module replacement, and grammar modification operations are added to GE's evolutionary loop.

These variations of GE are tested on four typical benchmark problem types for GP: the Santa Fe Ant Trail, Even 7 Parity, $x^5 - 2x^3 + x$ Symbolic Regression, and 8×8 Lawn Mower problems. Fitness is minimized for each of these benchmark problems. Each of these problems is used here because it has been shown to benefit from modularity [59, 74, 127, 165].

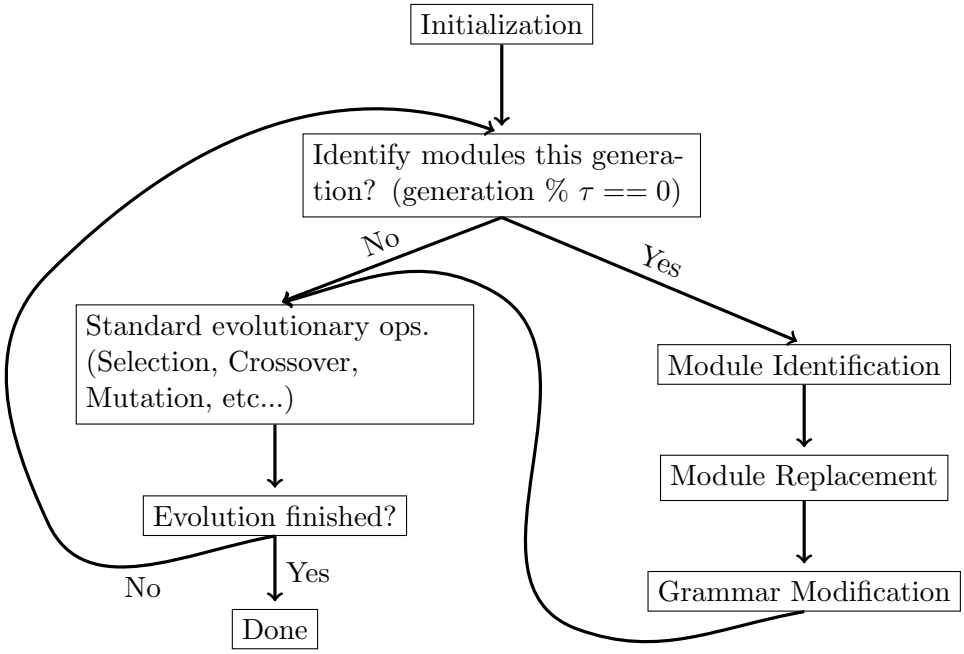


Figure 4.6: The evolutionary cycle with module identification and grammar modification

Table 4.1: Experimental setup for all evolutionary runs unless otherwise noted

Parameter	Value
Runs	50
Fitness Evaluations	100000
Population	500
Selection	Tournament (Size 5)
Wrapping	None
Crossover	Single Point (90%)
Mutation	Int Flip (1%)
Elites	50
Initialization	Ramped Half and Half
Replacement	Generational
Max. Derivation Tree Size	25 (Lawn Mower - 100)
κ	5, 20, All
ρ	75% of n
n	50 (number of evaluations per candidate module)
τ	5, 10, 20

Another characteristic of the module identification approaches is in regards to the time overhead for identifying modules and modifying the grammar more or less frequently. When looking for new modules, each individual is re-evaluated 50 times to estimate the fitness contribution of a candidate module (recall Section 4.1). After the new modules are added to the grammar, each individual must be re-evaluated again. For the experiments carried out in this chapter, each module identification and grammar modification uses at most 25500 additional fitness evaluations. While it is possible for the different variants to use fewer, they tend to use between 19000 and 24500 on average. This alone raises issues concerning how many fitness evaluations are used to identify modules. These issues are addressed in Chapter 6.

4.4 Results and Discussion

This section identifies and explains the effects of different variations for modifying GE's grammar. Primarily, the effects of using and not using module libraries in the grammar, how frequently the grammar is modified, and how many modules are used when the grammar is changed. These are all analyzed together because modifications to one of these features will impact the others. Because there are a number of variations for collecting and using modules, the effects of these are compared and contrasted against each other before comparing them to a typical GE setup. Table 4.2 lists the abbreviations used to differentiate the various approaches examined in this section.

4.4.1 Module Libraries

The first variations to be discussed are the manner in which the grammar is enhanced with modules. Recall from Section 4.2 that modules may be added to GE's grammar in one of two ways: using modules libraries or not using module libraries. The comparisons here use

Table 4.2: List of abbreviations that represent different variations of the approaches used in the following experiments

Abbreviation	Meaning
NL	Approach does not use module libraries
UL	Approach uses module libraries
κ-5	Approach uses $\kappa = 5$
κ-20	Approach uses $\kappa = 20$
κ-All	Approach keeps all modules discovered
τ-5	Approach uses $\tau = 5$
τ-10	Approach uses $\tau = 10$
τ-20	Approach uses $\tau = 20$

$\tau = 5$. When modules are identified, three values for κ are examined: 5, 20, and keeping all the modules identified. Also, approaches using (**UL**) and not using (**NL**) module libraries to add modules to GE's grammar are given. As each of the problems under examination exhibit different characteristics with these parameters, results and discussion of each will be given in turn.

Santa Fe Ant Trail

The first problem for discussion is the performance of the best grammar modification variation on the Santa Fe Ant Trail in Figure 4.7. The most notable characteristic of this graph occurs at 30000 fitness evaluations. At this point modules have been identified, added to GE's grammar, and the individuals have been remapped to use the new grammar. What this shows is the immediate, negative impact of modifying the grammar and remapping each individual using the new grammar. Some of the variants in Figure 4.7 also exhibit interesting behavior. One of the most noticeable differences is the performance of the worst setup compared to the best setup. For this problem, the combination of not using module libraries and keeping every module identified (the **κ-ALL NL** approach) had the worst fitness when it reached the fitness evaluation limit. Once modules are identified and added to GE's grammar, the average best fitness of this approach takes a drastic reduction in

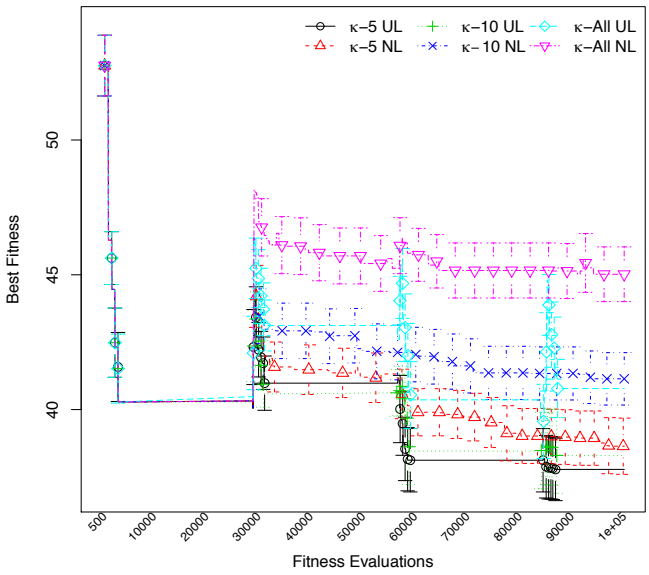


Figure 4.7: This figure shows the average best fitness and standard error for the Santa Fe Ant Trail problem.

fitness. From this change in fitness, the population is never able to recover. The cause for this is that many modules are being added directly to their respective rules in the grammar, making it difficult for GE to produce individuals which use modules and also the original terminal set. In one run of this setup, 155 modules were discovered and added to the grammar on the first module identification and grammar modification step. The Santa Fe Ant Trail grammar is given in Figure 4.8. Out of these 155, 87 were added to the `<code>` rule, 26 were added to the `<line>` rule, and 42 were added to the `<opcode>` rule. After this first grammar modification, in the best case, the genotype-to-phenotype mapping process has a $\frac{1}{13}$ chance to create new information. Any future modules discovered will only further decrease this probability. By the end of the trials, the average best fitness is still much worse than it was pre-grammar modification. Contrary to this, the best performing approaches: **κ-5 UL**, **κ-20 UL**, and **κ-5 NL** are each able to overcome the initial spike

```
<code> ::= <line>
         | <code> <line>
<line> ::= <condition>
         | <op>
<condition> ::= if(food_ahead){ <opcode> }else{ <opcode> }
<opcode> ::= <op>
           | <opcode> <op>
<op> ::= left
       | right
       | move
```

Figure 4.8: Standard Santa Fe Ant Trail grammar

in fitness and continue improving until the end of their trials.

Even 7 Parity

The next problem to examine is the Even 7 Parity. Like the Santa Fe Ant Trail, the Even 7 Parity problem (Figure 4.9) shows a significant change in fitness after the grammar has been changed. After the grammar modification is complete, none of the approaches to adding modules to the grammar are able to recover. Unlike the Santa Fe Ant Trail, the fitness of the best individuals is never as good as it was before the grammar was first modified. However, similar to the Santa Fe Ant Trail, the best performing approaches on the Even 7 Parity used module libraries while the worst performing ones did not. At the end of the runs, the distance between the best and worst performing variants were not as pronounced as was shown in the Santa Fe Ant Trail experiments. This suggests that using module libraries on the Even 7 Parity problem gives benefit over not using the libraries. But none of the approaches to modularity are able to reach their pre-grammar modification level of fitness.

$x^5 - 2x^2 + x$ Symbolic Regression

Next in line for analysis is the $x^5 - 2x^2 + x$ Symbolic Regression problem. A graph of the average best fitness of individuals attempting to solve this problem is given in Figure 4.10.

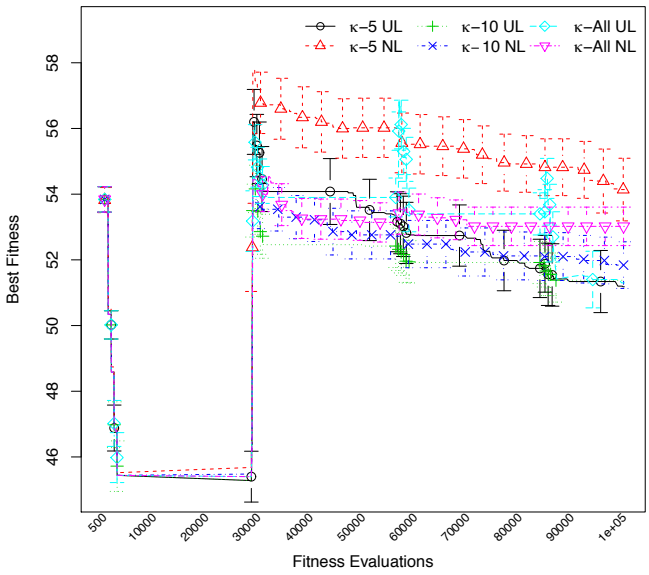


Figure 4.9: This figure shows the average best fitness and standard error for the Even 7 Parity problem.

Unlike the previous two problems, some variants of modifying GE's grammar are not so destructive that they prevent individuals from overcoming the spike in fitness that occurs when modules are first added to the grammar. In fact, the two best approaches come close to improving and do improve on the best fitness found before the grammar was modified. This suggests that using module libraries is more beneficial than not on this problem.

8×8 Lawn Mower

The final problem to examine it the 8×8 Lawn Mower problem. Like the Symbolic Regression and Even Parity problems, the two best performing approaches use module libraries. Unlike the previous problems, four out of six of the approaches to modifying the grammar are able to continually make large improvements in performance after the initial grammar modification. These results indicate that even the more destructive grammar

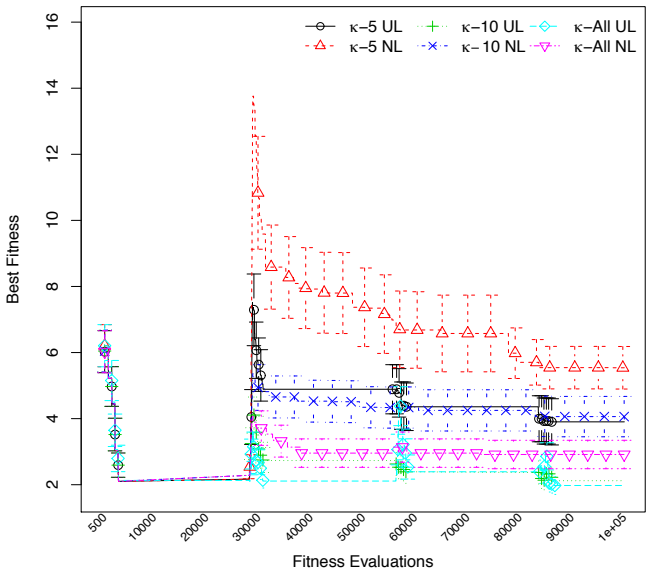


Figure 4.10: This figure shows the average best fitness and standard error for the $x^5 - 2x^2 + x$ Symbolic Regression problem.

modification methods offer some benefit to individuals attempting to solve this problem.

Across most of the problems examined, by the end of the evolutionary runs, the **NL** approaches perform worse than the **UL** ones. The most likely reason for this is that the **NL** approaches make creating new information with the grammar difficult. Consider the grammar in Figure 4.12, which was used in the Symbolic Regression problem. In its current form, the grammar allows for new information to be created easily by expanding the `<op> <expr> <expr>` production in the rule with `<expr>` as the left-hand-symbol (LHS). When expanding the `<expr>` non-terminal, there is a 50% ($\frac{1}{2}$) chance the first production will be picked and more information may be created. When the grammar is modified without module libraries numerous additional productions may be added to the rule with `<expr>` as its LHS. If only 5 modules are added to that rule, the probability of creating new information drops to approximately 14% ($\frac{1}{7}$). In the current experimental setup, either 5

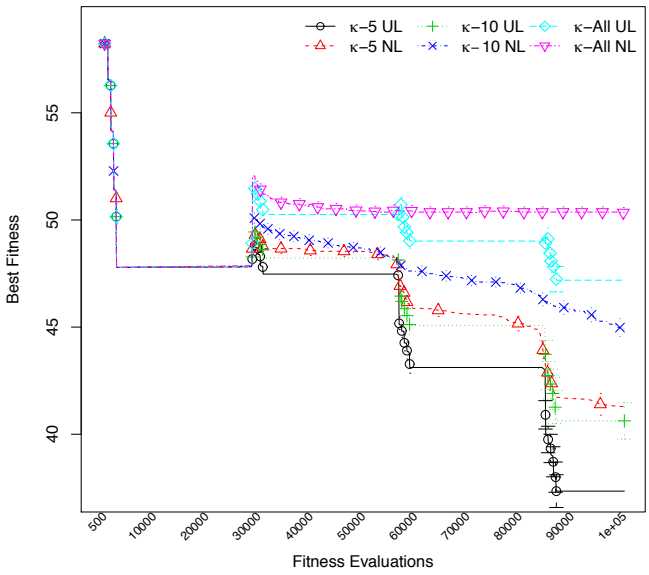


Figure 4.11: This figure shows the average best fitness and standard error for the 8×8 Lawn Mower problem.

or up to 500 modules may be added to that single production. With the addition of larger numbers of productions to the rule, the odds of actually creating new information drops drastically and the genetic operators are more likely to reuse different combinations of the modules identified. However, when module libraries are used, only one production is added to the original rule. Once again considering the rule with `<expr>` as its LHS, instead of repeatedly adding a production for each module to that rule, only one is added and a new rule is created to store the modules. With this approach, the probability of still being able to create new information never decreases below $33.\bar{3}\%$. This explains the particularly poor performance of the variation which does not use module libraries and keeps every module identified. There may be so many modules added to the grammar, the probability of not using a module is very small. These results suggest that the **NL** approach has the potential to do more harm than good on the problems examined.

```
<expr> ::= <op> <expr> <expr>
         | <var>
  <op> ::= +
         | -
         | *
         | /
 <var> ::= X
         | 1
```

Figure 4.12: Grammar for the $x^5 - 2x^3 + x$ symbolic regression problem.

Additional features of the graphs in Figures 4.7 - 4.11 that deserve attention are the large gaps between data points from approximately 2500 fitness evaluations to 30000 fitness evaluations. This gap represents fitness evaluations that are used for module identification instead of evolution of the main population. This feature will be called a **MIG**, or module identification gap. Examples of large MIGs are given in Figure 4.13. After the initial gap, some approaches have longer, shorter, or non-existent MIGs. Good examples of different sized MIGs can be seen in Figure 4.11. In this figure, the ***κ*-5 UL** method has large MIGs meaning evolution spends many fitness evaluations identifying modules. Opposite to this, ***κ*-All NL** has very small gaps between data points after the first MIG, meaning that very few or no fitness evaluations are being spent identifying modules.

The final notable feature of Figures 4.7 - 4.11 is the change in the fitness of the population after the first time modules are identified, when compared to the change in the population's fitness after each of the subsequent module identification steps. When modules are first identified, the addition of the new modules into GE's grammar causes new rules and productions to be added to the grammar. This initial change to the grammar is the largest and causes the most individuals to change. During each of the following module identification and grammar modification steps, the change to the grammar is less severe. Fewer modules may leave and/or enter the grammar, meaning that when individuals are remapped, they are more likely to remain unchanged.

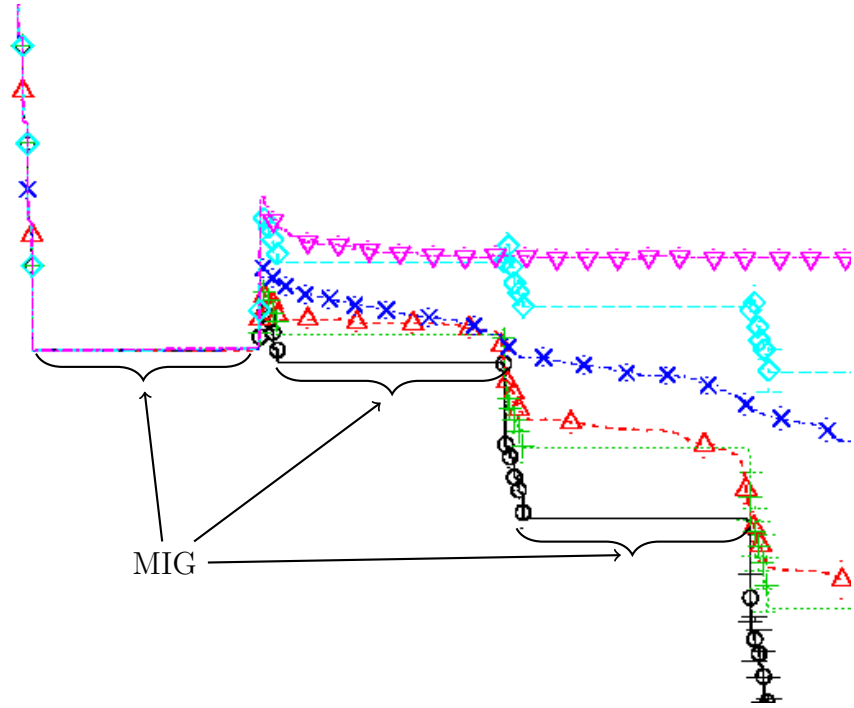
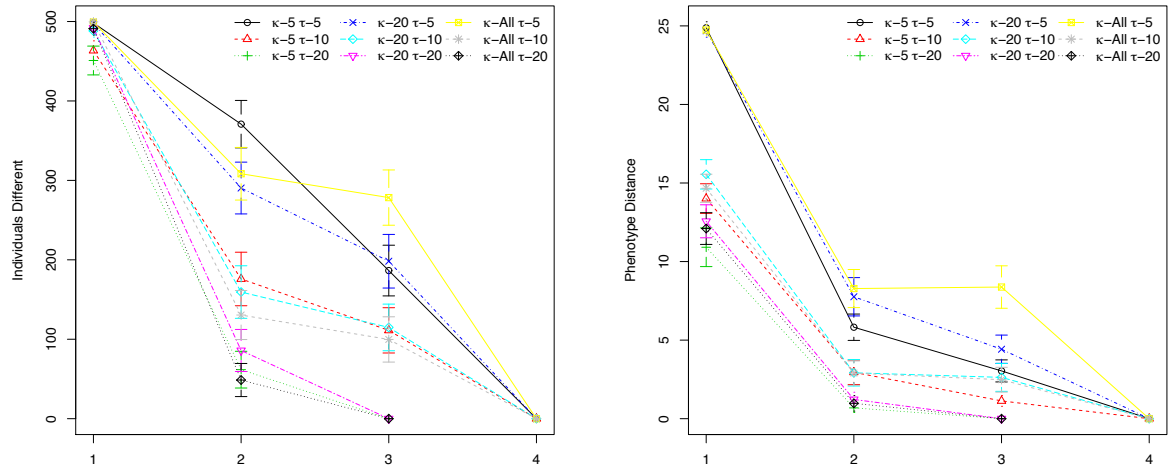


Figure 4.13: This figure shows examples of large MIGs, where many fitness evaluations are required to identify modules

4.4.2 Identification Frequency

The next item for discussion is the frequency with which the grammar is modified. After the initial grammar modification, a number of observations can be made. The first is the consistent way in which the fitness of the population is affected between the different setups when the grammar is modified at different intervals. The best fitness of runs where the grammar is modified every 5 or 10 generations is usually worse than runs where the grammar is only modified every 20 generations. One reason approaches with longer gaps between modifying the grammar tend to perform better rests in the fact that modifying the grammar and remapping the individuals to use the new grammar has the tendency to be very destructive to the population's performance. Giving the population more time before the first grammar modification allows for the development of individuals with more fit sub-derivation trees which would make more useful modules. This can be thought of



(a) Number of individuals changed

(b) Average phenotype distance

Figure 4.14: This figure shows how many individuals change (4.14(a)) and, on average, how much they change (4.14(b)) when the grammar is modified. Figure 4.14(b) measures phenotype change using the Levenshtein distance between an individual's phenotype before and after the grammar has changed and the individual has been remapped. The data in this graph comes from the Santa Fe Ant Trail, but the data is similar for all problems.

as the population going through a "shock" when the grammar is modified. This shock generally has a negative impact on the fitness of all the individuals, and the population needs time to recover in order for the best fitness to continually improve. Figure 4.14(a) shows how many individuals were different after the each of the grammar modification and remapping occurrences. After the first grammar modification, between 450 and 500 (out of a population size of 500) individuals had different phenotypes than before. Changing so many of the individuals in this way is similar to starting evolution from the beginning with an altered function set. The data in Figure 4.14(a) shows that approaches allowing more time between subsequent grammar modifications causes fewer individuals to be have different phenotypes after they have been remapped. This also reflects similarly in Figure 4.14(b), where modifying the grammar more often causes a larger average change in the phenotypes of individuals when the grammar is changed and they are remapped.

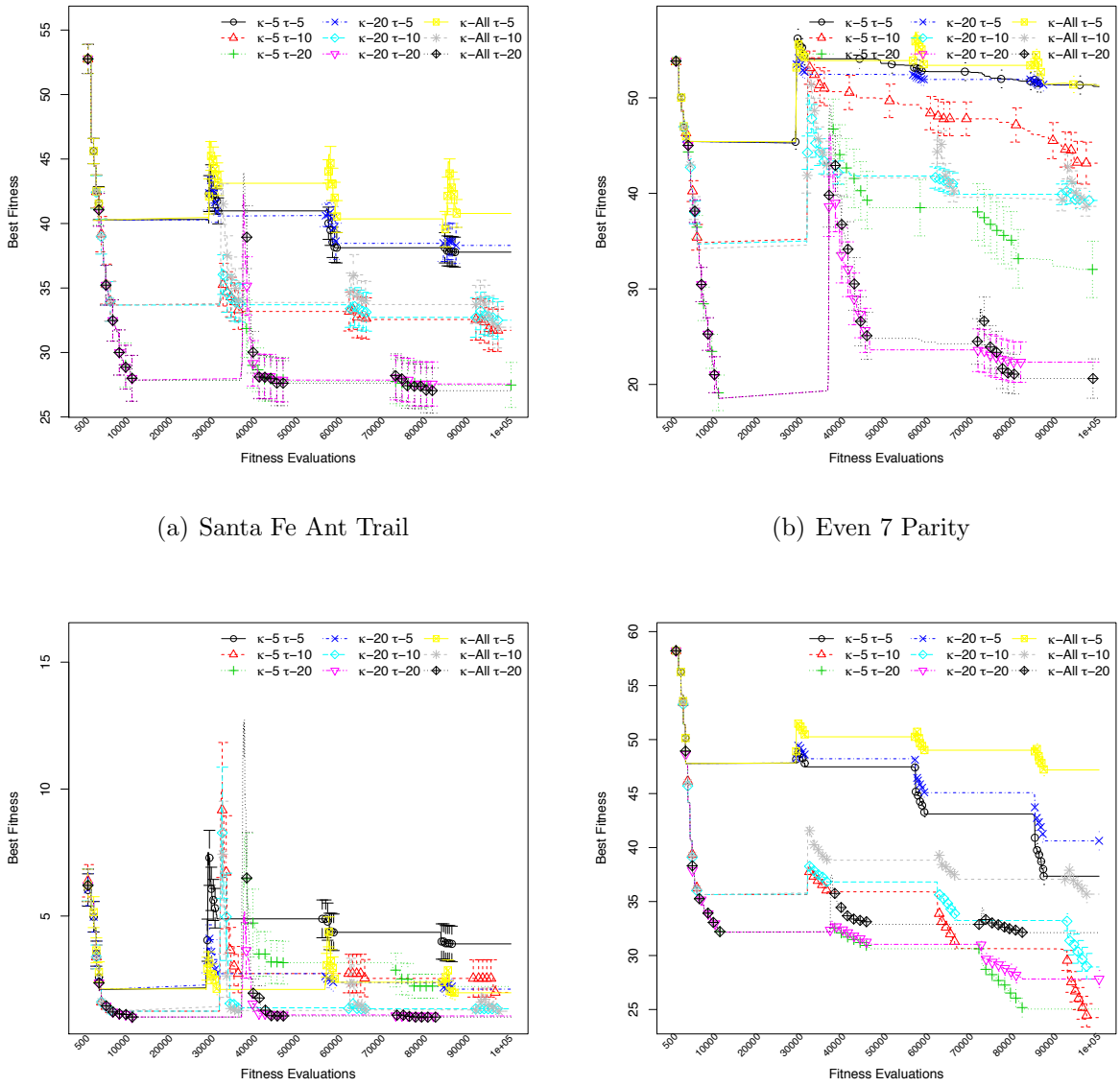
For each of the problems in Figure 4.15, the worst performing approaches modify GE's grammar every 5 generations, where the best performing approaches only modify the grammar every 10 or 20 generations. While there is no single best parameter for how often the grammar should be modified, the data presented here indicates that two things should be taken into consideration:

1. There should be a sufficient number of generations between grammar modification occurrences so evolution is able to recover from the previous change in representation;

2. Evolution should also be allowed enough generations after the population has recovered to find uses for the new features of the grammar in order to make more fit individuals.

4.4.3 Comparison with Standard GE

After identifying the most promising approaches to enhancing GE's grammar with modules, a comparison is made to determine how they fare against standard GE. The results of this comparison can be seen in Figure 4.16. For these experiments each of the approaches to modularity identifies modules and adds them to GE's grammar every 20 generations ($\tau = 20$). When the grammar is modified, module libraries are used. As is evident in all the problems except the 8×8 Lawn Mower problem, approaches to modifying the grammar are not able to achieve the same performance as standard GE. When examining the performance difference between approaches which modify the grammar and those which do not, there are some very noticeable features. The first to point out is that modifying the grammar may cause a drastic loss in performance, but on some problems, evolution is able to recover from the shock of the modified representation. The best example of this is $x^5 - 2x^3 + x$ Symbolic Regression problem in Figure 4.16(d). At generation 20 modules are identified and added to the grammar. When the population is remapped with the new



(a) Santa Fe Ant Trail

(b) Even 7 Parity

(c) $x^5 - 2x^3 + x$ Symbolic Regression

(d) 8×8 Lawn Mower

Figure 4.15: These graphs show the average best fitness and standard error for the Santa Fe Ant Trail (Figure 4.15(a)), Even 7 Parity (Figure 4.15(b)), 8×8 Lawn Mower (Figure 4.15(d)), and $x^5 - 2x^3 + x$ Symbolic Regression (Figure 4.15(c)). The ***τ*-5**, ***τ*-10**, and ***τ*-20** in the legends indicate steps of 5, 10, and 20 generations between module identification and grammar modification.

Table 4.3: This table shows the resulting p-value from performing a Wilcoxon rank-sum test on the best fitness values of GE and GE with varying numbers of modules added to its grammar after 100000 fitness evaluations. A P-value < 0.05 is considered significant. An asterisk (*) denotes an approach that significantly outperforms standard GE. These results were calculated using the Wilcoxon rank-sum test provided by the R programming language [128].

	Santa Fe Ant Trail		
	κ-5	κ-20	κ-All
GE	9.37×10^{-5}	1.56×10^{-5}	7.66×10^{-4}
	Even 7 Parity		
	κ-5	κ-20	κ-All
GE	7.45×10^{-9}	3.09×10^{-7}	2.14×10^{-7}
	$x^5 - 2x^3 + x$ Symbolic Regression		
	κ-5	κ-20	κ-All
GE	6.20×10^{-9}	9.73×10^{-9}	3.56×10^{-9}
	8×8 Lawn Mower		
	κ-5	κ-20	κ-All
GE	4.29×10^{-4}*	3.50×10^{-1}	1.06×10^{-6}

grammar, there is a drastic change in fitness; it gets much worse. However, in under 10 generations, fitness is able to recover and continue making similar progress to a standard GE run. The same happens again in generation 40. This suggests that evolution is able to overcome the disruption caused by remapping the population with a new grammar. Despite evolution making somewhat of a recovery after the loss of fitness caused by the changing grammar, standard GE outperforms each of the approaches for adding modules to GE's grammar. Table 4.3 shows the results of performing Wilcoxon rank-sum tests on GE and the approaches to modularity in Figure 4.16. In the case of the Even 7 Parity problem (Figure 4.16(b)), this is not so much the case. There are large disturbances in fitness when the grammar is modified, and evolution is able to somewhat recover, but it never recovers enough to improve on or even match GE's performance. Another interesting problem is the Santa Fe Ant Trail in Figure 4.16(a). When the grammar is modified in generation 20, fitness only improves a miniscule amount for the rest of the run. This suggests that the change in the population's grammar causes search to become stuck in a local optima for

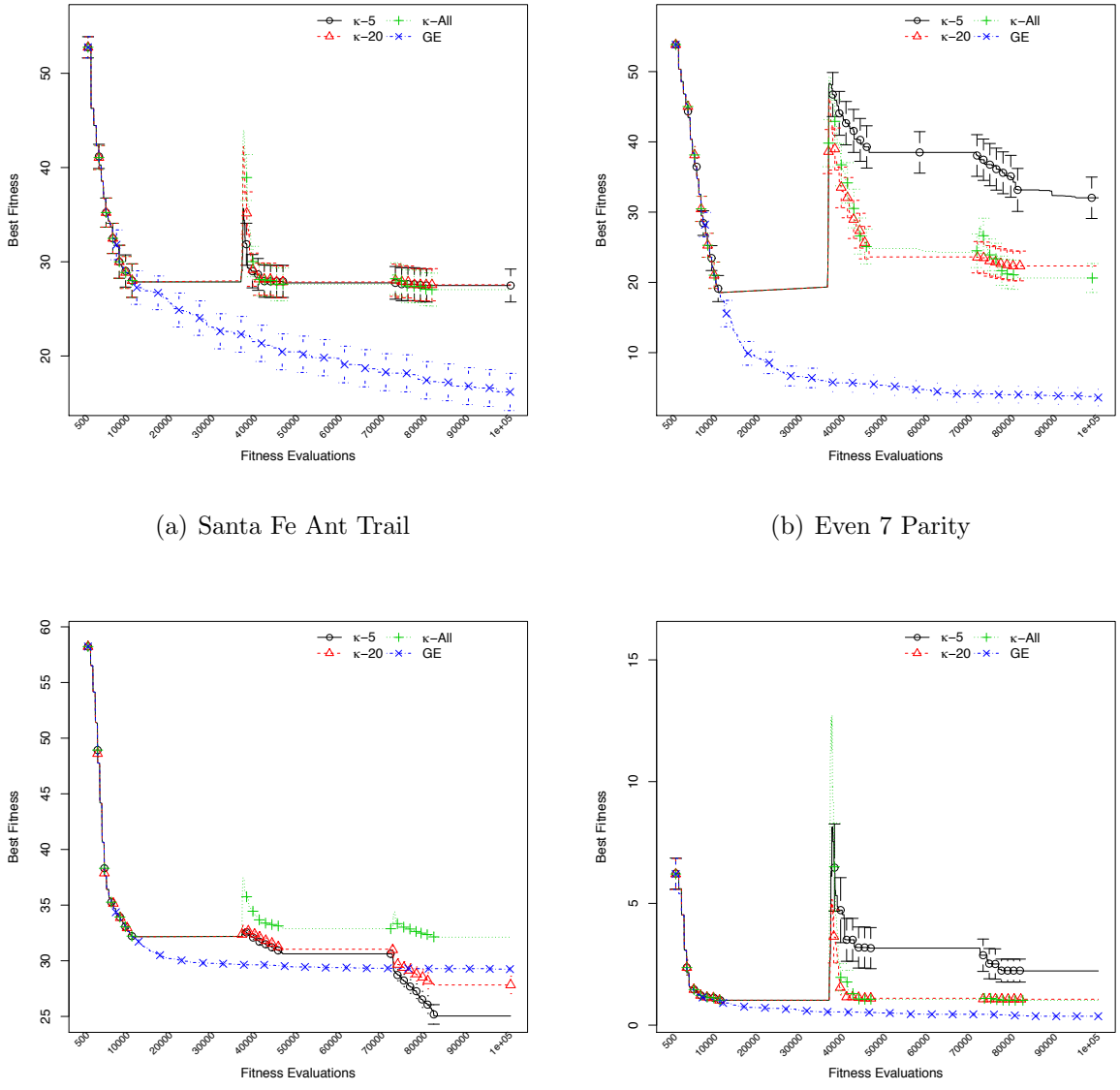
relatively long periods during evolution. The final and most encouraging results comes from Figure 4.16(c), the 8×8 Lawn Mower problem. This is the sole problem that was able to perform even slightly better than standard GE and used a shorter interval between module identification and grammar modification (5 generations). While standard GE made small progress from approximately generation 30 on, using modules to encapsulate information and enhancing the grammar with those modules, allowed evolution make more progress towards solving the problem.

4.5 Summary

This chapter introduces a basic method of identifying modules and examines a variety of parameters involved in incorporating them back into an evolving population. Using these methods some benefits and drawbacks can be seen. The first lesson to be taken from this chapter is that simply identifying good information and using that to modify GE's grammar mid-evolutionary run will not necessarily benefit search. In most cases, it hurts evolutionary progress. However, for one of the benchmark problems being examined, modifying the grammar to make good information more easily available actually enhanced performance. Is is also clear from the results shown here that for any significant progress to be made, a non-destructive method of incorporating modules into the grammar must be identified. Three of the questions proposed in Section 1.2.1 are answered in this chapter:

Research Question 5 - How should modules be made available to GE's population during evolution? When adding modules directly to GE's grammar, module library productions allow modules to be used by the population without lessening the possibility of individuals using productions from the original grammar.

Research Question 6 - Does updating the set of modules available to individuals more or less frequently change the performance of GE? On each of the benchmark problems



(a) Santa Fe Ant Trail

(b) Even 7 Parity

(c) 8×8 Lawn Mower

(d) $x^5 - 2x^3 + x$ Symbolic Regression

Figure 4.16: These graphs show the average best fitness and standard error across 50 trials of the Santa Fe Ant Trail (Figure 4.16(a)), Even 7 Parity (Figure 4.16(b)), 8×8 Lawn Mower (Figure 4.16(c)), and $x^5 - 2x^3 + x$ Symbolic Regression (Figure 4.16(d)). The **GE** in the legends represent standard GE with no enhancements.

tested in this chapter, at least two of the best three approaches used the largest time interval ($\tau = 20$) for the number of generations between module identification. This suggests that longer time periods between module identification and grammar modification allow evolution more time to make use of the new grammar.

Research Question 7 - Does the number of modules made available to GE during evolution affect GE's performance? On three of the four benchmark problems, the middle value ($\kappa = 20$) of how many modules should be made available to GE's grammar at a time gave the best average fitness values. These results suggest a balance is needed in terms of allowing too many or too few modules into the grammar at once.

The next chapter describes a remedy for the destructive nature of modifying GE's grammar without accounting for the changes that may occur in the individuals' phenotypes, a genotype repair operation. It then compares the performance of the genotype repair against the approach taken here and against standard GE. This chapter also raises questions about how appropriate the approach to identifying modules described here is. Additional means for identifying modules are discussed in Chapter 6.

Chapter 5

Genotype Repair

After the introduction to module identification and grammar modification in Chapter 4, it is clear that there are certain parameters for how often and how many modules are added to the grammar which can be detrimental to GE's performance. For example, modifying GE's grammar too frequently or with too many modules can cause a large loss in fitness. Chapter 4 also shows that the method in which modules are added to the grammar (using and not using module libraries) is also important to consider. After the first grammar modification, a loss in fitness occurred on each of the benchmark problems. On three of the four benchmark problems tested, evolution either never or only slightly improves on the fitness it reached before the initial grammar modification. This chapter describes two extensions to remedy the detrimental behavior shown in the Chapter 4: genotype repair and module expansion mechanisms. It then shows that after the same number of fitness evaluations, GE's fitness is able to improve over that of the **remap** method when using the approaches described here. The experiments presented in this chapter are based on, and extend, those published by Swafford et al.[156].

The remainder of this chapter introduces and examines the performance of two new operations, which function as extensions of the grammar modification approaches described

in Chapter 4. In the following sections of this chapter, a **remap** approach is mentioned. This approach refers to modifying GE's grammar and remapping each individual using the new grammar. Evidence from the previous experimentation in Chapter 4 has shown that simply adding modules to GE's grammar during an evolutionary run using the **remap** method damages the population's fitness. This is due to GE's genotype-to-phenotype mapping process. When any rule in the grammar has productions added or subtracted from it, the probability of that rule retaining its previous mapping is low (Section 4.4 gives an explanation of how changing GE's grammar alters this probability). This means that after the grammar changes, many individuals will not map as they previously did. The data presented in Chapter 4 shows how the process of modifying GE's grammar and remapping individuals causes many individuals to immediately change their phenotypes. This change in the individuals' phenotypes changes the population's fitness in that almost every individual becomes significantly less fit after the change. To remedy this, a genotype repair function and a module expansion function are added to the grammar modification operation. The following sections describe these extensions and how they enable GE to continue with evolution without the loss in performance caused by using the **remap** approach. These extensions allow evolution to use modules at its own pace and show significant improvements in fitness when compared to the **remap** method.

5.1 Genotype Repair and Module Expansion

To alleviate the loss of fitness brought on by grammar modification a genotype the novel **repair** operation is introduced. The repair operation ensures each individual maintains its phenotype after the grammar has changed and the genotype-to-phenotype mapping is performed again. This is done by iterating the individual's chromosome and modifying each necessary codon such that during the mapping process it picks the production it picked

Algorithm 5.1 The genotype repair algorithm for ensuring that an individual maintains its original phenotype after GE's grammar has changed. For this algorithm to function correctly, every production and rule in the previous grammar used by the population must also be present in the updated grammar.

```
G = get_original_grammar()
G_new = add_modules_to_grammar(M, G)

for Individual I in Population do
    G_old = I.get_grammar()
    I' = copy_of(I)
    I'.set_grammar(G_new)
    Γ = I.get_chromosome()
    Γ' = I'.get_chromosome()

    i, j = 0
    for i = 0 ; i < length_of_coding_region(Γ); i++ do
        prod_old = production_picked_by(Γ, i, G_old)
        prod_new = production_picked_by(Γ', j, G_new)

        if prod_old ! = prod_new then
            Γ' = update_codon_to_pick_production(Γ', j, G_new, prod_old)
            j = update_codon_index(Γ', G_new, j, Γ, G_old, i, prod_old)
```

before the grammar was modified. Figure 5.1 shows an example of how this repair method functions, and pseudocode for the algorithm is given in Algorithm 5.1. In Algorithm 5.1, the `production_picked_by()` function determines which production in the grammar is picked by the given codon. The `update_codon_to_pick_production()` method modifies the codon at index j in the chromosome to pick the appropriate production. In some cases of grammar modification, deterministic rules, which use no codons, become non-deterministic. When this happens, any needed additional codons are inserted into the chromosome to allow the individual to finish the genotype-to-phenotype mapping correctly. The final method, `update_codon_index()`, updates the chromosome index counter to ensure that the repair process continues after any inserted codons.

However, the genotype repair operation has a problem. If individuals use a module that

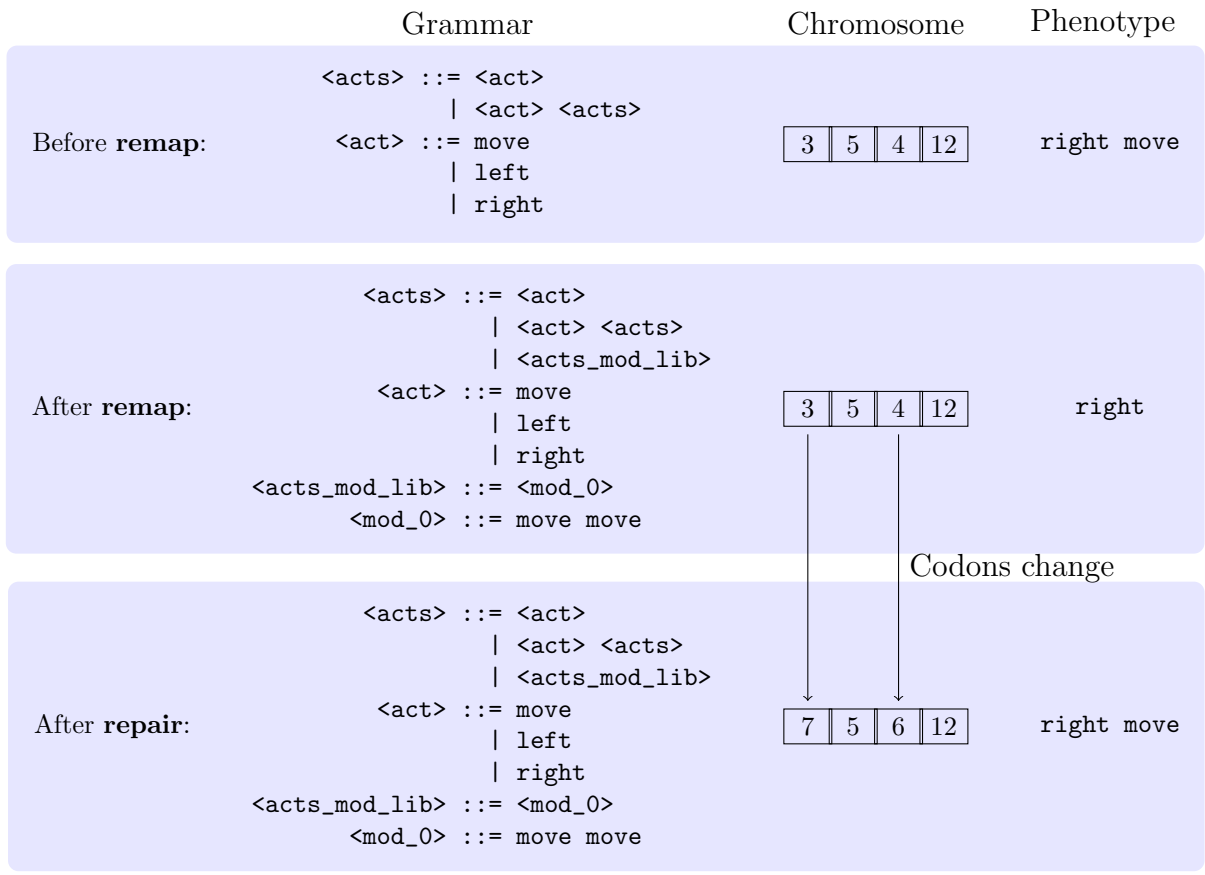


Figure 5.1: This figure shows a grammar, chromosome, and phenotype for an individual before the grammar is modified, after a module has been added to the grammar and the **remap** approach has been applied, and after the **repair** extension fixes the individual's genotype.

has been removed from the grammar, the **repair** method will be unable to ensure that those individuals will retain their original phenotypes. Two solutions for this problem are explored. The first is simply adding modules back into the grammar while individuals are still using them. This approach is easy to implement, but it also presents a new conflict. A module, or modules, has been removed from GE's grammar because more beneficial modules have been discovered and will take its place. But some individuals may still be using this "unfit" or "outdated" module. There could be any number of individuals using this module, and it may be better to remove it from the grammar. After multiple module identifications, a minority of individuals can potentially keep many modules in the grammar

that may be useless. This is the default approach when using the **repair** operation. A novel alternative to leaving outdated modules in the grammar is replacing them with the full sub-derivation tree from which they were created. This replacement is referred to as the **expand** extension. The implementation of this operation is more difficult than leaving old modules in GE's grammar, but ensures that each individual can go through the genotype repair process without a problem and modules deemed unnecessary or outdated by the module replacement criteria are still removed from the population. An example of the **expand** process is given in Figure 5.2. Both of these methods are also able to save fitness evaluations in comparison to the **remap** method as the individuals' phenotypes do not change after the grammar has been modified and do not require additional mapping and re-evaluation.

There also exist alternatives to the **expand** operator which could be implemented. One such alternative, similar to Koza's work with architecture altering operations [72], could replace codons that map to modules that have been removed from GE's grammar with new codons that map to modules which currently exist in the grammar. Another alternative, inspired by Majeed and Ryan [88] in their work identifying modules, would be to replace codons which map to deleted modules with codons that will map to the identity function of first operator which would take the deleted module as an argument. However, both of these alternatives have a side effect in common; individuals operated on by these alternative methods may still map to a different phenotype than they did before the grammar was changed, which was suggested as undesirable by the results given in Chapter 4.

5.2 Experimental Design

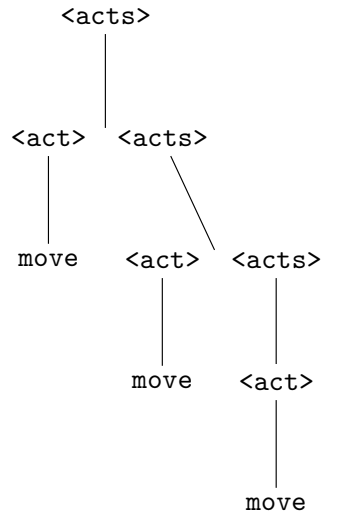
The experiments carried out in this chapter enable a better understanding how the **remap** approach from Chapter 4 performs in relation to the **repair** and **expand** methods described

```
<acts> ::= <act>
         | <act> <acts>
         | <acts_mod_lib>
 <act> ::= move
         | left
         | right
<acts_mod_lib> ::= <mod_1>
       <mod_1> ::= move move move
```

(a) Grammar with original module

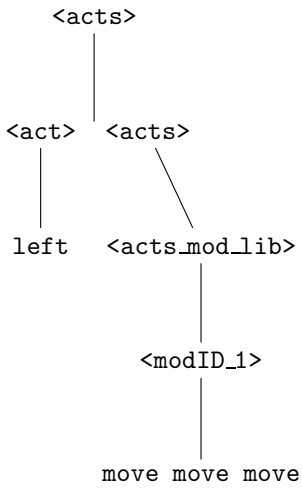
```
<acts> ::= <act>
         | <act> <acts>
         | <acts_mod_lib>
 <act> ::= move
         | left
         | right
<acts_mod_lib> ::= <mod_2>
       <mod_2> ::= left move left
```

(b) Grammar with `mod_1` replaced by `mod_2`



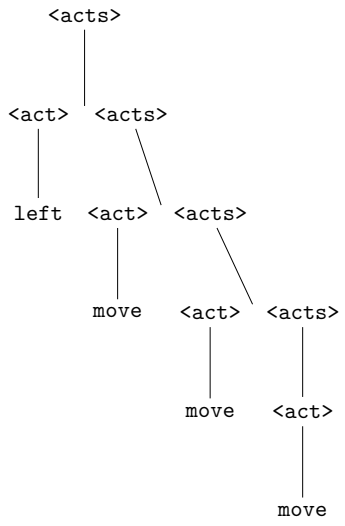
7	15	22	42	12	33

(c) An example module and its chromosome



9	31	23	17

(d) Individual using the module



9	31	25	7	15	22	42	12	33

(e) Individual with the expanded module

Figure 5.2: This figure shows how modules are expanded once they have been removed from the grammar and are still in use by individuals. Figure 5.2(a) shows the grammar for the Lawn Mower problem with one module added to it. Figure 5.2(b) shows the same grammar but with the initial module replaced with a newer one. Figure 5.2(c) shows the sub-derivation tree and chromosome that produces the module in Figure 5.2(a). An individual created using the grammar in Figure 5.2(a) that uses the module is given in Figure 5.2(d). Figure 5.2(e) shows the same individual from Figure 5.2(d), after its grammar has changed (Figure 5.2(b)) and it has gone through the **expand** process.

Table 5.1: Experimental setup for all evolutionary runs. These are the same parameters used in Chapter 4. The parameters for κ and τ are used due to the experimental results from Chapter 4.

Parameter	Value
Independent Runs	50
Fitness Evaluations	100000
Population	500
Selection	Tournament (Size 5)
Wrapping	None
Crossover	Single Point (90%)
Mutation	Int Flip (1%)
Elites	50
Initialization	Ramped Half and Half
Replacement	Generational
Max. Derivation Tree Size	25 (Lawn Mower - 100)
κ	20
ρ	75% of n
n	50
τ	20

above. Table 5.1 shows the experimental parameters used in these experiments. The results from Chapter 4, indicate that using module libraries is the most suitable method for adding modules to the grammar. It also suggests that the values of τ (module identification, replacement, and grammar modification interval) and κ (number of modules to keep after replacement) in Table 5.1 are also reasonable. As in Chapter 4, the Even 7 Parity, Santa Fe Ant Trail, 8×8 Lawn Mower, and $x^5 - 2x^3 + x$ Symbolic Regression problems are used.

5.3 Results and Discussion

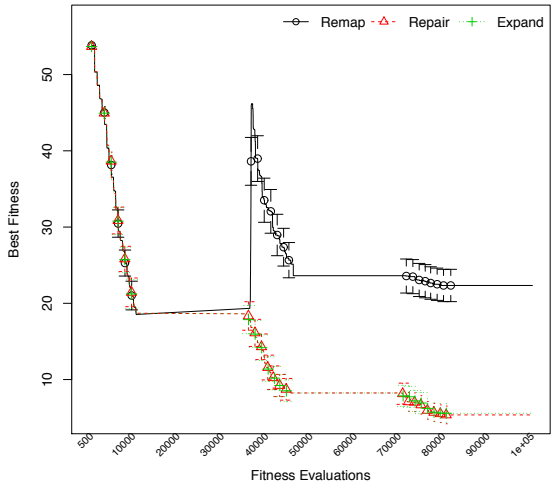
This section presents and analyses the performance of the **repair** and **expand** approaches for modifying GE's grammar in comparison to the previously examined **remap** method. The first comparisons are given in Section 5.3.1 and concern the effects each method has on the fitness of the evolving population. Next, Section 5.3.2 examines how many of the discovered modules are used by the **remap** , **repair** , and **expand** methods. Finally, in

Section 5.3.3 the performance of all three approaches for handling modules added to GE's grammar are compared to standard GE.

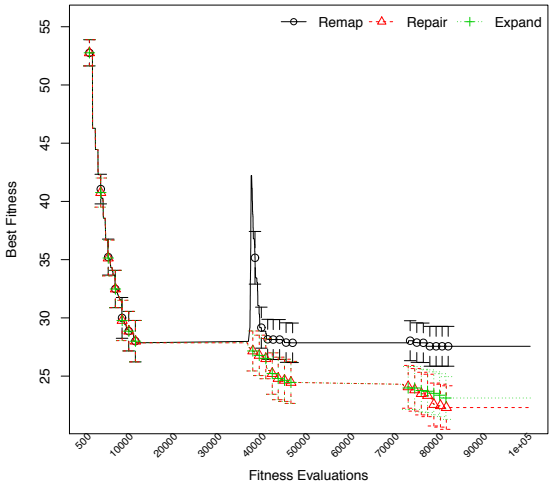
5.3.1 Genotype Repair and Module Expansion Results

The first comparison to make is the performance differences between the **remap**, **repair**, and **expand** approaches. The average best fitness for these methods on the four benchmark problems is given in Figure 5.3. The most notable difference between these approaches is the lack of a fitness spike in the **repair** and **expand** lines. For each of the benchmark problems in Figure 5.3, the **repair** and **expand** methods improve fitness after modules have been discovered. It should also be noted that each of the approaches used the same 50 random number generator seeds for their trials. This means that the same modules are being discovered by each method at the first module identification step and the deviation in the shapes of their fitness plots is caused by how the modules are being used by evolution in the proceeding generations. Even when the **remap** data shows improvement when modules are discovered, the **repair** and **expand** approaches show greater improvement.

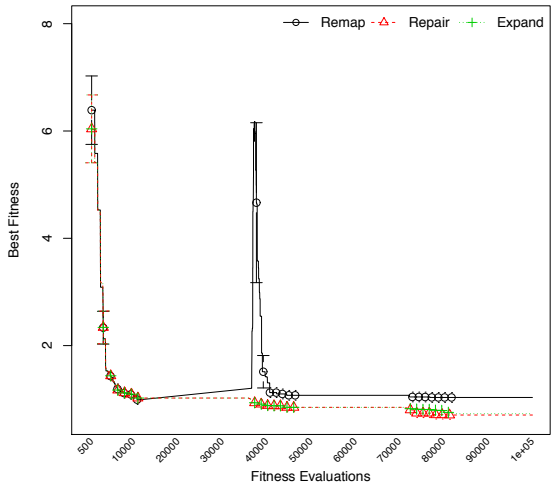
The data showing the **repair** and **expand** outperforming the **remap** approach on the Even 7 Parity (Figure 5.3(a)) and 8×8 Lawn Mower (Figure 5.3(d)) problems is fairly conclusive about the utility of the extensions introduced earlier in this chapter. However, the data for the $x^5 - 2x^3 + x$ Symbolic Regression (Figure 5.3(c)) and Santa Fe Ant Trail (Figure 5.3(b)) are not as convincing. For this reason, Wilcoxon rank-sum tests are performed on the best fitness values at the end of each run for each combination of the **remap**, **repair**, and **expand** variants. The results of these tests are given in Table 5.2. These comparisons show that the **repair** and **expand** extensions significantly outperform (p-value < 0.05) the **remap** approach on each of the benchmark problems examined. The data in Table 5.2 also shows that there is no significant difference in performance between the **repair** and **expand** approaches on any of the problems except the 8×8 Lawn



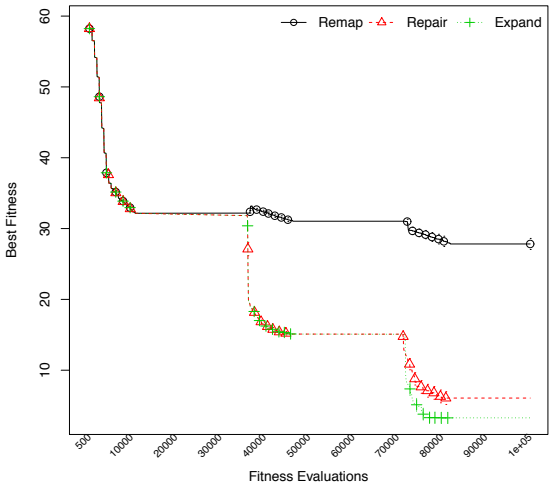
(a) Even 7 Parity



(b) Santa Fe Ant Trail



(c) $x^5 - 2x^3 + x$ Symbolic Regression



(d) 8×8 Lawn Mower

Figure 5.3: This figure shows the average best fitness of the **remap**, **repair**, and **expand** approaches to grammar modification on the Even 7 Parity (Figure 5.3(a)), Santa Fe Ant Trail (Figure 5.3(b)), $x^5 - 2x^3 + x$ Symbolic Regression (Figure 5.3(c)), and 8×8 Lawn Mower (Figure 5.3(d)) problems.

Table 5.2: This table reports the p-value of Wilcoxon rank-sum tests performed on the average best fitness values of each approach after 100000 fitness evaluations. The p-values reported are calculated with a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

		Remap	Repair	Expand
Santa Fe Ant Trail	Remap		0.013*	0.036*
	Repair	0.013*		0.780
Even 7 Parity	Remap		5.808×10^{-7}*	1.708×10^{-6}*
	Repair	5.808×10^{-7}*		0.991
$x^5 - 2x^3 + x$ Symbolic Regression	Remap		0.001*	0.005*
	Repair	0.001*		0.706
8×8 Lawn Mower	Remap		7.8×10^{-10}*	7.8×10^{-10}*
	Repair	7.8×10^{-10}*		0.047*

Mower problem, where the **expand** extension outperforms the other approaches. The only difference between the **repair** and **expand** methods that can account for this deviation in performance is a direct side-effect of the **expand** operation. After individuals have been expanded, they may have much larger derivation trees, making it possible for evolution to create bigger and/or better modules in future module identification steps.

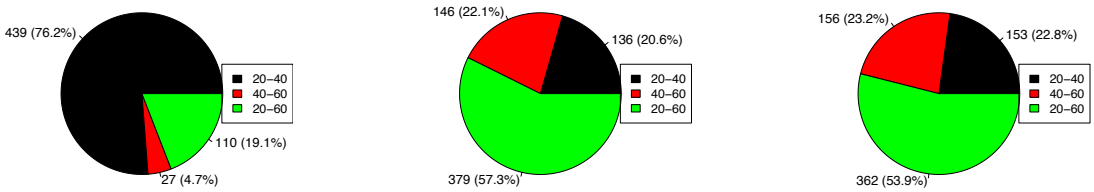
5.3.2 Module Usage

If modules are being identified and not used, the search space is increased with no performance gain, and the computational effort used to discover modules is wasted. This section describes how the various methods for making modules available to the population and handling their removal from GE's grammar impact how modules are used. For the following analysis, only modules that appear in 1% or more of individuals in the population (with a population size of 500 individuals) are considered.

On the Santa Fe Ant Trail problem, the total number of modules used by the **remap** method across all 50 runs is 576. Out of these 576 modules, 439 are used only in the 20 generations after they are identified. Pie charts showing these values can be found in Figure 5.4. The likely reason for this stems from the individuals being remapped as soon

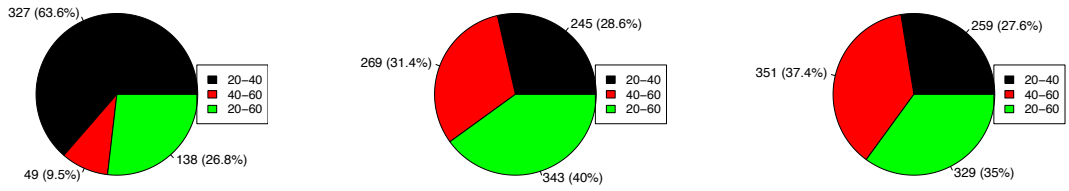
as the grammar has been modified. This means that many modules will get used as soon as they are added to GE's grammar solely for this reason as opposed to evolution using them at its own pace. In the case of the Santa Fe Ant Trail, 76% of the modules used are used because of this. Following that, 19% of modules are identified during the first module identification step and survive module replacement to be present in the population after the second module identification step. Additionally, only 5% of the modules used are discovered in the second module identification step. Contrary to this, the **repair** and **expand** methods divide their module usage more evenly. The **repair** approach only uses 21% of the modules after the first and before the second module identification occurrence, with 57% getting used after the first and second module identification and replacement steps. The remaining 22% of modules are only present after the second module identification step. Module usage percentages are similar for the **expand** method. Because the same modules are discovered at the first module identification step for each of these approaches, the differences in how the modules get used come from how the modules are integrated into the population. By allowing mutation and crossover operations to introduce modules into the population, the population is not shocked all at once by remapping each individual to use the new grammar. The $x^5 - 2x^2 + x$ Symbolic Regression problem also shows these same trends. The **remap** approach used the least amount of modules and used 64% of those modules exclusively in the 20 generations after they were identified. Only 9% of the modules used were discovered in the second module identification step. Figure 5.5 gives this data in a pie chart.

The **repair** and **expand** methods identify more modules (857 and 939 compared to **remap's** 514), and their usage is distributed more evenly through the generations. The 8×8 Lawn Mower and Even 7 Parity problem show slightly different behaviors. For both of these problems, **remap** uses far fewer modules (440 on Even 7 Parity and 561 on 8×8 Lawn Mower) than the **repair** and **expand** approaches which use between 1370 and 1700



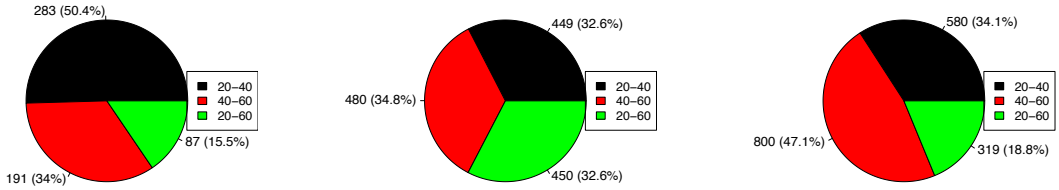
(a) Remap (b) Repair (c) Expand

Figure 5.4: This figure shows the module usage for the Santa Fe Ant Trail problem. The black sections of the pie charts in this figure represent the amount of modules that are only present in generations 20 - 40. The red sections denote how many modules are only present in generations 40 - 60. The green sections represent how many modules are present from generation 20 to generation 60.



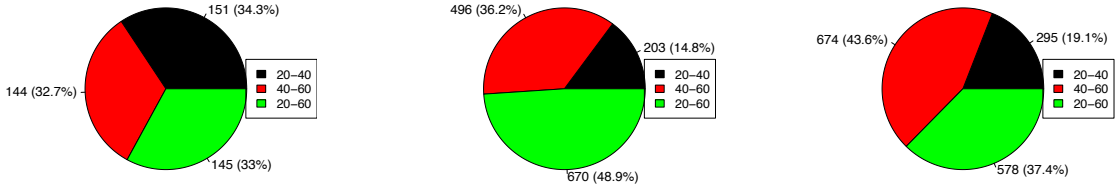
(a) Remap (b) Repair (c) Expand

Figure 5.5: This figure shows the module usage for the $x^5 - 2x^2 + x$ Symbolic Regression problem.



(a) Remap (b) Repair (c) Expand

Figure 5.6: This figure shows the module usage for the 8×8 Lawn Mower problem.



(a) Remap (b) Repair (c) Expand

Figure 5.7: This figure shows the module usage for the Even 7 Parity problem.

modules for each problem. The data here suggests that a large contributing factor in the deficit of performance of the **remap** approach comes from its inability to allow evolution to use modules. The data for these problems can be found in Figures 5.6 and 5.7.

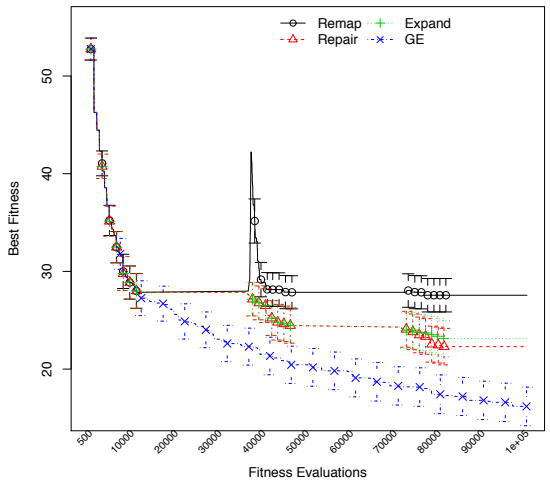
5.3.3 Comparison with Standard GE

The final comparisons of this chapter show how the **remap**, **repair**, and **expand** approaches compare to standard GE. The average best fitness of each of these methods over time can be seen in Figure 5.8. This figure shows that even though the **repair** and **expand** approaches are able to improve over the unfavorable performance of the **remap** approach, they are not always able to beat standard GE's fitness levels. On both the Santa Fe Ant Trail (Figure 5.8(a)) and the $x^5 - 2x^3 + x$ Symbolic Regression (Figure 5.8(c)) problems,

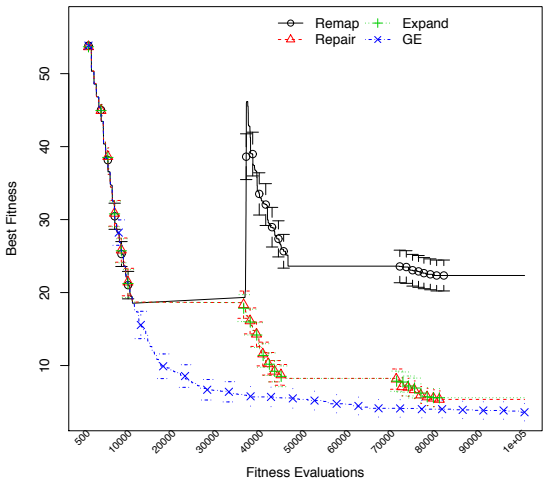
standard GE outperforms all the approaches to modularity examined. This is rather unexpected as various approaches to modularity have been shown to improve performance over these problems in standard GP[74, 165]. However, evidence of this undesirable performance is not present for each benchmark problem. For the Even 7 Parity problem (Figure 5.8(b)), the **repair** and **expand** approaches have slightly worse average best fitness values, but are not significantly worse than standard GE. Finally, on the 8×8 Lawn Mower problem, both the **repair** and **expand** methods significantly outperform standard GE and the **remap** approach. The most likely reason for **repair** and **expand** methods' good performance on the Lawn Mower problem is the nature of the problem itself. As modules are able to encapsulate multiple mowing instructions into a single production, individuals using modules are able to cover more area of the lawn and use fewer codons in doing so. Taking this under consideration, it is easy to see why these extensions of the **remap** perform so well. While these results do not show any significant improvement over standard GE on three of the benchmark problems, there are many factors that may account for this which are out of the scope of this chapter. Some of these issues include, but are not limited to, deficiencies in how modules are evaluated, how module parents are selected, and the size of modules. These are examined in the following chapters of this thesis.

5.4 Summary

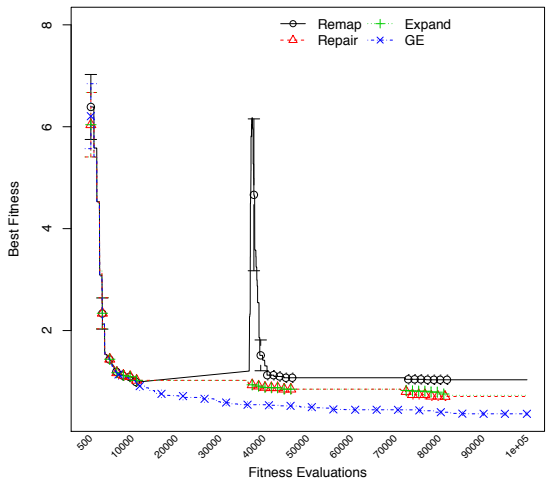
This chapter introduces two extensions for the **remap** approach to adding modules to GE's grammar: **repair** and **expand** . Both of these extensions show significant improvements in fitness over the **remap** approach for all of the benchmark problems examined. Despite the boost in performance over the **remap** method for incorporating modules, both the **repair** and **expand** are unable to outperform standard GE on three out of the four benchmark problems examined. However, the purpose of this and the previous chapter is to identify



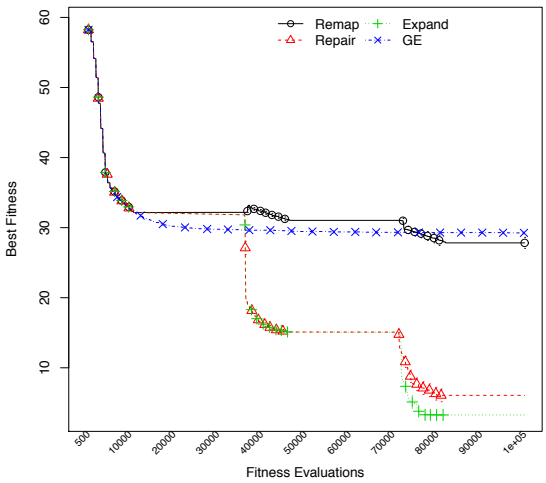
(a) Santa Fe Ant Trail



(b) Even 7 Parity



(c) $x^5 - 2x^3 + x$ Symbolic Regression



(d) 8×8 Lawn Mower

Figure 5.8: This figure shows the average best fitness of the **remap**, **repair**, and **expand**, approaches to incorporating modules into GE's grammar as well as standard GE.

Table 5.3: This table give the average best fitness and standard error for each of the benchmark problems examined in this chapter. The p-value reported is calculated by Wilcoxon rank-sum tests performed on the average best fitness values of each approach after 100000 fitness evaluations. The p-values are calculated with a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

Approach	Average Best Fitness	Standard Error	p-value (Compared to GE)
Santa Fe Ant Trail			
Remap	27.560	± 1.711	1.751×10^{-4}*
Repair	22.300	± 1.870	0.033*
Expand	23.120	± 1.832	0.028*
GE	16.180	± 1.976	NA
Even 7 Parity			
Remap	22.340	± 2.118	2.002×10^{-7}*
Repair	5.360	± 1.160	0.309
Expand	5.580	± 1.245	0.366
GE	3.600	± 1.185	NA
8×8 Lawn Mower			
Remap	27.826	± 0.781	0.437
Repair	6.068	± 0.905	7.8×10^{-10}*
Expand	3.270	± 0.790	7.7×10^{-10}*
GE	29.250	± 0.163	NA
$x^5 - 2x^3 + x$ Symbolic Regression			
Remap	1.033	± 0.092	$2.034 \times 10^{-}8$*
Repair	0.706	± 0.079	$9.077 \times 10^{-}4$*
Expand	0.730	± 0.078	$3.040 \times 10^{-}5$*
GE	0.364	± 0.034	NA

methods for making modules available to GE's evolving population. A likely possibility for the less than desirable performance is that the modules being discovered are not "good" modules. The following chapters will examine different mechanisms for identifying modules in an attempt to remedy this problem.

Part III

Module Identification: Methods and Analysis

Chapter 6

Examining Methods for Module Identification

The previous two chapters of this thesis (Chapters 4 and 5) explore adding modules to GE's grammar as a mechanism for incorporating them into the evolving population. This is undertaken because of the importance of making the information encapsulated by modules available to the population without disrupting the progress it has made thus far. Specifically, the following three methods for incorporating modules into GE's grammar are tested:

1. The **remap** approach adds modules to GE's grammar and remaps each individual in the population to use the updated grammar.

2. The **repair** approach extends **remap** and ensures that, no matter how the grammar changes, all individuals maintain their previous phenotypes by altering their genotypes. When modules are removed from the grammar but individuals are still using them, **repair** adds these modules back to the grammar until they are no longer in use.

3. The **expand** approach extends **repair** by replacing occurrences of modules that have been removed from the grammar with the entire sub-derivation tree the module was created from.

Chapters 4 and 5 demonstrate that the ability to make modules available to the population without harming the progress made by the population thus far is crucial but is only one of the two issues surrounding the exploitation of modularity in GE. The remaining issue is identifying the modules themselves. This chapter compares four methods for identifying modules and a variety of parameters for these methods. This chapter also answers questions 1.2.1 and 1.2.1 from Section 1.2.1. The work presented in this chapter is based on, and extends, that of Swafford et al.[157].

6.1 Module Identification Methods

In Chapter 4, one novel method for identifying modules is introduced: mutation module identification (M-ID), however, it would be naïve to assume that one method for module identification is suitable for all problems. For example, using the M-ID method may not perform as well on the Santa Fe Ant Trail problem when compared to an alternative module identification strategy. Simply picking random sub-derivation trees or frequently used sub-derivation trees may be more appropriate. With this in mind, one additional novel module identification method, (Insertion Identification) and two simple methods (Random and Frequency Identification) have been implemented to discover modules that may be beneficial to the population in different ways:

Mutation Identification (M-ID): For each individual, I, in the population, a node on its derivation tree is randomly picked. The sub-derivation tree starting with this node is the candidate module, λ. The fitness of I, I_{f} is recorded. Next, the depth of λ is recorded as λ_{d}. Now, a copy of I is created, I', and a randomly generated

sub-derivation tree with a depth equal to λ_{d} replaces λ in I'. The fitness of I' after the replacement is calculated and the difference between I_{f} and I'_{f} is recorded. This process of creating random sub-derivation trees, replacing λ, and evaluating the newly created I' is repeated n times. To determine if λ should be kept as a module, the number of recorded fitness differences that are greater than 0 is counted. If this value is greater than ρ and there are no pre-existing modules with the same sub-derivation tree as λ, the mean of all the saved fitness differences is assigned as the module's fitness, λ_{f}, and the module is saved in a module list. Pseudocode for this algorithm is given in Algorithm 6.1 and an illustration of how the candidate module is replaced by randomly created sub-derivation trees is given in Figure 6.1.

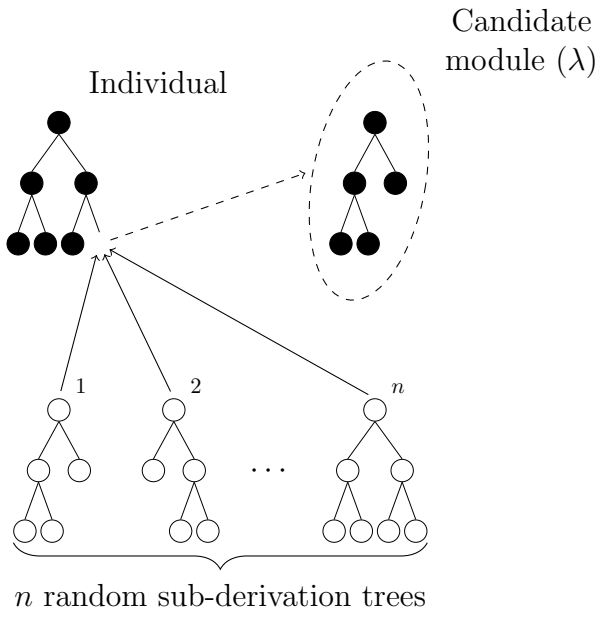


Figure 6.1: This figure shows how a candidate module is replaced by n randomly created sub-derivation trees of the same depth using the M-ID approach.

Algorithm 6.1 The mutation module identification (M-ID) algorithm. This is the same algorithm as the M-ID algorithm given in Chapter 4 (Algorithm 4.1). It has been given again for ease of reference.

```
M = new_list()
for Individual I in Population do
  I_f = get_fitness(I)
  λ = get_random_sub_derivation_tree(I)
  λ_d = get_depth(λ)
  i_λ = index_of_sub_derivation_tree_in_individual(I, λ)
  fitness_diffs = new_array_of_size(n)

  i = 1
  while i ≤ n do
    I' = copy_of(I)
    rand_i = create_random_sub_derivation_tree_with_depth(λ_d)
    replace_sub_derivation_tree_at_index(I', rand_i, i_λ)
    I'_f = get_fitness(I')
    fitness_diffs[i] = I_f − I'_f
    i++

  better_count = 0
  for Δf in fitness_diffs do
    if Δf < 0 then
      better_count++

  keep_candidate = true
  if better_count < round(ρ × n) then
    keep_candidate = false

  for Module μ in M do
    if get_sub_derivation_tree(μ) == λ then
      keep_candidate = false

  if keep_candidate then
    λ_f = mean(fitness_diffs)
    M.save_as_module(λ)
```

Insertion Identification (I-ID): For this module identification approach, n test individuals are generated using the same initialization method as the population. The fitness of each test individual is calculated and recorded. Next, like the M-ID method, each individual, I, is iterated and a candidate module, λ, is randomly picked from I. The depth, λ_{d}, and symbol of the root node, λ_{r}, of λ are also recorded. Then, a of copy each test individual is created and a randomly picked sub-derivation tree with the same depth as λ_{d} and the same root node symbol as λ_{r} is replaced with λ. After λ has been inserted into a test individual, it is re-evaluated and the difference between its original and new fitness values is recorded. If ρ of the stored fitness difference values for λ are less than 0 and there are no pre-existing modules with the same sub-derivation tree as λ, the mean of the saved fitness differences is assigned as the module's fitness, λ_{f}, and the module is saved in a module list. The pseudocode for this method is given in Algorithm 6.2 and an illustration of how the candidate module is inserted into randomly created individuals is given in Figure 6.2.

Algorithm 6.2 The insertion module identification (I-ID) algorithm

```
M = new_list()

i = 0
test_individuals = new_array_of_size(n)
test_fitness = new_array_of_size(n)
while i < length(text_individuals) do
  test_individuals[i]=(initialize_new_individual())
  test_fitness[i]=get_fitness(test_individuals[i])
  i++

for Individual I in Population do
  λ = get_random_sub_derivation_tree(I)
  λ_d = get_depth(λ)
  λ_r = get_root_node_symbol(λ)
  fitness_diffs = new_array_of_size(n)

  j = 0
  while j <length(test_individuals) do
    I'_T = copy_of(test_individuals[j])
    i_λ = find_node_index_with_depth_and_symbol(I'_T, λ_d, λ_r)
    replace_sub_derivation_tree_at_index(I'_T, λ, i_λ)
    fitness_diffs[j] = test_fitness[j] - get_fitness(I'_T)
    j++

  better_count = 0
  for Δf in fitness_diffs do
    if Δf < 0 then
      better_count++

  if better_count >= round(ρ × n) then
    keep_candidate = false

  keep_candidate = true
  for Module μ in moduleList do
    if get_sub_derivation_tree(μ) == λ then
      keep_candidate = false

  if keep_candidate then
    λ_fit = mean(fitness_diffs)
    M.save_as_module(λ)
```

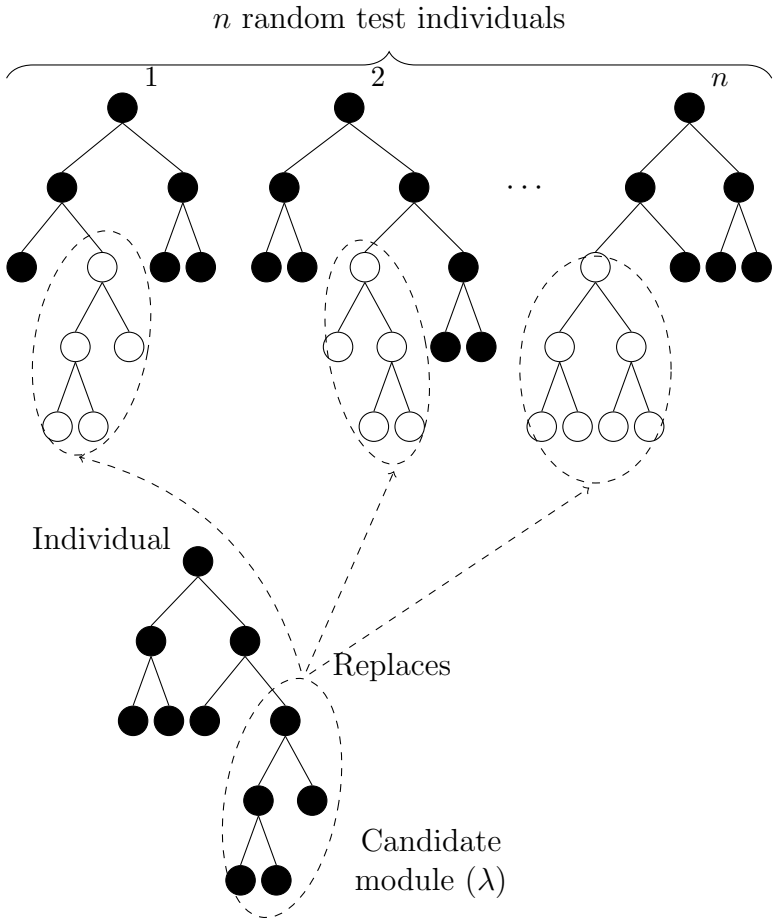


Figure 6.2: This figure shows how a candidate modules replaces n sub-derivation trees in the set of randomly created test individuals in the I-ID approach.

Frequency Identification (F-ID): This method counts the occurrence of every sub-derivation tree, σ, in the population, except for single non-terminals. The most common sub-derivation trees are used as modules and given fitness values based on their frequency: $\lambda_{\mathrm{f}} = \frac{\#\text{ of occurrences}}{\text{total }\#\text{ of subtrees}}$. Pseudocode for this approach is given in Algorithm 6.3.

Random Identification (R-ID): This method picks a random sub-derivation tree from each individual in the population and creates a module out of it. Since no evaluation of the module occurs, the fitness of the parent individual is used as the module's fitness: $\lambda_{\mathrm{f}} =$ fitness of parent. This method's pseudocode is shown in Algorithm 6.4. While this form of module identification is called "Random Identification," it has some non-random aspects that should be noted. Because modules are selected from individuals, as opposed to being randomly generated, they will contain information that has been developed over the course of evolution. Because these modules are also assigned their parent's fitness value, modules from highly fit individuals will be more likely to persist in the population than modules from less fit individuals regardless of their content.

Algorithm 6.3 The frequency module identification (F-ID) algorithm

```
subtree_map = new_map()
for Individual I in Population do
  for sub-derivation tree σ in I do
    if subtree_map.has_key(σ) then
      σ_count = subtree_map.get(σ)
      σ_count++
      subtree_map.put(σ, σ_count)
    else
      subtree_map.put(σ, 1)
```

Algorithm 6.4 The random module identification (R-ID) algorithm

```
M = new_list()
for Individual I in Population do
  λ = get_random_sub_derivation_tree(I)
  λ_fit = get_fitness(I)
  keep_candidate = true

  for Module μ in moduleList do
    if get_sub_derivation_tree(μ) == λ then
      keep_candidate = false

  if keep_candidate then
    M.save_as_module(λ)
```

Each of these methods was developed with a particular motive. The M-ID approach samples how the picked sub-derivation tree contributes to the fitness of the individual it appears in. I-ID estimates how well a sub-derivation tree performs in multiple individuals, as opposed to only the individual in which it appears. The R-ID method is added as a control to examine how M-ID, I-ID, and F-ID compare to a random approach. Testing this set of module identification methods will give insight into what types of approaches perform best on certain problems. Under each identification approach, once a module is identified, the phenotype of that module is "locked," meaning it is not allowed to be modified by crossover or mutation events. Unlike some of the previously developed approaches to modularity, such as ADFs, the content of modules are unchangeable. Their evaluation suggests that their current phenotypes are useful as they are (except with the R-ID method which performs no evaluation of modules).

6.2 Experimental Setup

The aim of this work is to examine different methods for identifying modules and to establish which of these different methods are more, or less, beneficial for different problems. As per the previous experimental chapters (Chapters 4 and 5), these approaches are used on the Santa Fe Ant Trail, $x^5 - 2x^3 + x$ Symbolic Regression, Even 7 Parity, and 8×8 Lawn Mower problems. After modules have been identified, they are made available to the population by using the **expand** method discussed in Chapter 5. All tests for statistical significance that are performed use a Wilcoxon rank-sum test with a 95% confidence interval.

Table 6.1: Experimental setup for all evolutionary runs unless otherwise noted. These parameters are taken from Chapter 5.

Parameter	Value
Independent Runs	50
Fitness Evaluations	100000
Population	500
Selection	Tournament (Size 5)
Wrapping	None
Crossover	Single Point (90%)
Mutation	Int Flip (1%)
Elites	50
Initialization	Ramped Half and Half
Replacement	Generational
Max. Derivation Tree Size	25 (Lawn Mower - 100)
κ	20
ρ	75% of n
n	50
τ	20

6.3 Initial Results with Module Identification

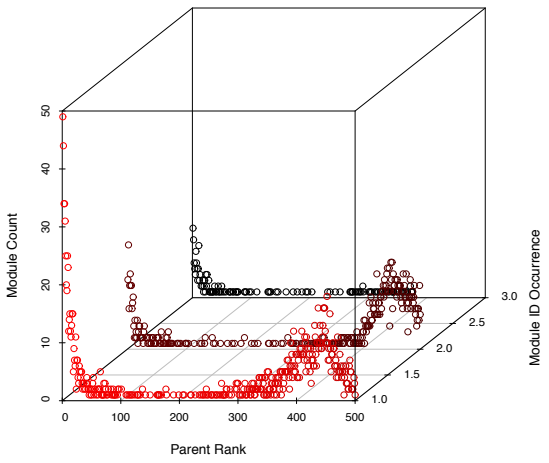
The first set of experiments carried out compare GE and the four approaches to modularity described in Section 6.1. The results from these experiments show that there are large discrepancies between how many generations each approach completes before the fitness evaluation limit is reached. In particular, the M-ID and I-ID methods finish approximately 59 and 99 generations respectively before reaching the fitness evaluation limit. This is due to the number of fitness evaluations used to identify modules instead of evolving the population. This is a noticeable contrast to F-ID, R-ID, and GE which finish between 190 and 200 generations depending on the problem and approach. There are two sources for this deviation in the number of generations completed:

1. the number of times each candidate module, λ, is evaluated (n from Chapters 4 and 5) and,

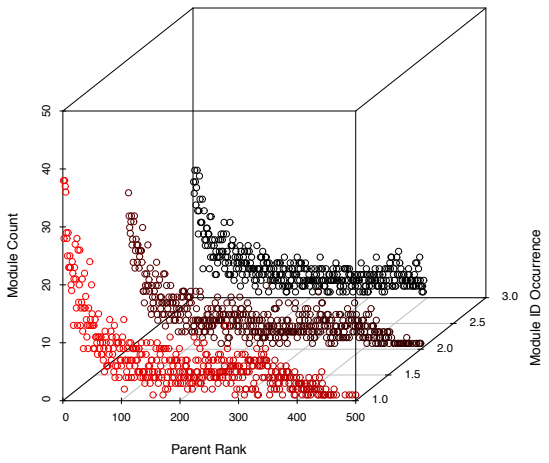
2. the number of individuals that candidate modules are picked from.

Permitting fewer individuals to contribute candidate modules and reducing the number of fitness evaluations used to identify candidate modules, evolution will have more generations to use the discovered modules to improve the performance of the population. Even if the M-ID and I-ID methods are finding better modules than the R-ID and F-ID methods, evolution is not given many generations to work with those modules to create better solutions. To address the second point above, the individuals of the population from which the most modules are identified is given. It is possible that each individual contains information that may be valuable as a module. But the probability of finding good modules in low fitness individuals is low in comparison to the probability of finding good modules in the most fit individuals. If individuals that are likely to provide good modules can be identified, focus can be given to them and the rest of the population can be ignored, saving many fitness evaluations.

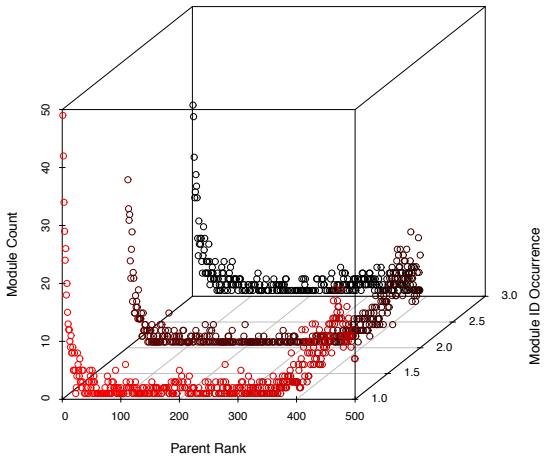
To determine which individuals focus should be given to, the number of modules that come from different ranked individuals at the module identification steps is plotted. The graphs are given in Figure 6.3. Each of the plots in Figure 6.3 shows that the most modules are identified from the highest ranking individuals in the population. This suggests that the top-ranking individuals are more valuable in terms of contributing modules and focus can be given to them. Based on the data provided in Figure 6.3, it is estimated that the most modules are identified from the top $7 - 10\%$ of the population. But this area of the population is only comprised of elite individuals. For future experiments, the top 15% of the population is used for discovering modules because it allows individuals that are fit but not preserved through elitism to contribute modules. On all problems except the Even 7 Parity, there are small peaks in the lower ranks of individuals showing that a notable number of modules are coming from less fit individuals. One possible explanation for this may be due to the fitness diversity in the population. As evolution progresses, the population tends to be less diverse in terms of the fitness values of the individuals. It is



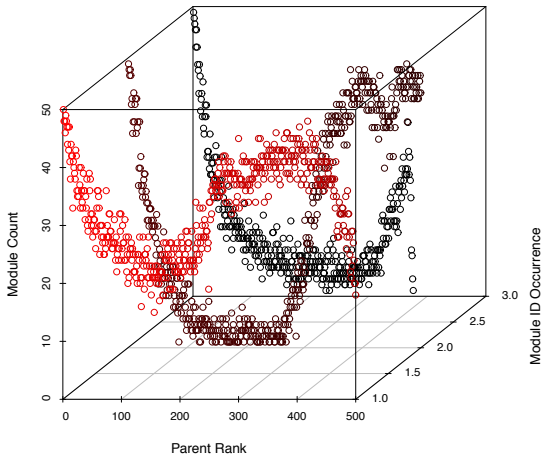
(a) Santa Fe Ant Trail



(b) Even 7 Parity



(c) $x^5 - 2x^3 + x$ Symbolic Regression



(d) 8×8 Lawn Mower

Figure 6.3: These figures show which section of the population contributes modules during the identification process. The x-axis represents an individual's rank in the population based on fitness, 0 being the best and 500 being the worst. The y-axis (height) represents how many times individuals with a given rank contributed a module over the course of 50 runs. This value does not take into account the quality of the module discovered. The z-axis (depth) denotes how many times modules have been identified. The M-ID approach was used for these figures. Module parent rank data for the R-ID approach is given in Figure A.1 in Appendix A.1.

also possible that the diversity of the population's fitness will increase over time. However, the data observed in this thesis shows that an increase in diversity occurs either in early generations and is followed by a decrease in diversity for the rest of the run, or persists throughout the run but never reaches a high level of diversity. Figure 6.4 shows how the diversity of fitness values in a standard GE run change over time on the Santa Fe Ant Trail problem. In this figure, the diversity of fitness values is measured using entropy[1] [145]. As modules are being taken from the top individuals, if they are discovered again in later individuals, they are rejected since duplicate modules are not kept. However, the least fit individuals are likely to have been more recently created by crossover and mutation events. The modules discovered here may be different due to the variety of phenotypes in this area of the population. This is supported by the movement of this peak in Figure 6.3(d). For the first module identification step in this figure, the upwards slope of the data begins around the 150^{th} to 200^{th} ranked individual. However, for the second module identification step, the upwards motion of the data does not begin until just before the 300^{th} ranked individual. During the third module identification step, this slope begins still later around the 310^{th} ranked individual. The data presented in Figure 6.3 only account for the M-ID method. An interesting characteristic of the I-ID approach is that it rarely discovers any modules at all. In fact, for most runs it discovers no modules at all. This suggests that the value of ρ (75% of n) is too strict of a threshold for this particular identification method. Additional modules may be identified with a decreased ρ value.

After considering which individuals yield the most modules across all problems and how many modules are being identified, the following was implemented:

1. lower values of ρ for the M-ID and I-ID approaches,

2. selecting modules from the top 15% of individuals for all modules identification ap-

[1]Entropy is used to measure uncertainty associated with a random variable. Shannon[145] gives the formula for calculating entropy as $\mathrm{H}(x) = -\sum_{i=1}^{n} p(x_i)\, log_b p(x_i)$, where $p(x_i)$ is the probability of outcome x_i. Entropy is commonly used to measure uncertainty or volatility.

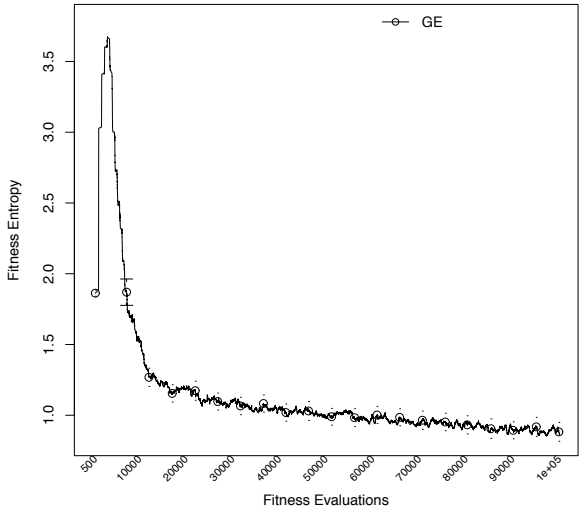


Figure 6.4: This figure shows how the diversity of the population's fitness values changes over time during standard GE runs. The data is averaged across 50 runs. Diversity is measured using Shannon entropy [145]. Fitness diversity data for the remaining three benchmark problems is given in Appendix A.1 in Figure A.2.

proaches,

3. and lower values of n to reduce the number of fitness evaluations used to identify modules.

A final extension is also implemented based on data from the initial experiments. The largest improvements in the fitness of the population are made in the first 20 generations. Modules are only identified every 20 generations, meaning they are not available initially for evolution to build upon and take advantage of. The final variation identifies modules from the initial generation, in addition to every 20 generations. The following section analyzes the results of every permutation of these variations when applied to each of the original setups discussed in Section 6.1.

6.4 Effects of Module Identification on Parameters

This section discusses the results of applying the additional variations from Section 6.3 to the module identification methods. Analysis of the various methods is broken down by the variations mentioned in Section 6.3.

6.4.1 Reducing Module Selection Pressure

One of the observations made in Section 6.3 is that the I-ID method finds very few modules in total. The reason for this is that it is difficult to find sub-derivation trees that can improve the fitness of multiple individuals. Under the I-ID approach, ρ corresponds the number of individuals that must be improved by a candidate module. Higher values for ρ mean more difficulty for candidate modules to be accepted for use in GE's grammar. In this section, the value for ρ is reduced to facilitate more candidate modules passing their evaluations and being made available to individuals. Values of $\rho = 75\%$, 50%, and 25% of n are examined on both the M-ID and I-ID methods on all four benchmark problems mentioned in Section 6.2. For the experiments carried out in this section, modules are identified from the entire population and each module is evaluated 50 times ($n = 50$). The fitness data gathered and statistical tests on this data is given in Tables A.1 - A.8 in Appendix A.2.

For all of the problems except the Santa Fe Ant Trail, I-ID finds no modules at all when $\rho = 75\%$ of n (See Figure 6.5). When ρ is reduced to 50% of n, still, very few modules are identified. On 8×8 Lawn Mower and Even 7 Parity problems, no modules are found. However, with the lowest value of $\rho = 25\%$ of n, more modules are discovered for all problems except Even 7 Parity where no modules are discovered. In the case of the Even 7 Parity problem, the candidate modules are not passing the required number of evaluations to become full modules. Unlike the M-ID approach, variations of the I-ID

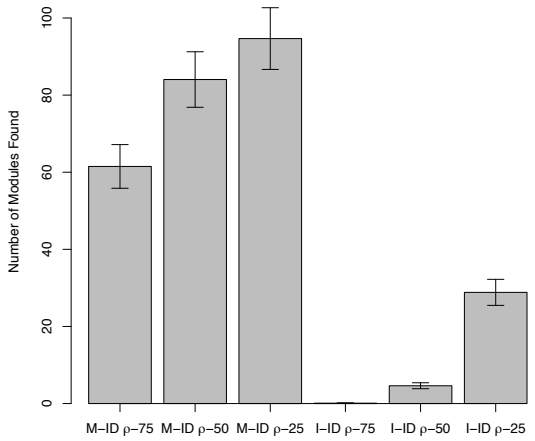
method that use a lower value for ρ have better fitness than those that use higher values, but statistical tests show no significance between these approaches.

The graphs in Figure 6.5 show that the M-ID approach has no issue identifying many modules during its runs, but having a more strict threshold such as $\rho = 75\%$ of n actually yields better fitness values than the lower values of 50% and 25% of n. Depending on the benchmark problem and the value of ρ, anywhere from 60 to almost 650 modules are identified. This suggests that when sufficient numbers of modules are being discovered, having a more rigorous filter on how many modules pass their evaluation results in better fitness values. Although, in the data examined there was no statistical significance amongst the M-ID methods.

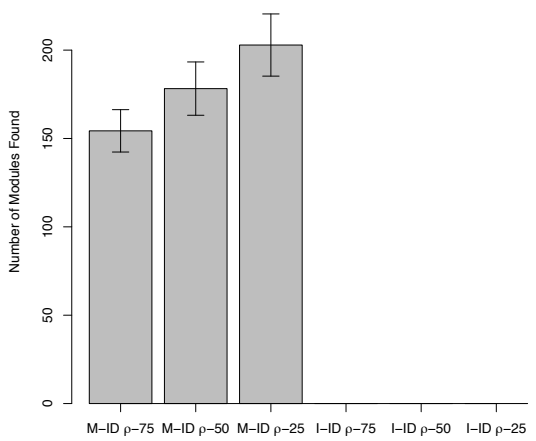
The lesson to be taken away from these experiments is that an appropriate balance must be found for the amount of modules identified. If large numbers of modules are discovered, many could be "bad" modules. On the other hand, if too few modules are discovered, evolution is not able to use them to improve its search. To find this balance, the problem, module identification approach, and threshold for identifying modules must all be taken into consideration. For the M-ID method, it is able to identify anywhere from 60 to 600 modules. When so many modules are identified, using a more strict threshold for accepting modules is advantageous. But with the I-ID method, significantly fewer modules are discovered and a less strict criteria for accepting modules is more reasonable. For these reasons, all the experiments in the remainder of this chapter use $\rho = 75\%$ of n for variations of the M-ID method and $\rho = 25\%$ of n for I-ID approaches.

6.4.2 Reducing Fitness Evaluations

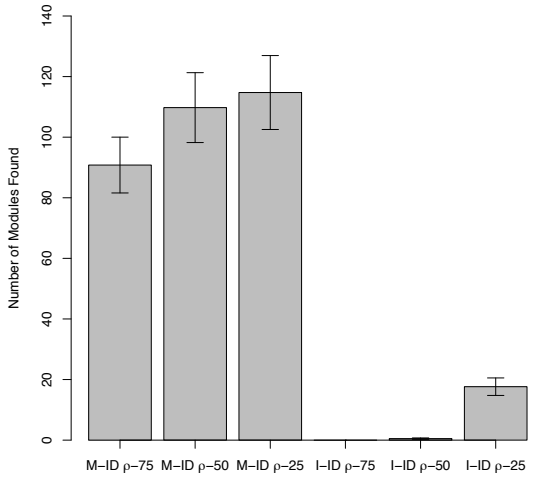
A problem with the M-ID and I-ID methods is that they both require additional fitness evaluations to identify modules. The drawback of this is that the fitness evaluations used in the module identification process are subtracted from the number of fitness evaluations



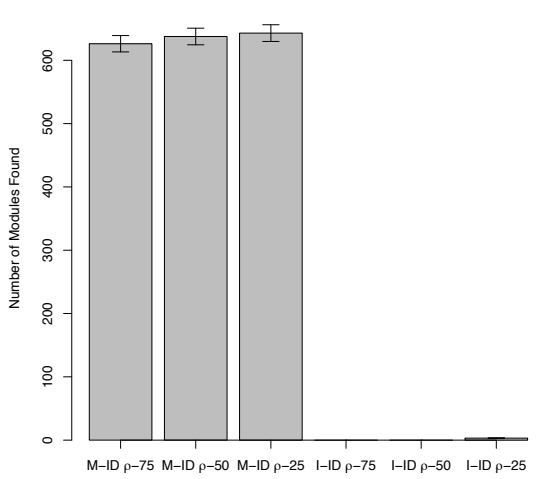
(a) Santa Fe Ant Trail



(b) Even 7 Parity



(c) $x^5 - 2x^3 + x$ Symbolic Regression



(d) 8×8 Lawn Mower

Figure 6.5: These bar charts show the total number of modules discovered using different module identification threshold (ρ) values over the course of evolution averaged over 50 runs. Note the difference in the scale of the y-axis for the different runs. It should also be noted that M-ID and I-ID discover modules in significantly different manners, and the quantity of modules by the two approaches can not be fairly compared.

that may be used for the evolution of the population. While the identified modules may be beneficial and speed up evolution, sufficient generations are required for evolution to take advantage of these modules. It would be an improvement for the M-ID and I-ID methods if fewer fitness evaluations can be used during module discovery. This would afford evolution more time to work with the existing modules and individuals to further improve the performance of the population. As was discussed in Section 6.3, identifying modules from only the top 15% of individuals and reducing the number of evaluations that is performed on candidate modules (the value for n) can greatly reduce the fitness evaluation cost of identifying modules. For the experiments carried out to test these variations, each permutation of identifying modules from the entire population, only the top 15% of the population, $n = 50$, $n = 25$, and $n = 10$ are used. Based on the data given in Section 6.4.1, the M-ID approaches in this section use a ρ value of 75% of n and I-ID approaches use a ρ value of 25% of n. The fitness data gathered and statistical tests on this data is given in Tables A.9 - A.16 in Appendix A.3.

The first point to discuss is that across all problems the original setup for M-ID and I-ID of identifying modules from the entire population using a value of 50 for n has a worst average best fitness than most, if not all, of the other approaches of the same identification method. One exception to this is the I-ID method on the 8×8 Lawn Mower problem. In this case, I-ID methods taking modules only from the best individuals are the worst performing approaches. On the 8×8 Lawn Mower and $x^5 - 2x^3 + x$ Symbolic Regression problems, both the M-ID and I-ID approaches are outperformed significantly by at least one variation that uses fewer fitness evaluations. On the Santa Fe Ant Trail problem, methods which use less fitness evaluations are able to significantly outperform the original M-ID setup. There are approaches with better fitness values than the original I-ID method, but the results are not significant. There are methods which use less fitness evaluations that are able to outperform the original M-ID and I-ID approaches on Even 7 Parity problem in

a statistically significant manner. These results present a good case for limiting the fitness evaluations used to discover modules. It shows the previous used value $n = 50$ as excessive in its use of fitness evaluations and smaller values of n (25 and 10 in this case) are more appropriate.

An additional observation worth mentioning is that there appears to be no trend in the performance of approaches which identify modules from the entire population compared to approaches which only identify modules from the most fit 15% of individuals. The collected data shows that the performance of these variations is problem dependent and may vary even within the problems themselves. For example, on the Santa Fe Ant Trail problem, the best performing approach to modularity is I-ID searching the top 15% of individuals and using $n = 10$. However, the second through fourth best performing approaches to modularity use the entire population to identify modules. Then, the fifth best uses on the top 15% of the population. The lack of a constant best approach is seen in all the benchmark problems and suggests that under the M-ID, R-ID, I-ID, and F-ID approaches, using the entire population or using just a portion of it for module parents is not significant.

The data examined in this section does not conclusively show that any one combination of a.) reducing the number of evaluations each module undergoes (the value for n) and b.) searching only the best individuals for modules performs best on any of the problems examined. It does, however, show that searching the entire population for modules and evaluating each module 50 times is more generally more harmful than good. Based on the results gathered from the experiments performed for this section, the combination of identifying modules from the entire population and using $n = 50$ will be omitted from further sections in this chapter.

6.4.3 Initial Generation Identification

The final variation to identifying modules mentioned in Section 6.3 is identifying them after the initial generation of individuals has been evaluated. This has been proposed as an additional extension to each of the module identification methods because it allows the population to work with modules from an early stage in evolution. In the benchmark problems examined in this thesis, GE's performance improves in the largest intervals during the first 20 to 40 generations. After this point, the progress of evolution slows down. By making modules available at the earliest stages of evolution, the population may learn how to best exploit the modules and improve its performance over that of standard GE. The results reported in this section cover the M-ID, I-ID, R-ID, and F-ID methods. Each combination of identifying module from the entire and top 15% of the population and n = 10, 25, and 50 are used. As was done in Section 6.4.2, the value of ρ for the I-ID approach is 25% of n and 75% for the M-ID approach. The exception to this is variations identifying modules from the entire population or using $n = 50$. Results discussed in Section 6.4.2 show this combination of parameters to be one of the consistently worst performing approaches across all the benchmark problems examined. The data discussed in this section can be found in Tables A.17 - A.24 in Appendix A.4.

The results of identifying modules after the initial generation show that the fitness of many of these approaches improves over the corresponding approach that does not identify modules in the first generation. This is improvement in fitness is particularly prevalent in the Santa Fe Ant Trail and Even 7 Parity problems. However, there is a danger to identifying modules an additional time. For the M-ID and I-ID approaches, this extra module identification will expend more fitness evaluations discovering modules. The benefit of making modules available to the population in the early generations of evolution may be overshadowed by the loss of fitness evaluations that would otherwise be used in the evolution of the population. Returning to the issue of using fitness evaluations to identify

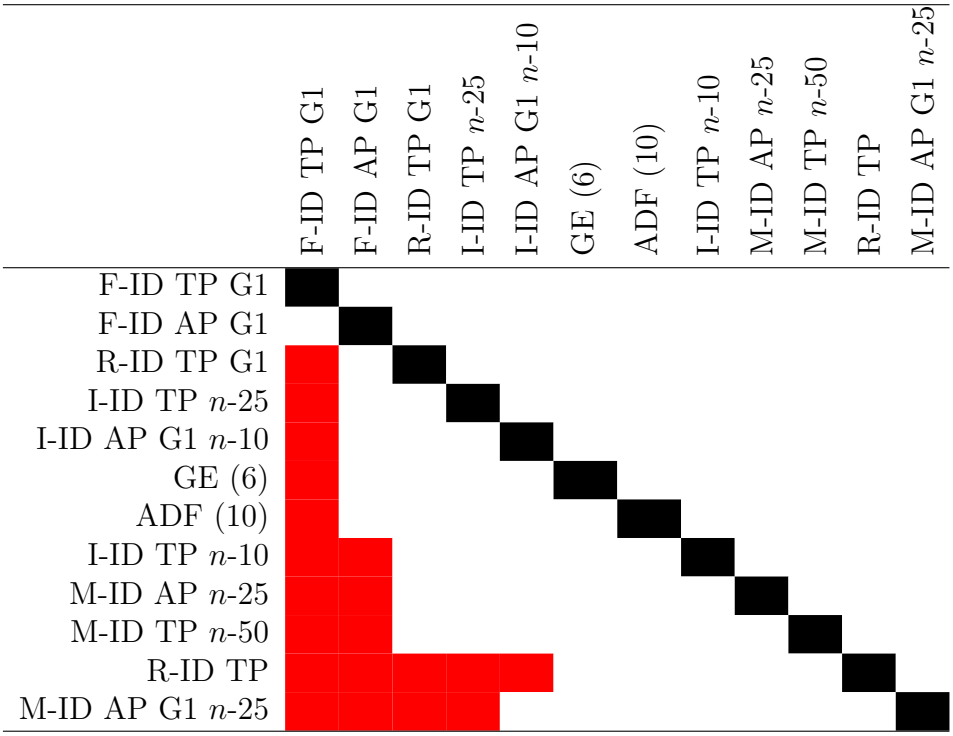
modules, at least one of the top 5 performing approaches to modularity is a variation of the R-ID or F-ID methods, and for every problem except the 8×8 Lawn Mower, at least one of the R-ID or F-ID methods identifies modules after the initial generation. These results suggest that the extra module identification occurrence at the beginning of evolution is especially useful when used with methods that use no fitness evaluations to find modules.

6.4.4 Comparison to Standard GE

Thus far, Sections 6.4.1, 6.4.2, and 6.4.3 examine various parameters that can be used with the module identification approaches defined in Section 6.1. The analysis given in these sections compares the various parameters for module identification against each other and discusses how said approaches alter GE's fitness. However, it does not address one of the most important performance comparisons: how do the various approaches compare to standard GE? This section covers this topic. GE with ADFs is also added as an additional benchmark for comparing how the methods for module identification described in this chapter perform in relation to an already established method for incorporating modularity into GE.

Due to the many variations examined in the preceding sections and the large quantity of data gathered, only the top 5 approaches for module identification, bottom 5 approaches for module identification, standard GE and GE with ADFs are discussed here. Tables 6.2, 6.3, 6.5, and 6.4 present the results of Wilcoxon rank-sum test comparing each of these approaches (The complete set of data for all approaches, not just the top and bottom five, can be found in Appendix A.5 in Tables A.25 - A.32.). In these tables at least one of the approaches to modularity defined in Section 6.1 achieves a statistically significant improvement in fitness over standard GE for each problem. Only on the Santa Fe Ant Trail and $x^5 - 2x^3 + x$ Symbolic Regression are one of the M-ID, I-ID, R-ID, or F-ID approaches able to significantly outperform GE with ADFs. One interesting observa-

Table 6.2: This table shows the 5 best and worst methods for identifying modules based on their average best fitness at the end of 50 independent runs on the Santa Fe Ant Trail problem. Red cells indicate a significant difference between approaches based on a Wilcoxon rank-sum test with a confidence interval of 95%. Rows and columns with TP in their labels denote approaches where modules are identified only from the top 15% of the population. Rows and columns with AP in their labels mark methods in which modules are identified from the entire population. The G1 in row and column labels means modules are identified after the initial generation.



tion about these results is that the same approach to modularity and parameters for that approach are never the best performing on more than one problem. This speaks for the problem dependence of the various methods for identifying modules and the parameters they use. Another trend in these tables is that approaches which perform significantly better than standard GE often use as few as possible additional fitness evaluations to discover modules. The 8×8 Lawn Mower problem is an exception to this, where the M-ID methods significantly outperform all other approaches. However, GE with ADFs is able to solve many more instances of the 8×8 Lawn Mower problem than any other approach examined.

Table 6.3: This table shows the 5 best and worst methods for identifying modules based on their average best fitness at the end of 50 independent runs on the Even 7 Parity problem.

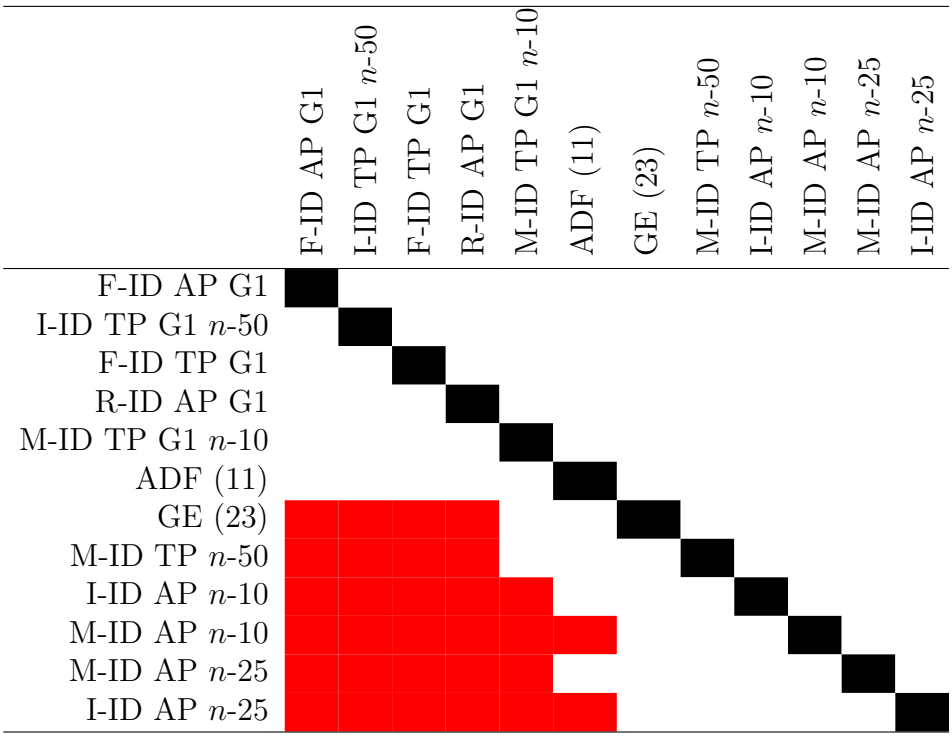
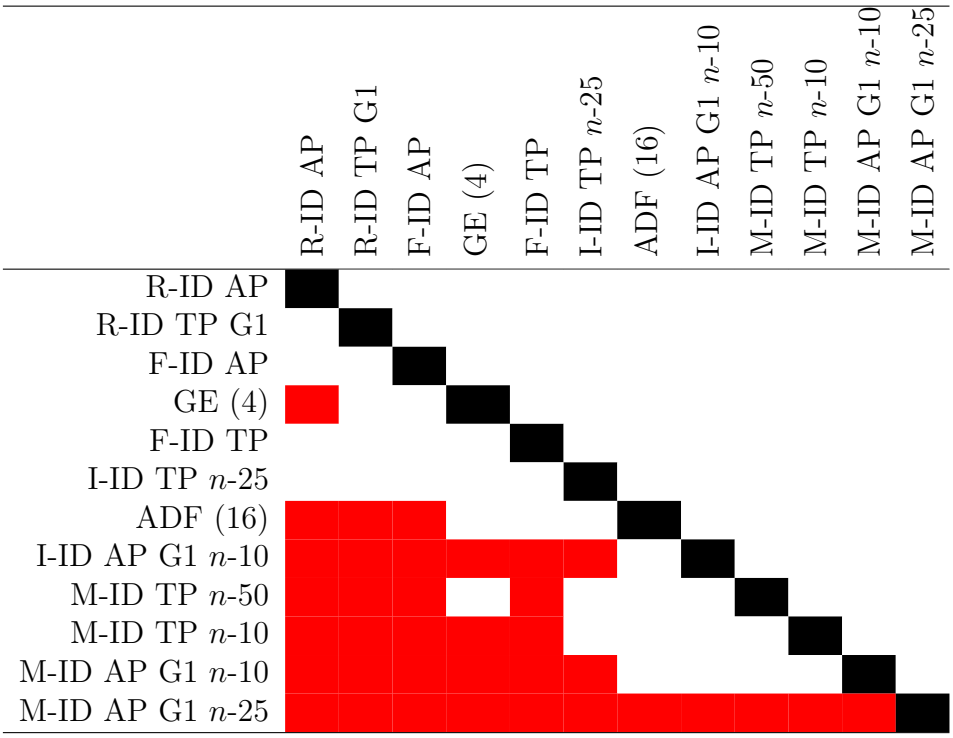
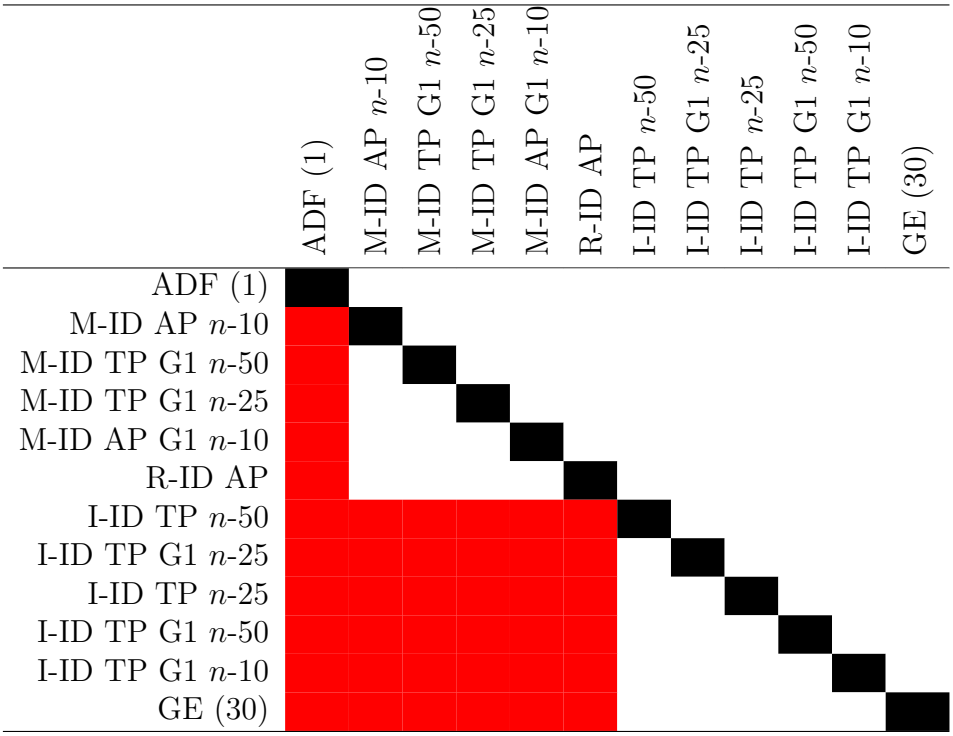


Table 6.4: This table shows the 5 best and worst methods for identifying modules based on their average best fitness at the end of 50 independent runs on the $x^5 - 2x^3 + x$ Symbolic Regression problem.



Based on the data collected in these experiments, it can be stated that each of the methods for identifying modules introduced in this chapter is able to significantly improve the performance of standard GE when used with certain parameters on an appropriate problem. However, identifying which module identification method and parameters should be used can be a long and painstaking task. The results presented in this section suggest that a "less is more" approach to using fitness evaluations for module discovery is generally a wise idea, and making modules available to the population from an early generation is also useful.

Table 6.5: This table shows the 5 best and worst methods for identifying modules based on their average best fitness at the end of 50 independent runs on the 8×8 Lawn Mower problem.



6.5 Summary

This chapter defines and examines four methods for identifying modules in GE and compares them to standard GE and GE with ADFs. The different approaches are tested on four different benchmark problems, and various parameters for the module identification operations are examined. Specifically, the individuals modules are taken from, how many times candidate modules are evaluated, how strict the module evaluation process is, and if modules are useful in the initial generations are analyzed. The data collected in this chapter shows that each of these settings can play an important role in how the various approaches to module identification perform. Most importantly the following should be considered:

- How rigorous the "passing" criteria for candidate modules is should be tailored to the module identification approach;
- Fitness evaluations are precious to the progress of evolution and as few as possible should be spent identifying modules;
- Evolution benefits from being able to use modules in early generations.

Depending on how these parameters are tuned and the undertaken problem, at least one of the approaches to modularity is able to significantly outperform standard GE. An unexpected result is that the random and frequency-based approaches to discovering modules are usually the best performing when compared to the fitness-based approaches to finding modules. This speaks towards the difficulty of defining a problem-independent method capable of identifying useful modules. The good performance of the R-ID and F-ID on most of the problems examined in this chapter shows that identifying and making modules available to the population can be beneficial. However, more examination is needed to understand the characteristics of the modules that are being identified and used

to improve GE's search.

These points can be related to two of the questions presented in Section 1.2.1:

Research Question 1 - Should modules be identified using a fitness-based or a usage-based approach? The data shown in Sections 6.4 show that both fitness-based (M-ID and I-ID) and frequency-based (F-ID) approaches to identifying modules can yield better fitness values than standard GE. But the approach that gives a statistically significant improvement over standard GE varies from problem to problem. As an example, only one F-ID approach outperforms standard GE on the Santa Fe Ant Trail problem, but every M-ID variation outperforms standard GE and every F-ID variation on the 8×8 Lawn Mower problem. These results suggest that there is no single approach that can be applied to common GP and GE benchmark problems and achieve a significant improvement in fitness over standard GE.

Research Question 2 - How much computational power is reasonable to use in identifying modules? The results from the experiments carried out in Section 6.4 show that using as few additional fitness evaluations to evaluate modules as possible should always be a goal. But, it is also possible to sacrifice some fitness evaluations for the sake of discovering modules and still show some improvement in fitness. The R-ID and F-ID methods, which use no additional fitness evaluations to identify modules, are typically among the top-performing approaches. However, the one variation which added fitness evaluations and tended to show improvements in fitness was identifying modules in the first generation of evolution.

This chapter also presents a number of questions. The most interesting questions concern the kinds of modules that are being discovered by the various module identification methods and how they are used by the population. Do the various methods identify modules that encode large or small amounts of phenotypic information? Are modules used often by highly fit or unfit individuals? Does the quality of an individual that yields a

module affect how that module is used? These questions will be examined in Chapter 7. More questions arise from the work discussed in this chapter, but are out of the scope of this thesis. The most notable being: is there some way to combine aspects of the various module identification approaches defined in Section 6.1 to make a hybrid approach capable exploiting the benefits of each method?

Chapter 7

An Analysis of Modules

In Chapter 6, four methods of discovering modules are defined and examined with a variety of parameters. Chapter 6 explores different values for the following parameters:

- how many evaluations are performed on each module,
- how many evaluations a module must pass,
- what area of the population should contribute modules,
- and if modules should be identified from the initial population.

Two of the four module identification methods find modules based on estimates of how much they contribute to an individual's fitness (M-ID) and how much they contribute to multiple individuals' fitness (I-ID). The other two methods for module identification select modules randomly (R-ID) and based on their frequency of occurrence in the population (F-ID). The results of these experiments showed that there was always at least one module identification method capable of significantly outperforming standard GE on four different benchmark problems. However, an unexpected outcome of the previous chapter is that the R-ID and F-ID approaches were the only methods for identifying modules that constantly outperformed standard GE across all the problems. This chapter will examine the

modules found by the various module identification methods to determine what characteristics modules discovered by the best performing approaches exhibit. It also examines how the modules discovered by the various modules identification approaches are used by the population. Specifically, this chapter examines the sizes of modules found (Section 7.2), the phenotypic content of the modules (Section 7.3.1), and the meaning, or semantics, of the modules (Section 7.3.2). In addition to the characteristics of the modules themselves, how frequently modules are used over time and the fitness of individuals that use them is also examined in Section 7.4. The conclusions from this chapter answer Research Questions 1.2.1 and 1.2.1, which are proposed in Section 1.2.1.

7.1 Experimental Setup

In order to understand what causes the performance difference between the various approaches to modularity, the modules themselves are examined. More specifically, this chapter reports the differences in size, content, semantics, and usage of modules found by the different module identification methods. This examination will give insight into what characteristics of modules (size, phenotypic content, and meaning) are desirable, enabling the design and implementation of more effective module identification operations. The examination of each characteristic of the modules is performed on all four of the benchmark problems from Chapter 6: Santa Fe Ant Trail, Even 7 Parity, 8×8 Lawn Mower, and $x^5 - 2x^3 + x$ Symbolic Regression.

7.2 Module Size

To begin this study of the characteristics of discovered modules, their size is analyzed. Figures 7.1 - 7.4 shows the average depth of modules discovered by the various mod-

ule identification setups. By examining the trends (or the lack of trends) exhibited by the various module identification methods, insight into which sizes of sub-derivation trees make better modules can be gained. Identifying appropriate sizes for modules will aid in designing more efficient future module identification methods.

7.2.1 Santa Fe Ant Trail

On the Santa Fe Ant Trail problem (Figure 7.1), there appears to be no trend that differentiates the best and worst performing approaches in terms of module size. The best performing method on this problem is frequency identification (F-ID), identifying modules in the first generation and only from the top 15% of the population. On average, this setup discovers modules from sub-derivation trees with depths between five and seven. These depths are between 65% and 92% of the depth of the derivation trees of their parents. However, on this problem, many of the approaches that perform poorly discover similar sized modules. GE with ADFs also generates ADFs which are similar in size to the modules discovered by the various other module identification approaches.

7.2.2 Even 7 Parity

In Figure 7.2, the modules discovered by the Even 7 Parity problem show a slightly different trend. The best performing approach is F-ID, identifying modules from the entire population starting in the first generation. This method discovers smaller modules with depths between three and four. These modules comprise between 35% to 44% of the parent individuals' total depths. All the F-ID approaches discover similar size modules, but the R-ID, I-ID, and M-ID methods discover modules with depths between four and eight. This suggests that for the Even 7 Parity problem, identifying shallower modules is more beneficial. GE with ADFs sets itself apart from the other approaches by generating ADFs

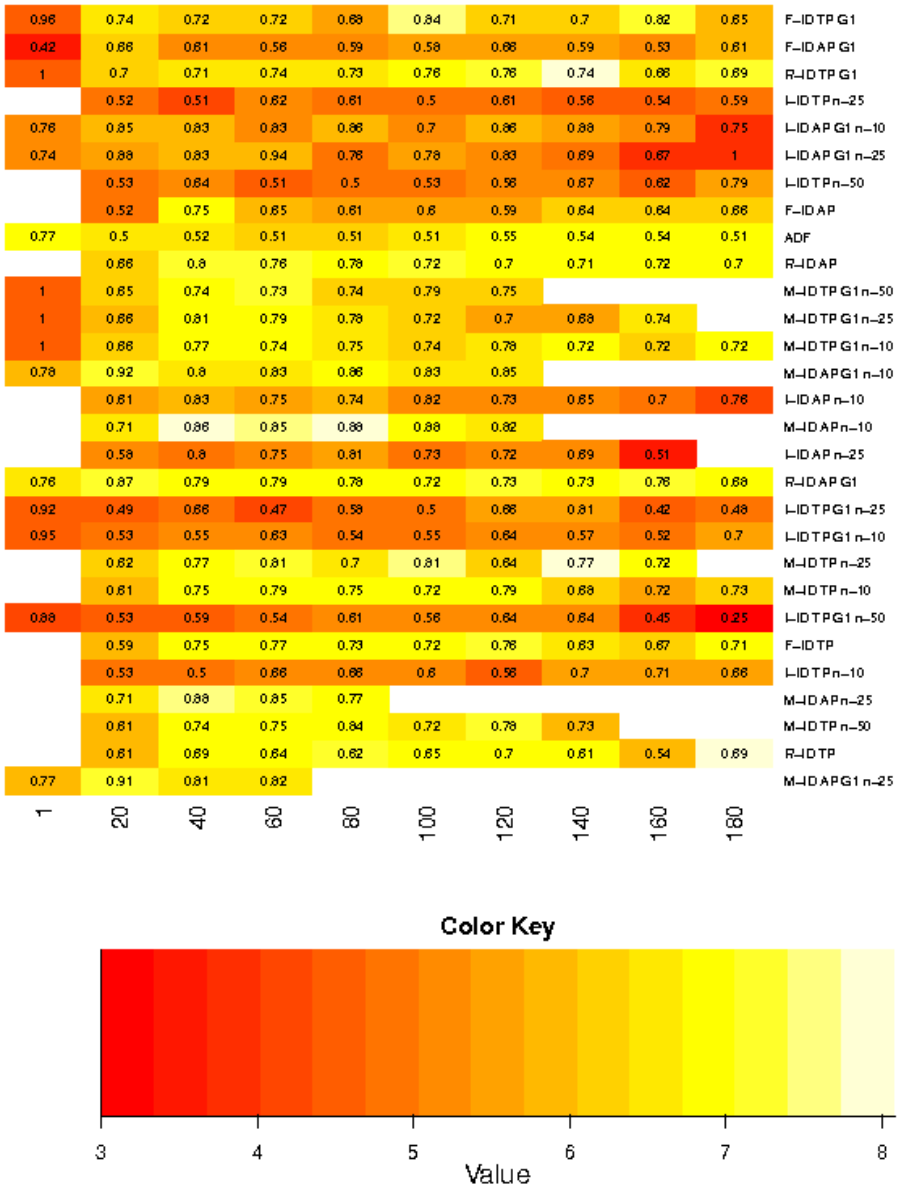


Figure 7.1: This figure shows the average depth of modules discovered by the various module identification setups on the Santa Fe Ant Trail problem. The x-axis on each heatmap represents the generation at which modules are identified. The y-axis shows the module identification variations sorted by their end-of-run average best fitness from top to bottom, top being the best and bottom being the worst. The color of each cell represents the average depth of modules discovered. The numeric value in each cell is the average ratio of the modules' depth compared to their parents. Blank (white) cells denote generations where no new modules are identified. The color key beneath the heat map represents the depth of modules discovered.

that are almost exclusively between depths 7 and 8.

7.2.3 $x^5 - 2x^3 + x$ Symbolic Regression

The only approach able to outperform standard GE on the x^5-2x^3+x Symbolic Regression problem is R-ID identifying modules from the entire population. This setup shows an increase in module size, from depths of four to fourteen, as more module identification operations occur. The modules identified increase from 72% the depth of their parents' derivation trees to 98%. This growth in module size suggests that evolution is able to take advantage of larger and larger modules as the generations progress. Part of this behavior could be attributed to bloat in the modules. However, there are other approaches to module identification that begin by identifying small modules and gradually identify larger modules that perform much worse than R-ID taking modules from entire population.

7.2.4 8×8 Lawn Mower

A new trend is shown by the 8×8 Lawn Mower problem in Figure 7.4. The top four approaches are all mutation identification (M-ID) methods. Each of these variations begins by identifying smaller modules, then identifying larger modules, and finally identifying smaller modules once more. To contrast this, the random (R-ID), insertion (I-ID), and F-ID methods show different behaviors. Both F-ID and R-ID start by finding small modules and, at each identification step, find larger modules. The I-ID, on the other hand, identifies only small modules for the duration of each run. This suggests that on this problem larger modules are more beneficial in earlier generations, but their utility expires and smaller modules become more useful. An interesting feature of GE with ADFs is that they are by far the best performing approach (recall the results from Section 6.4.4), yet they generated very small ADFs when compared to the modules generated by the best performing M-ID

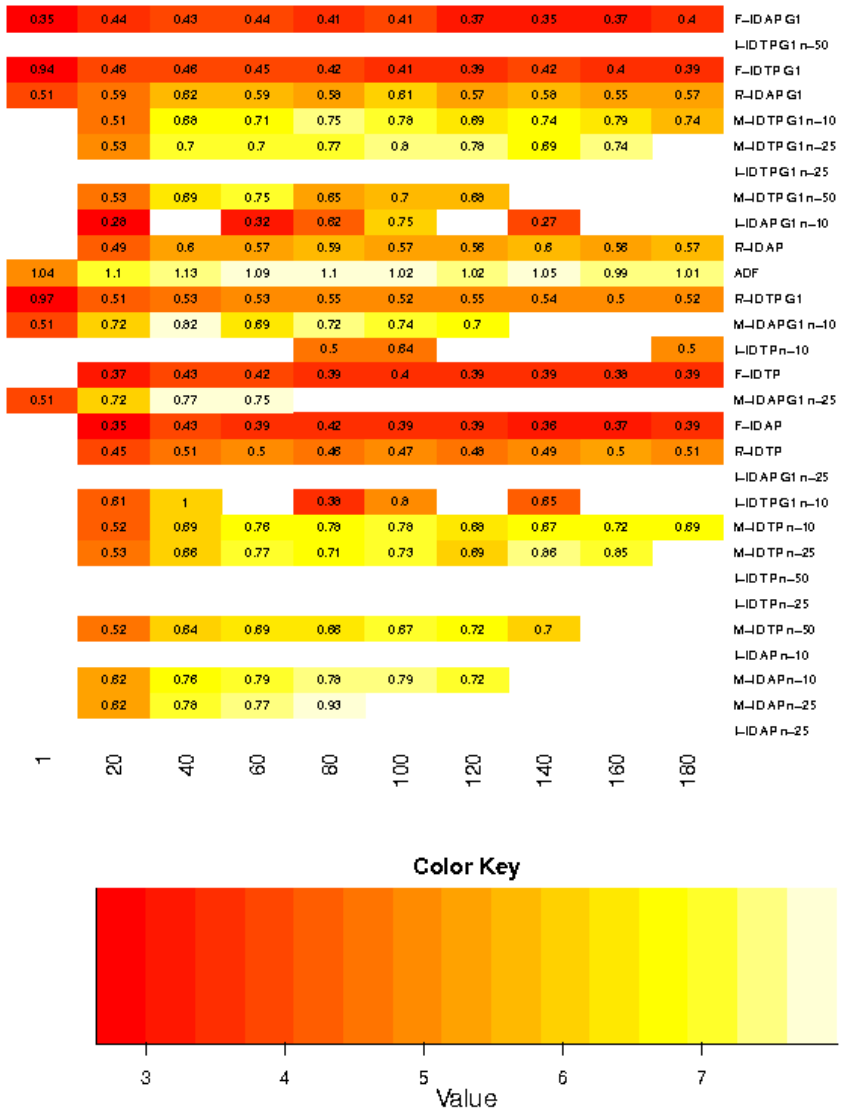


Figure 7.2: This figure shows the average depth of modules discovered by the various module identification setups on the Even 7 Parity problem.

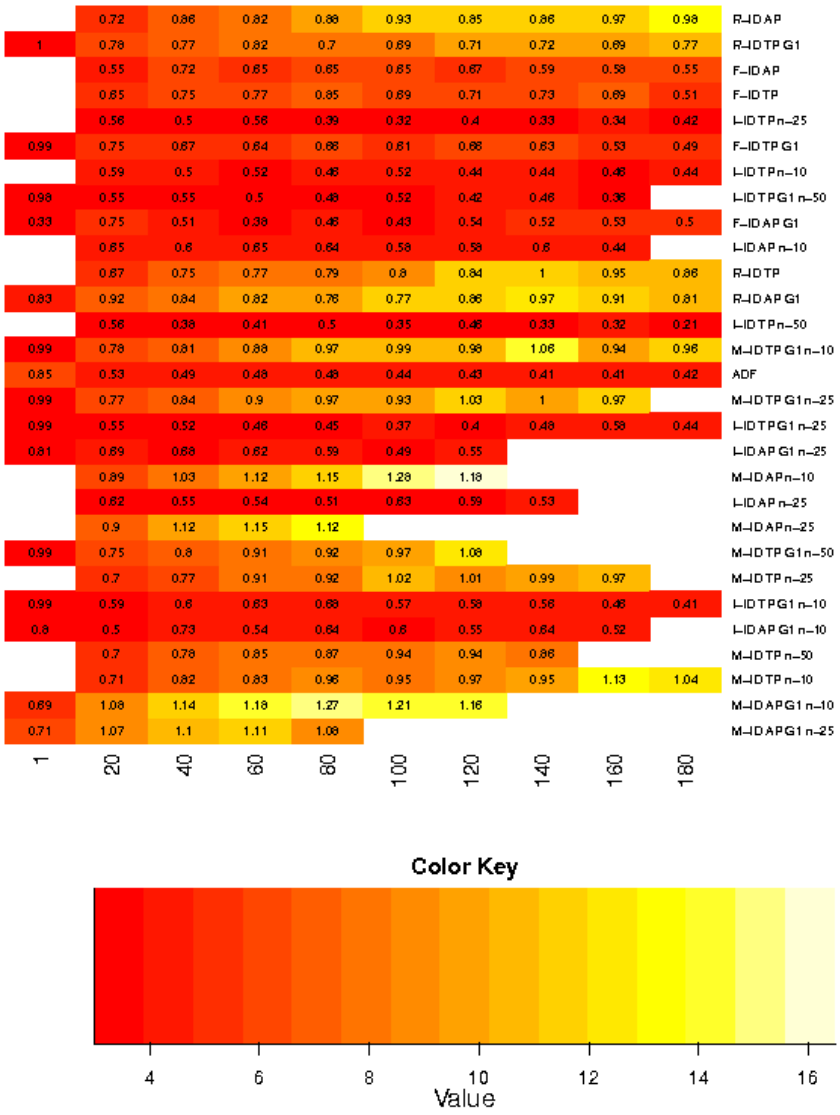


Figure 7.3: This figure shows the average depth of modules discovered by the various module identification setups on the $x^5 - 2x^3 + x$ Symbolic Regression problem.

methods. This shows that GE with ADFs are able to evolve smaller and more efficient modules on the Lawn Mower problem. It also suggests that there is a large amount of bloat in the modules discovered under the M-ID and R-ID methods.

7.2.5 Discussion

This analysis gives some insight into the sizes of modules on four benchmark problems. What can be seen is that modules of various sizes are more or less beneficial depending on the problem under examination. While the best approaches for identifying modules tend to exhibit particular behavior in terms of the sizes of modules discovered, worse-performing approaches may also show the same behavior. These results also suggest that bloat in the modules is an issue that should be considered. On both the Even 7 Parity and $x^5 - 2x^3 + x$ Symbolic Regression problems, the single-best performing method identifies small modules for the duration of evolution. Another set of parameters which could be useful is restricting the size of modules to a certain depth or using a method such as Dignum and Poli's operator equalization [35] to manage the size of modules. However, the size of modules discovered does not tell the complete story. To further understand the information being encapsulated by modules, Section 7.3 will examine the content of the discovered modules.

7.3 Module Content

To further explore the differences between the various module finding methods, this section examines the content of the modules discovered. This study of module content is approached by first examining the frequencies of different terminal symbols encapsulated into modules by the module identification operations. Next, the semantics of the modules is examined. These two examinations will give insight into the variety of modules found by the identification methods described in Chapter 6.

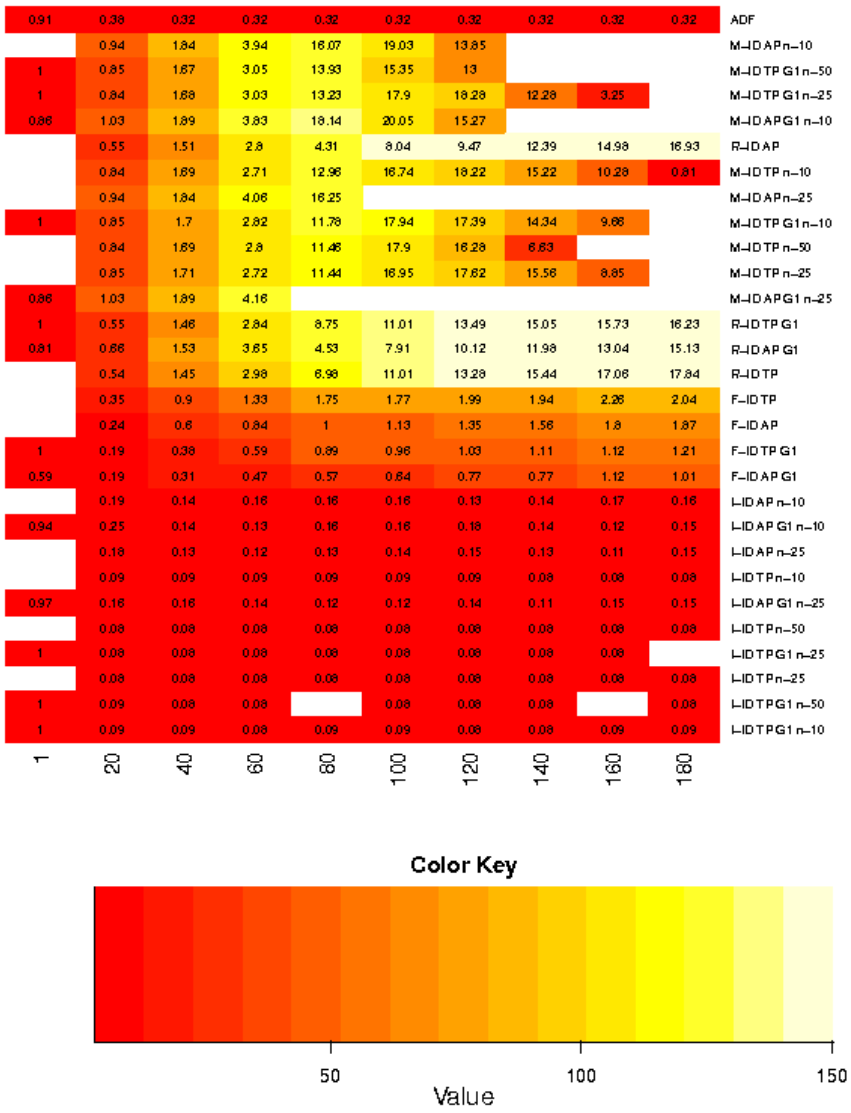


Figure 7.4: This figure shows the average depth of modules discovered by the various module identification setups on the 8×8 Lawn Mower problem.

7.3.1 Module Phenotypes

Here, the phenotypic elements that comprise discovered modules are examined. Looking at which symbols are used more or less often by modules gives another basis for differentiating the various module identification methods from each other. The benchmark problems used in this work each have known solutions. Examining which terminal symbols are encapsulated by modules will show if logical amounts of the more "important" symbols are being discovered as modules. For example, the target for the $x^5 - 2x^3 + x$ Symbolic Regression problem uses no division operators and many multiplication operators. However, it may be the case that some modules encapsulate division operators and few or no multiplication operators. Comparing the contents of modules in this way will show if some approaches encapsulate more or less useful module components. In the following sections, the phenotypic components of every module ever used in GE's grammar, in the case of the R-ID, I-ID, M-ID, and F-ID approaches, and every module created by ADFs are examined.

Santa Fe Ant Trail

The first set of modules to examine are those discovered on the Santa Fe Ant Trail problem shown in Figure 7.5. This figure shows that there is much similarity in terms of the proportion of terminal symbols used by modules from each of the identification approaches. Each module discovery method uses fewer `if(food ahead){...}else{...}` code blocks than other terminal symbols. Another feature that all the approaches in Figure 7.5 exhibit is the large proportion of `move` instructions encapsulated into modules. However, unlike all the other identification methods, the top two (*F-ID TP G1* and *F-ID AP G1*) have a slightly more even distribution of `move` and `right` instructions. This suggests that the best approaches still rely heavily on encapsulating `move` instructions (which is expected since many `move` instruction are needed for picking up food) but also use more `right` instructions to navigate the ant trail. This also suggests that for the Santa Fe Ant Trail, it

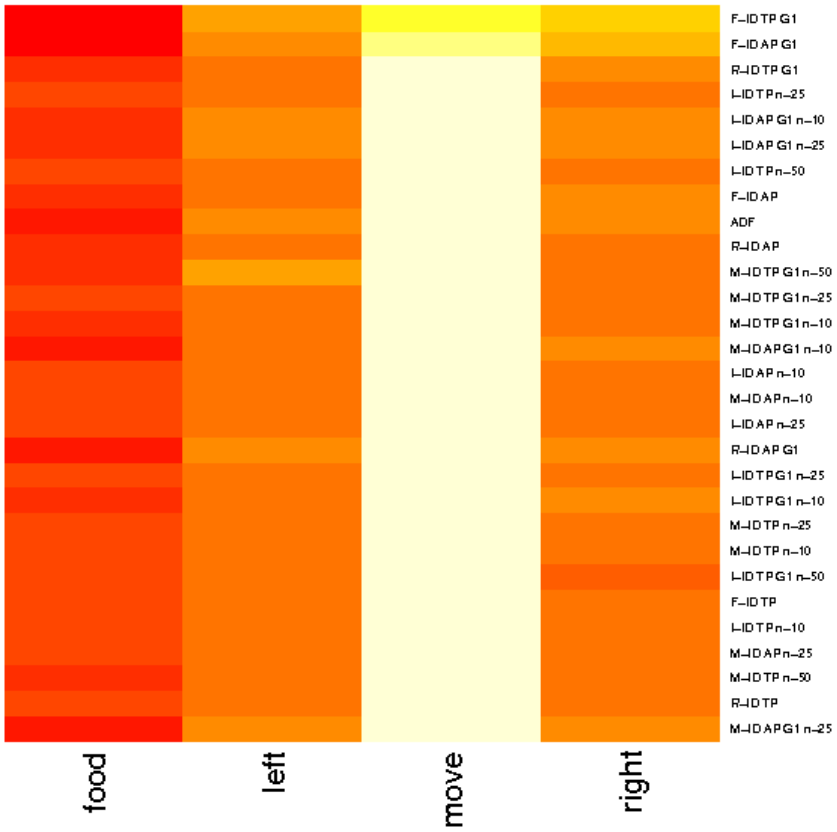


Figure 7.5: This figure shows the proportion of each terminal symbol the module identification approaches encapsulate into modules for the Santa Fe Ant Trail problem. Each row in the heatmap represents a module identification method and each column represents a terminal symbol. Darker red and orange colored cells denote smaller proportions, and yellow to pale yellow colors indicate larger proportions. The `left`, `right`, and `move` columns represent their respective any movement instructions. The `food` column denotes the `if(food ahead){...}-else{...}` logic statement.

may be beneficial to force a more even distribution of terminal symbols used in modules, similar to that of the (*F-ID TP G1* and *F-ID AP G1*) approaches.

Even 7 Parity

Next, Figure 7.6 shows the proportion of different terminal symbols used in modules on the Even 7 Parity problem. This figure shows that the `xor` and `not` operations are the most common among terminal symbols found in modules. For this problem, there does not appear to be any differences that separate the best performing approaches from the

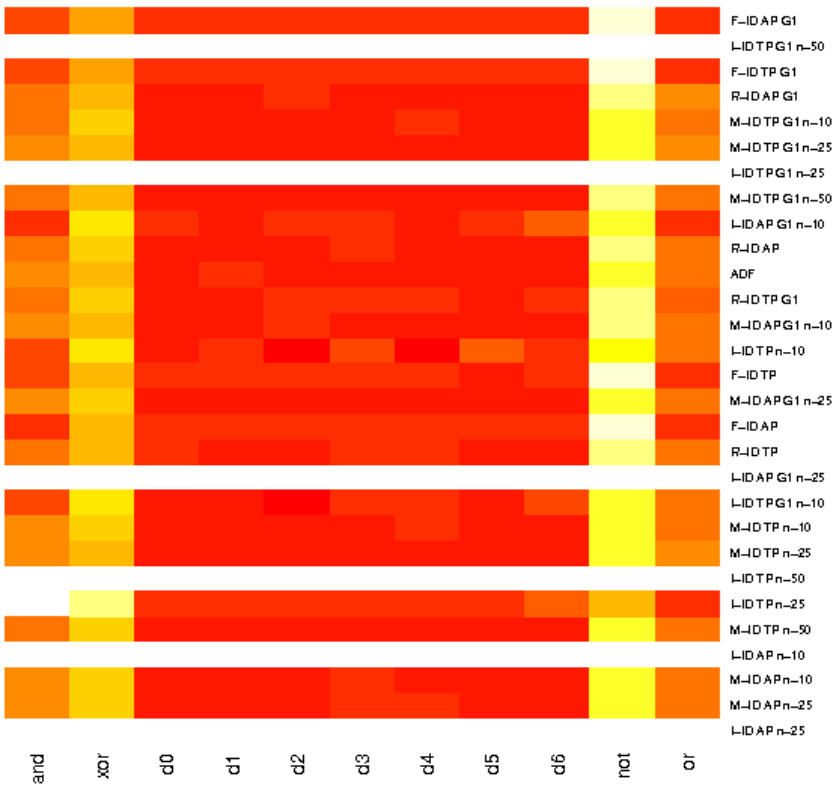


Figure 7.6: This figure shows the proportion of each terminal symbol the module identification approaches encapsulate into modules for the Even 7 Parity problem. Each row in the heatmap represents a module identification method and each column represents a terminal symbol. Darker red and orange colored cells denote smaller proportions, and yellow to pale yellow colors indicate larger proportions.

others in terms of the variety of terminal symbols found in the modules. This suggests that the key to the best module identification approaches' success can be found in the size, meaning, or usage of the modules.

$x^5 - 2x^3 + x$ Symbolic Regression

On the $x^5 - 2x^3 + x$ Symbolic Regression problem (Figure 7.7), there are some characteristics to mention that occur in terms of the proportions of the various terminal symbols encapsulated by modules from the various module discovery methods. The first feature of Figure 7.7 to notice is the block of the top performing module identification approaches

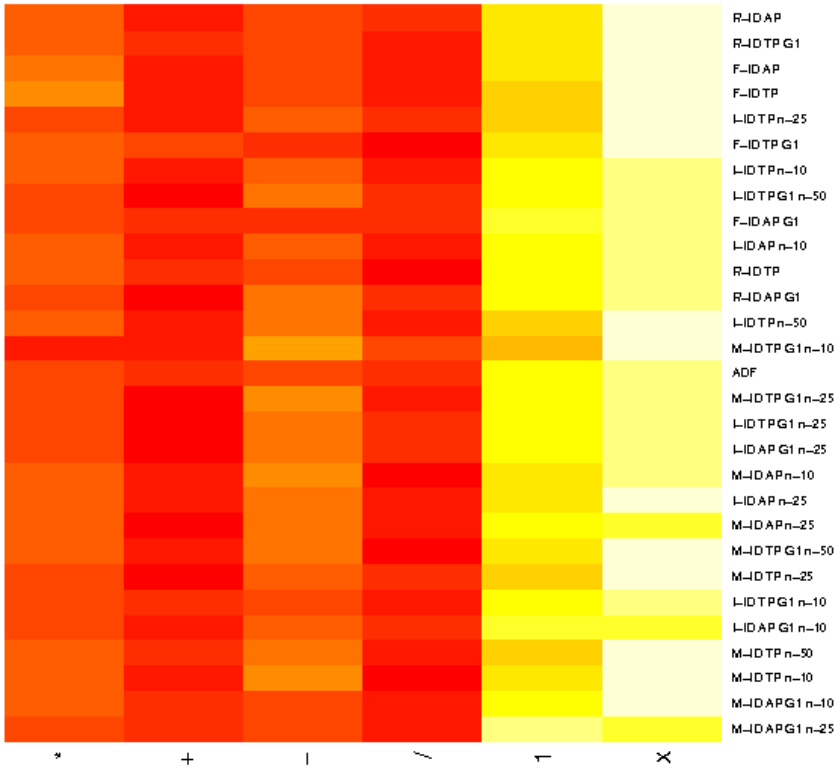


Figure 7.7: This figure shows the proportion of each terminal symbol the module identification approaches encapsulate into modules for the $x^5 - 2x^3 + x$ Symbolic Regression problem. Each row in the heatmap represents a module identification method and each column represents a terminal symbol. Darker red and orange colored cells denote smaller proportions, and yellow to pale yellow colors indicate larger proportions.

that use more x symbols than 1s. A reasonable explanation for this comes from the target, which is primarily composed of x operands and $*$ operators. Another interesting feature of Figure 7.7 is the amount of $-$ operators found in modules. The best performing approaches to discovering modules use some $-$ operators but not with the same frequency that they use $*$ operators. This suggests that the methods for finding modules that perform the best are able to encapsulate proportions of the terminal symbols which can aid in successfully reaching the target.

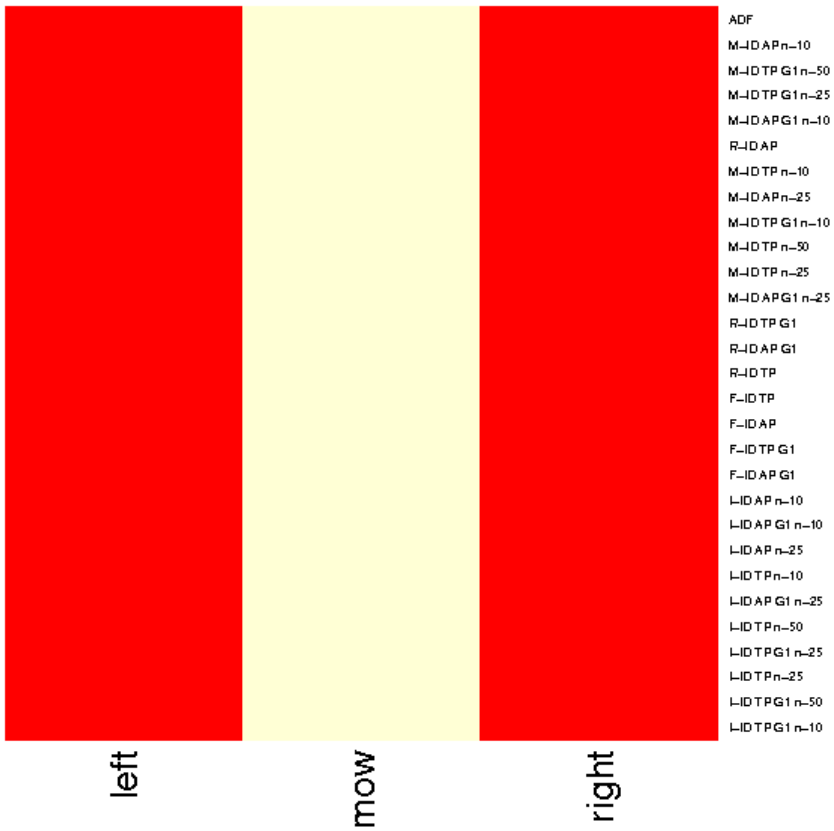


Figure 7.8: This figure shows the proportion of each terminal symbol the module identification approaches encapsulate into modules for the 8×8 Lawn Mower problem. Each row in the heatmap represents a module identification method and each column represents a terminal symbol. Darker red and orange colored cells denote smaller proportions, and yellow to pale yellow colors indicate larger proportions.

8×8 Lawn Mower

Similar to the Even 7 Parity problem in Section 7.3.1, the 8×8 Lawn Mower shows no differences in the proportions of terminal symbols that exist in modules (Figure 7.8. The most prominent terminal symbol used in modules is the `mow` symbol, which is unsurprising as it is the most important in terms of helping the lawn mower cover as much area as possible and achieving a better fitness value. Again, the differences in performance on this most likely due to the size, meaning, or usage of the modules.

Discussion

Examining the proportion of terminal symbols encapsulated by modules may uncover biases, or lack thereof, that are present in various module identification methods. However, Figures 7.5 - 7.7 show that there may be slight differences in terms of which terminal symbols are encapsulated as modules, but these differences are not usually large. This means most of module identification approaches are identifying modules with similar contents, and the cause of the differences in performance stem from another characteristic(s) of the modules.

7.3.2 Module Semantics

The next characteristic of modules to analyze is their meaning, or semantics. For the purpose of this analysis, the semantics of a module is defined as its fitness when evaluated on the same fitness function as the individuals in the population. Measuring the semantics of modules in this way gives insight into how potentially useful the discovered modules are and the diversity of behaviors exhibited by the modules.[1] But this is not the sole method for measuring semantics. Nguyen et al. [102] and McPhee et al. [93] both give different metrics for evaluating the semantics of sub-trees in GP. More recently, Moraglio et al. [96] and Krawiec and Pawlak [76] examine the geometry in addition to the semantics of individuals in GP. It is important to note that modules may have a good semantic value, but individuals must use them appropriately in order to take advantage of the modules. However, looking at the modules' semantics in this way does give a good idea of the diversity of performance exhibited by the modules. An expected result of this analysis would show approaches that discover modules with good semantic values when evaluated

[1]The use of semantics in this way somewhat contrasts the understanding of a module, which is meant to solve only part of a problem instead of the entire problem. However, for the purpose of this examination, the use of semantics in this manner is sufficient.

out of the context of individuals are also the best performing approaches during evolution. The data used to make Figures 7.9 - 7.12 can be found in Tables B.1 - B.4 in Appendix B.

Santa Fe Ant Trail

For the Santa Fe Ant Trail (Figure 7.9), the best performing approach to modularity (*F-ID TP G1*) did not discover modules that performed particularly well on average when they are removed from individuals. However, it did have the second largest standard error of the average semantics of modules discovered by this method, showing that the modules discovered by this approach have tend to be more diverse in terms of performance on the Santa Fe Ant Trail. One trend in the data in this figure is seen in the *I-ID* approaches. All the *I-ID* approaches have similar average semantic values, showing that on this benchmark problem the *I-ID* methods tend to identify modules with more similar behavior than modules discovered by other methods. A final characteristic to discuss concerning Figure 7.9 is the behavior of the ADFs discovered. The ADFs had the largest average semantic value and smallest standard error of all the approaches examined.

Even 7 Parity

On the Even 7 Parity problem, the best approach is *F-ID AP G1*. Figure 7.10 shows that this method identifies modules with relatively large average semantic values. Most approaches also have a very small standard error. One approach that deserves attention on this problem is the *I-ID TP n-25* approach. This set of parameters only finds one module throughout the duration of all of its combined runs. But this sole module has a average semantic value of 0, meaning it solves the Even 7 Parity problem. Similar to the Santa Fe Ant Trail results (Section 7.3.2), ADFs have one of the largest average semantic values. They also have the smallest standard error.

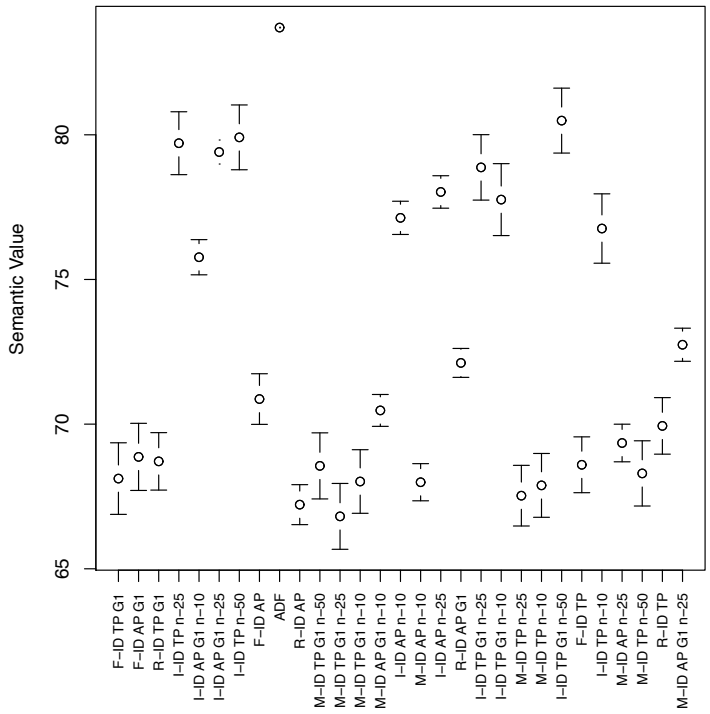


Figure 7.9: This figure shows the average semantic value and standard error of all the modules ever discovered on the Santa Fe Ant Trail problem. The semantic value is calculated by evaluating each module on the fitness function as if it was a stand-alone individual.

$x^5 - 2x^3 + x$ Symbolic Regression

Figure 7.11 shows the average semantic values for module identification methods on the $x^5 - 2x^3 + x$ Symbolic Regression problem. On this problem, the best performing module discovery method (*R-ID AP*) has one of the largest average semantic values and a low entropy value. There are also a number of methods for identifying modules with small average semantic values as well as lower and higher standard errors. On this problem, ADFs have the largest average semantic value and one of the largest standard errors.

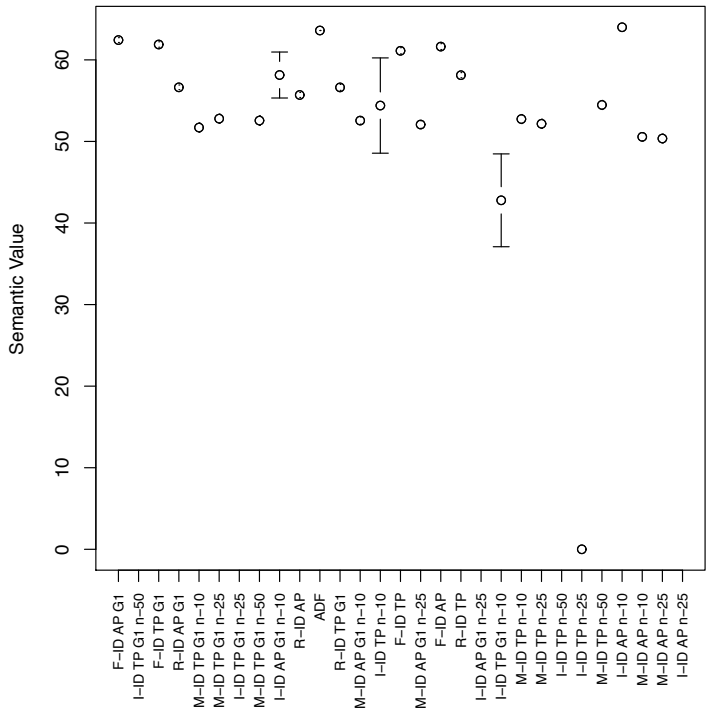


Figure 7.10: This figure shows the average semantic value and standard error of all the modules ever discovered on the Even 7 Parity problem. The semantic value is calculated by evaluating each module on the fitness function as if it was a stand-alone individual.

8×8 Lawn Mower

The 8×8 Lawn Mower problem (Figure 7.12) shows different trends than the previous problems. Here, Figure 7.12 shows the best performing approaches all identify modules with small average semantic values with similar standard error values. The exception to this is the best performing approach, ADFs. It has a significantly larger average semantic value than the other top-performing approaches. Aside from ADFs, as the module identification approaches perform worse, the modules identified by those approaches correspondingly have larger average semantic values.

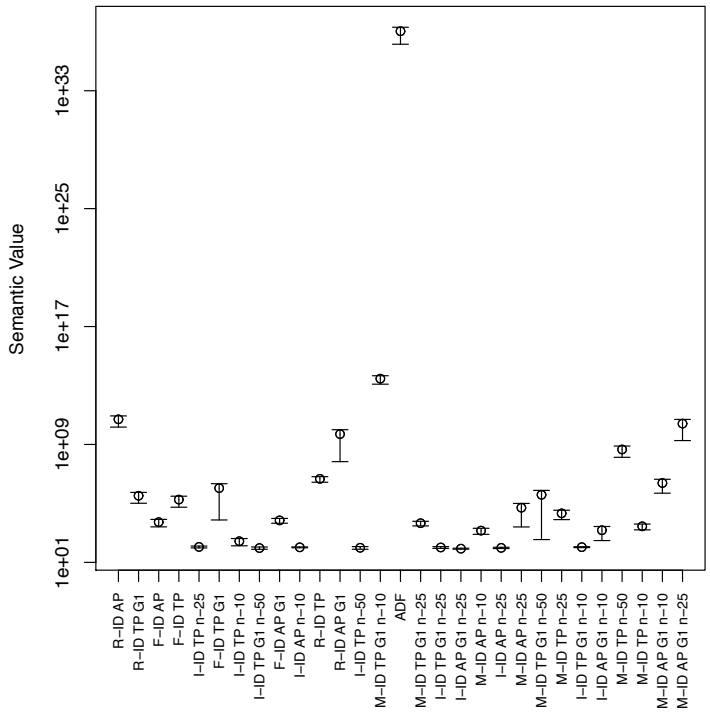


Figure 7.11: This figure shows the average semantic value and standard error of all the modules ever discovered on the x^5-2x^3+x Symbolic Regression problem. The semantic value is calculated by evaluating each module on the fitness function as if it was a stand-alone individual. The error bars on this graph appear skewed due to the log-scale y-axis.

Discussion

The data from the above sections show that on the Santa Fe Ant Trail, Even 7 Parity, and $x^5 - 2x^3 + x$ Symbolic Regression problems the average semantic value of modules when evaluated as stand-alone individuals is not necessarily a good indicator of their utility in improving GE's performance. On the other hand, in all cases except for ADFS, modules found on the 8×8 Lawn Mower problem may be evaluated on their own to estimate how useful they may be when made available to the population. ADFs tend to exhibit the most constant behavior across problems. They consistently have one of the largest average semantic values and smallest standard error, except on the x^5-2x^3+x Symbolic Regression problem, where it has the largest standard error as well as the largest average semantic

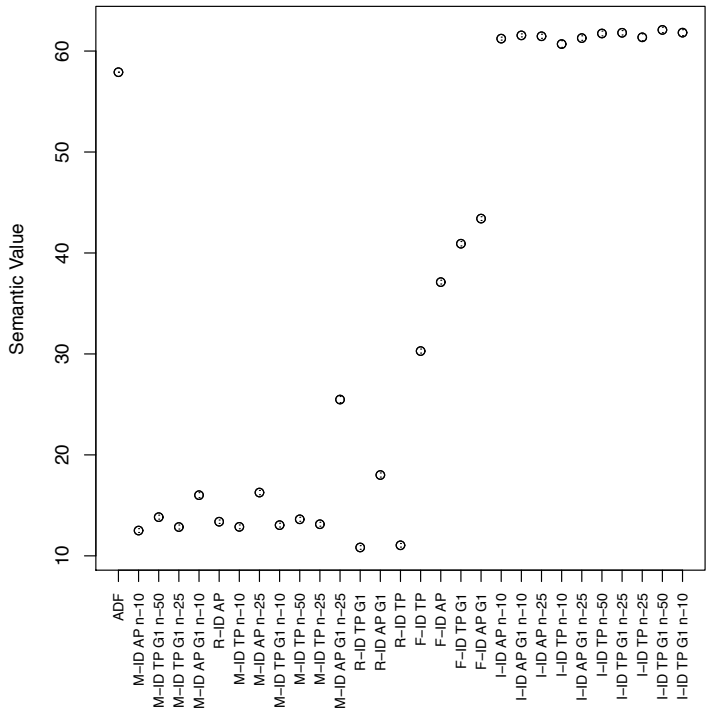


Figure 7.12: This figure shows the average semantic value and standard error of all the modules ever discovered on the 8×8 Lawn Mower problem. The semantic value is calculated by evaluating each module on the fitness function as if it was a stand-alone individual.

value.

On the Santa Fe Ant Trail, $x^5 - 2x^3 + x$ Symbolic Regression, and 8×8 Lawn Mower problems, the best performing methods have some of the largest standard errors for the semantic values exhibited by the modules they discover. These results suggest that when identifying modules, it may be beneficial to encourage or bias towards discovering modules that may have a variety of semantics. The exception for this is the Even 7 Parity problem, where the best performing approach to modularity has one of the smallest standard errors for the semantic values of the modules.

While the data in this section gives some insight into the how the semantics of modules may be used, and taking into account the data presented on module size (Section 7.2) and

phenotypic content (Section 7.3.1), it is clear that the semantics of a module should not be the only consideration when incorporating a bias for which modules should be considered for use in the population. Moreover, even if high quality modules are discovered, they must be used by individuals. The following section examines how the population uses modules over the course of evolution.

7.4 Module Usage

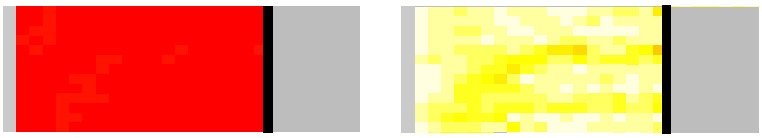
The previous sections of this chapter (Sections 7.2 and 7.3) discuss characteristics of the modules themselves. A knowledge of these characteristics may be useful, but it is also important to understand how modules are used by the population. This is the focus of this section. More specifically, the percentage of individuals in the population that use a given module and the average fitness of individuals using that module is examined. Looking at how modules are used in the population helps to determine if modules are being used in a "sensible" manner, meaning modules get used by fit, as well as unfit, individuals, and modules are used by many individuals in the population. To avoid repeatedly discussing similar and unremarkable results, this section covers only the differences between the best and worst approaches to identifying modules and compare and contrast how modules discovered by these approaches are used. One expectation of the outcome of this study is to point out the different usage patterns that exist between the best and worst performing modular methods. The analysis in this section is based on the work of Swafford et al. [155]. The data below only shows modules that appear in 30% of the population or more in at least one generation. This is done because there are too many modules in total to look at, and the most interesting features of the following data comes from the most used modules.

For the remainder of this section, heatmaps are used to present how modules are used by individuals over the course of their evolutionary runs. In Figures 7.15– 7.20, each row

of the heatmap represents a module and each column represents a generation. In figures labeled "Usage," red or dark orange colored cells mean that few individuals are using a module at that generation. White or yellow cells mean all or many individuals are using a module. Gray colored cells denote generations when a module is not used at all. In figures labeled "Fitness," the red and dark orange cells indicate that the average fitness of individuals using modules is a smaller value (small values signify better fitness). The white and yellow cells mean the average fitness of individuals using a module is large, indicating a worse fitness. Like the "Usage" heatmaps, gray cells show generations where the module is not used. The data for these graphs have been normalized between 0 and 1 by dividing each value by the maximum value in the data set. For all graphs, the black vertical lines indicate generations in which module identification and replacement takes place. The data presented in the following sections only shows modules that are used by 10% or more of the population at any generation.

7.4.1 Description of Heatmap Features

To assist the reader in more quickly recognizing the notable features of the data presented in Sections 7.4.2 - 7.4.5, a selection of these features are presented here with an explanation of what that particular feature signifies.



(a) Usage (b) Fitness

Figure 7.13: This figure shows an example of modules being used by a small number of individuals (Figure 7.13(a)) with a large average fitness (Figure 7.13(b)).

The first of these features is given in Figure 7.13. Here, Figure 7.13(a) shows a block of modules that are used by a small percentage of individuals. Correspondingly, the yellow and

white colors of Figure 7.13(b) denote the average fitness of individuals using these modules is a large value. The black vertical line denotes a generation where module identification and module replacement occurs. Cells after the black line are gray, meaning the modules are no longer used by individuals. Combining the information shown by these two slices of heatmaps, it can be inferred that the block of modules shown is used by a small amount of individuals with poor fitness values until the modules are no longer available for use in the population due to the module replacement operation.

The next feature to discuss is shown in Figure 7.14. The red and dark orange cells on the left of this figure show modules being used by a small percentage of the population. The transition from the red and orange cells to pale yellow and white cells denote the modules being used by a larger number of individuals.

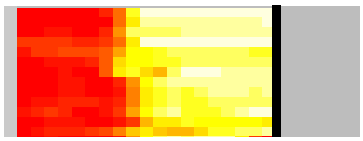
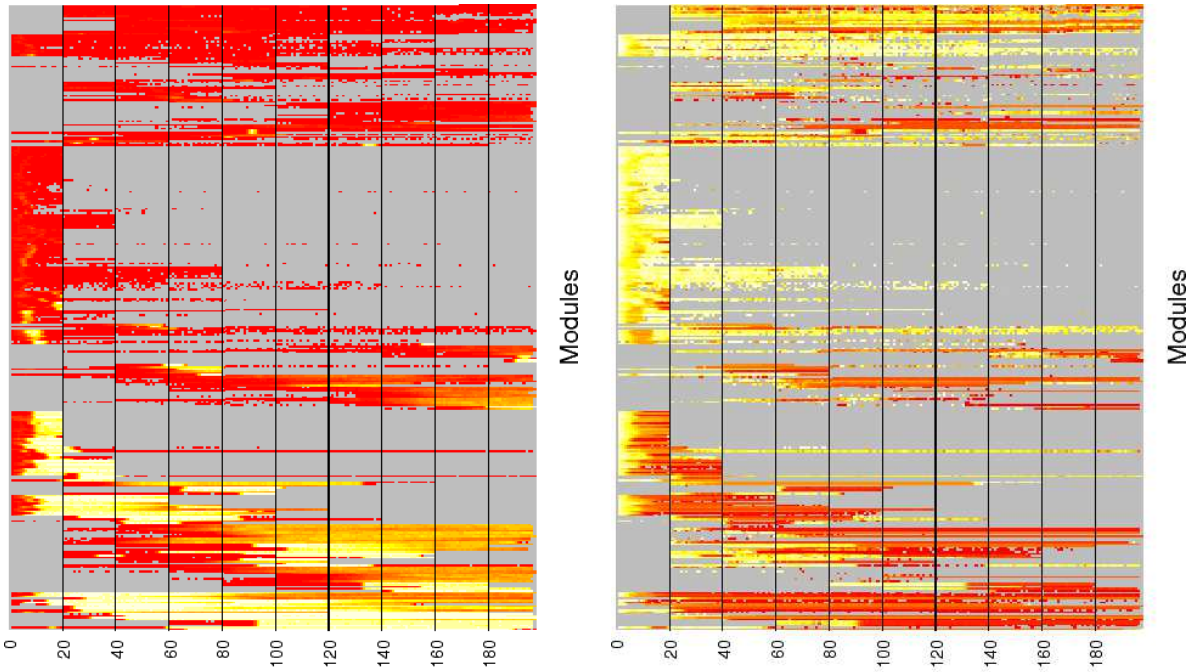


Figure 7.14: This figure shows an example of modules being used initially by a small amount of individuals and over time being used by a large amount of individuals.

7.4.2 Santa Fe Ant Trail

The best approach to identifying modules for the Santa Fe Ant Trail is *F-ID TP G1* (Figure 7.15). One of the first characteristics to discuss is the large amount of modules that are only used by a small percentage of individuals in the population. The large amount of red cells at the top of Figure 7.15(a) shows that many modules are used by small amounts of the population for many generations. When considering this in conjunction with the corresponding cells in Figure 7.15(b), it can be seem that some of the modules, despite the low frequency of their use, are used by individuals with small fitness values (Recall that better individuals have smaller fitness values), as well as individuals with large fitness

values. There are also numerous modules which are initially used by few individuals and as the generations progress are used by a large percentage of the population. These modules can be seen in the bottom halves of Figures 7.15(a) and 7.15(b). Another interesting feature of these figures is that many of the modules discovered in the first generation are later replaced by modules with much longer lifespans.

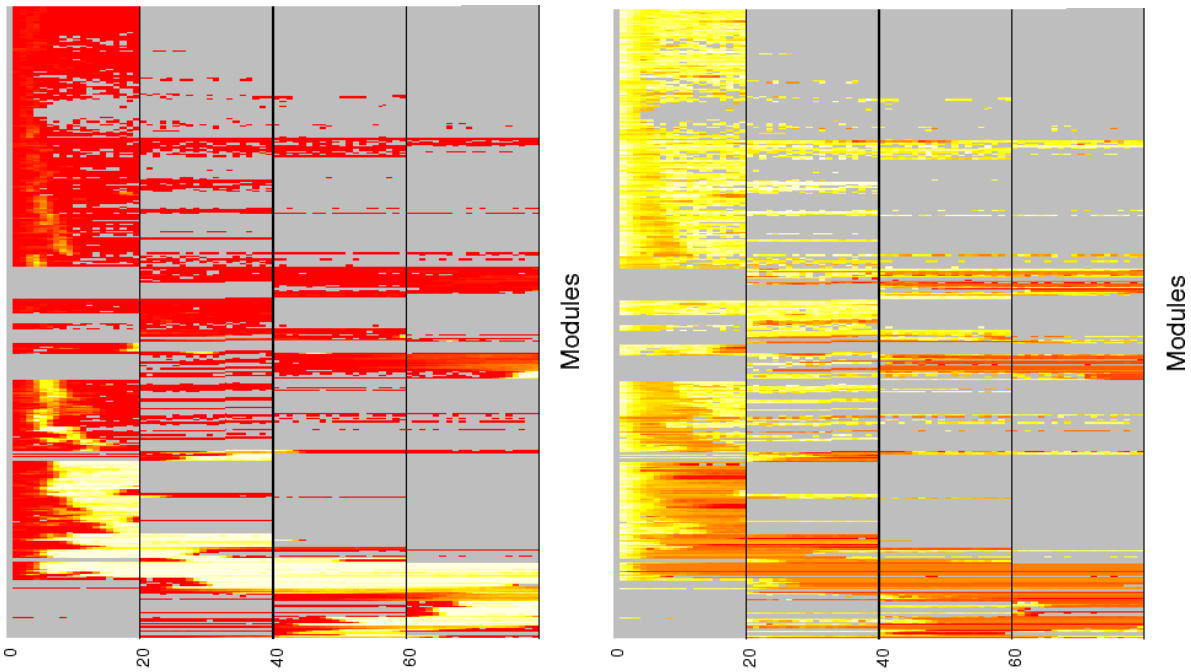


(a) Usage (b) Fitness

Figure 7.15: This figure shows the usage of modules in the population (Figure 7.15(a)) and the average fitness of individuals using modules (Figure 7.15(b)) on the Santa Fe Ant Trail. Modules are identified using *F-ID TP G1* method (the best performing approach on this problem).

The worst performing approach on this problem is *M-ID AP G1 n*–25, shown in Figure 7.16. It has similar characteristics as Figure 7.15 in terms of how often modules are used, how fit the individuals that use them are, and how long they last in the population. The largest difference between the two approaches is the number of generations completed. Recall this is discussed in more detail in Chapter 6. The data from these figures suggest that on the Santa Fe Ant Trail, the difference in the performance of the best and worst

approaches to modularity is not due to how the modules are used. The most likely sources of the difference in performance are the parameters that determine how often modules are identified and how many fitness evaluations are used to discover modules.

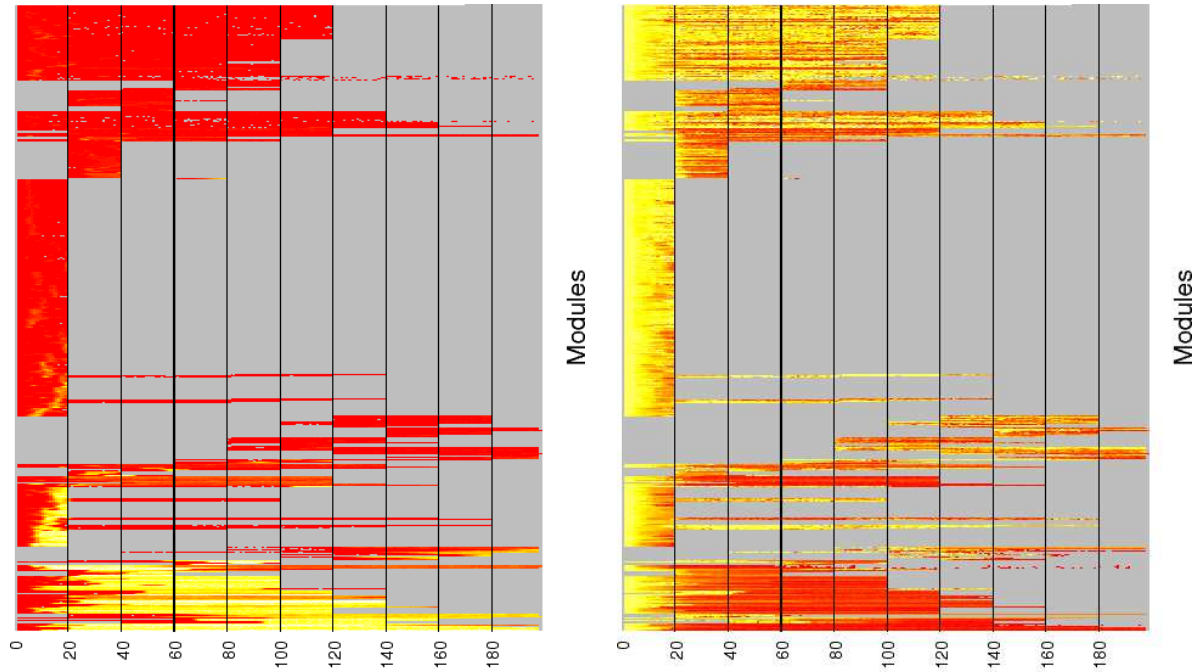


(a) Usage (b) Fitness

Figure 7.16: This figure shows the usage of modules in the population (Figure 7.16(a)) and the average fitness of individuals using modules (Figure 7.16(b)) on the Santa Fe Ant Trail. Modules are identified using *M-ID AP G1* n–25 method (the worst performing approach on this problem).

7.4.3 Even 7 Parity

The best approach to identifying modules on the Even 7 Parity problem (*F-ID AP G1*), given in Figure 7.17, shows similar characteristics to those seen in the best approach on the Santa Fe Ant Trail (Section 7.4.2). Many modules identified in the first generation are replaced at the next module identification/replacement occurrence by modules with longer lifetimes. There are also numerous modules which are used for many generations only by



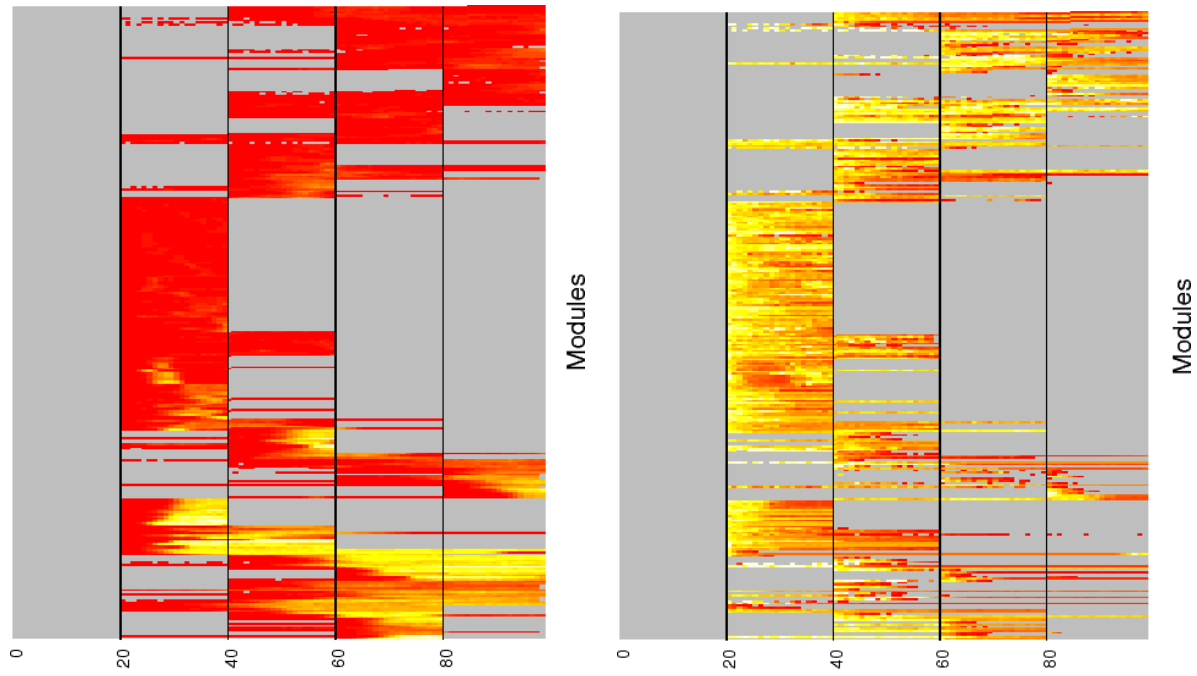
(a) Usage (b) Fitness

Figure 7.17: This figure shows the usage of modules in the population (Figure 7.17(a)) and the average fitness of individuals using modules (Figure 7.17(b)) on the Even 7 Parity. Modules are identified using *F-ID AP G1* method (the best performing approach on this problem).

a small percentage of the population. The worst approach for identifying modules on the Even 7 Parity problem is *M-ID AP n*–25. Modules discovered by this approach show much of the same behavior as modules found by the best module identification method (*F-ID AP G1*) for this problem. Like the difference in performance between the best and worst approaches for finding modules on the Santa Fe Ant Trail problem, the most probable cause of the differences in performance on the Even 7 Parity problem are the parameters which determine how often modules are identified.

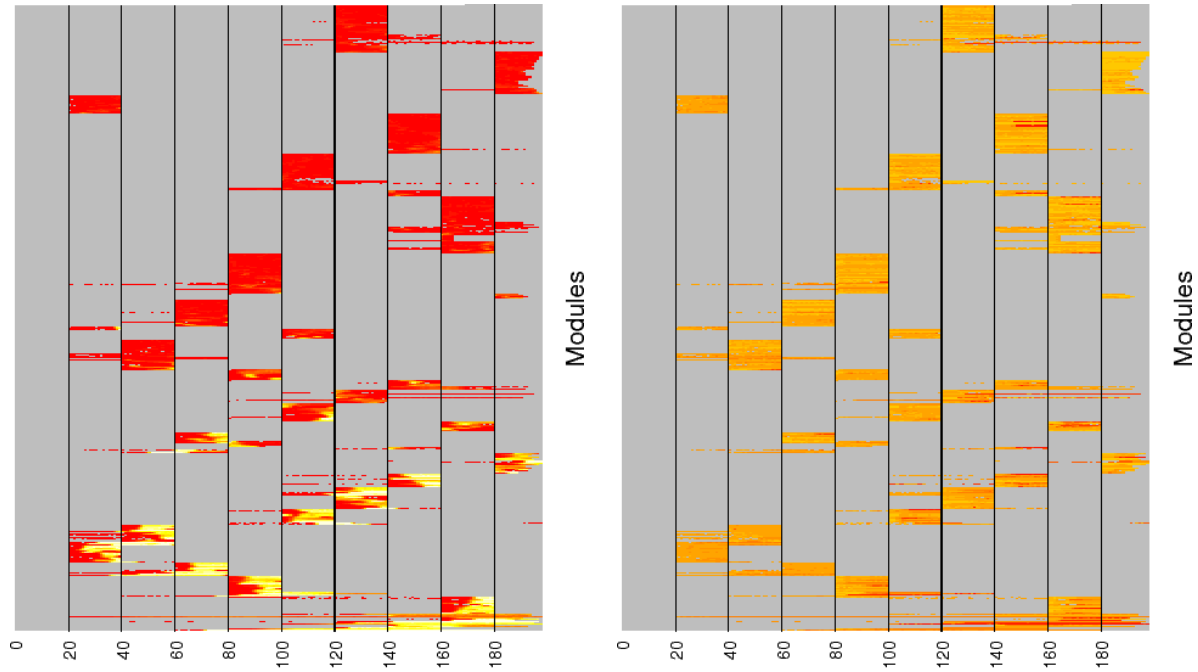
7.4.4 $x^5 - 2x^3 + x$ Symbolic Regression

The best performing approach on the $x^5 - 2x^3 + x$ Symbolic Regression problem, *R-ID AP*, can be seen in Figure 7.19. This method for identifying modules cycles modules in and out



(a) Usage (b) Fitness

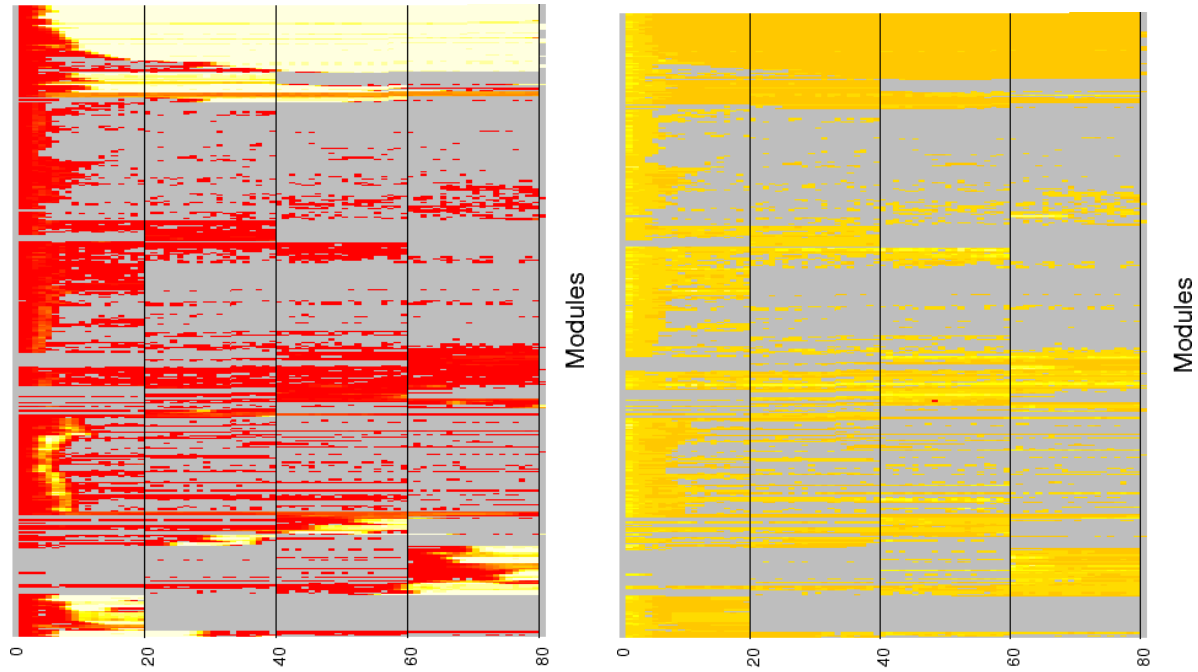
Figure 7.18: This figure shows the usage of modules in the population (Figure 7.17(a)) and the average fitness of individuals using modules (Figure 7.18(b)) on the Even 7 Parity. Modules are identified using *M-ID AP* n–25 method (the worst performing approach on this problem).



(a) Usage (b) Fitness

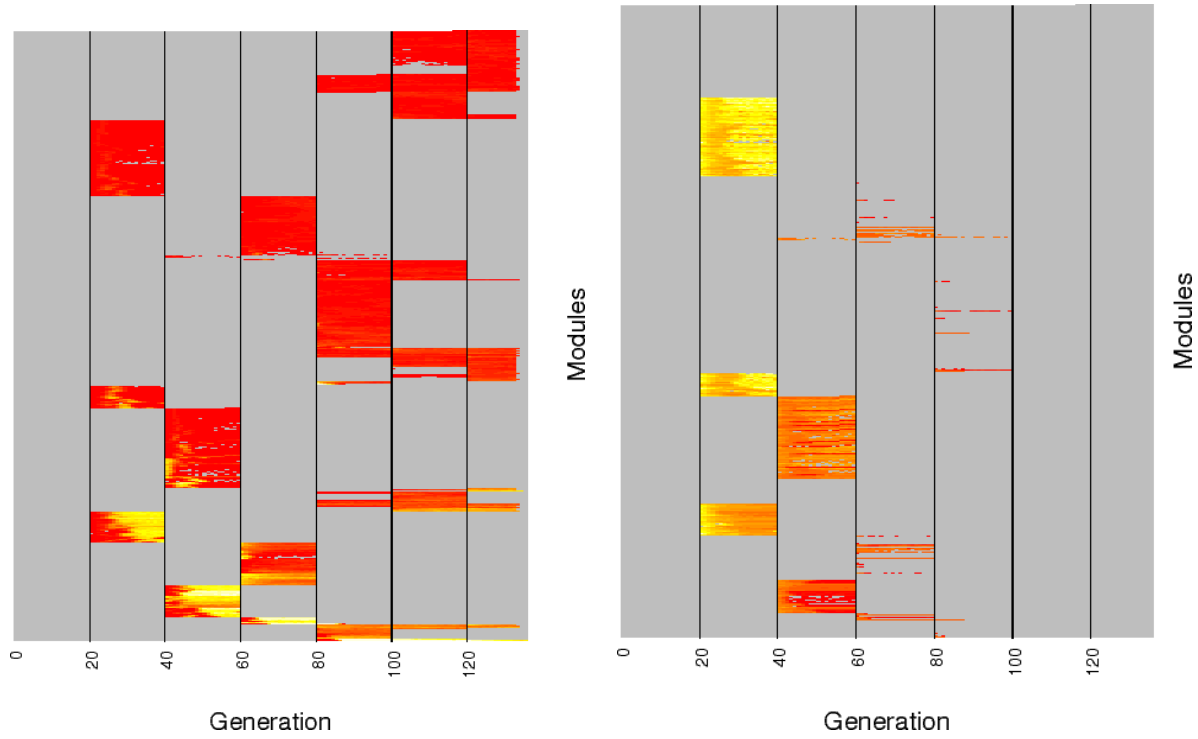
Figure 7.19: This figure shows the usage of modules in the population (Figure 7.19(a)) and the average fitness of individuals using modules (Figure 7.19(b)) on the $x^5 - 2x^3 + x$ Symbolic Regression. Modules are identified using *R-ID AP* method (the best performing approach on this problem).

of the population in a similar manner to the best performing approach to the 8×8 Lawn Mower problem Figure 7.21. Using the worst performing module identification approach given in Figure 7.20, many modules last 60–80 generations. There is very little module replacement, meaning that once a module is discovered and put into GE's grammar, it is unlikely they will be replaced by a new module, suggesting that the *M-ID AP G1 n*–25 method is unable to discover quality modules to replace the existing ones. Another characteristic of the graphs in Figure 7.20 is that many modules only get used by highly fit individuals. This suggests that crossover and mutation operations are not spreading these modules to new individuals.



(a) Usage (b) Fitness

Figure 7.20: This figure shows the usage of modules in the population (Figure 7.20(a)) and the average fitness of individuals using modules (Figure 7.20(b)) on the $x^5 - 2x^3 + x$ Symbolic Regression. Modules are identified using *M-ID AP G1* n–25 method (the worst performing approach on this problem).



(a) Usage (b) Fitness

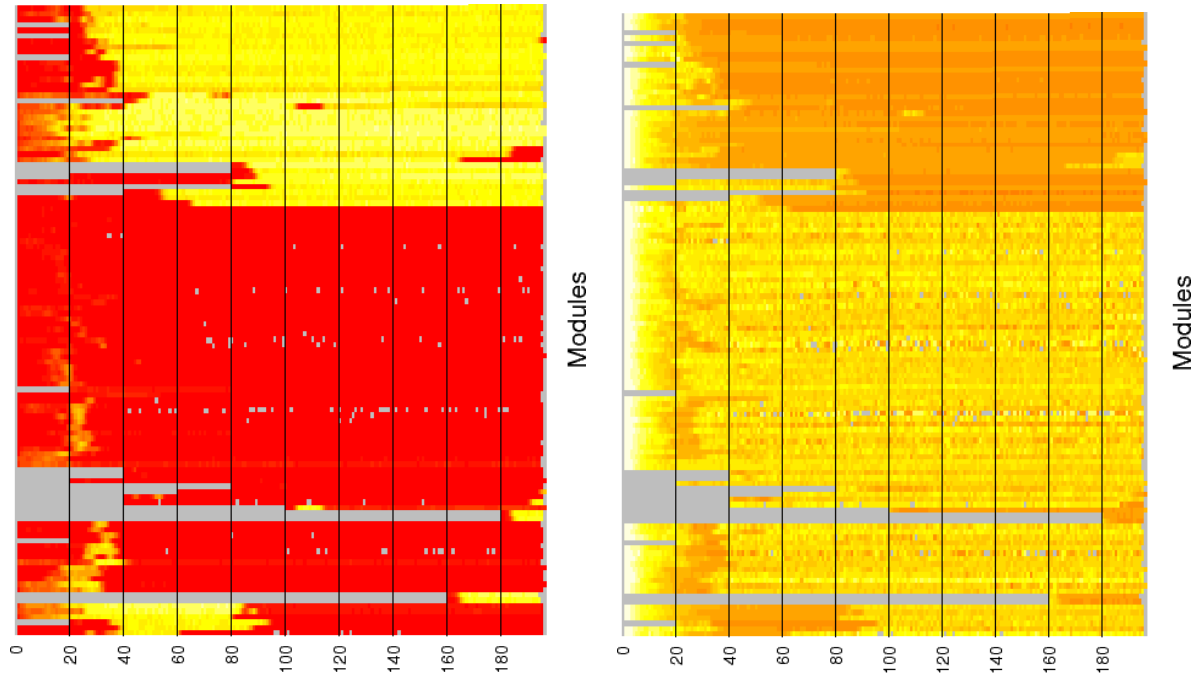
Figure 7.21: This figure shows the usage of modules in the population (Figure 7.21(a)) and the average fitness of individuals using modules (Figure 7.21(b)) on the 8×8 Lawn Mower. Modules are identified using *M-ID AP* n–10 method (the best performing approach on this problem).

7.4.5 8×8 Lawn Mower

On the 8×8 Lawn Mower problem, the best performing module ID method is *M-ID AP* n–10. Using this approach, there is a large amount module of turnover (Figure 7.21). Few modules last longer than the 20 generation module identification step. However, there begins to be less module replacement in the later generations. The worst performing approach is *I-ID TP G1* n–10 (Figure 7.22). This method exhibits no replacement of modules at all in the modules graphed. There are two possible causes for this:

1. not many modules are discovered and replacement is never needed,

2. many modules are discovered but the combination of the module replacement operation and I-ID method is not suitable for this problem.



(a) Usage (b) Fitness

Figure 7.22: This figure shows the usage of modules in the population (Figure 7.22(a)) and the average fitness of individuals using modules (Figure 7.22(b)) on the 8×8 Lawn Mower. Modules are identified using *I-ID TP G1* n–10 method (the worst performing approach on this problem).

Is it likely that the relatively poor performance of this module identification approach is a results from the lack of module replacement or discovery.

7.4.6 Discussion

The analysis of module usage does not tell much about what makes the modules themselves good or bad, but it gives insight into desirable behaviors for modules to exhibit in the population. After examining the behavior of the best performing approaches for identifying modules, one conclusion is that there should be a reasonable amount of replacement of the modules made available to the population. When some portion of the modules is replaced, evolution is finding more information in the population that is currently, and may be

in the future, valuable as a module. It is also important for modules to be propagated throughout the individuals as well. If modules are not used, they are unable to serve their purpose of improving search performance. A further study, out of the scope of this thesis, to investigate the effects of crossover and mutation on module propagation may prove valuable to understand how modules are spread through the population. The differences between module usage in the Santa Fe Ant Trail and Even 7 Parity problems and the 8×8 Lawn Mower and $x^5 - 2x^3 + x$ Symbolic Regression problems also shows problem dependence in terms of how modules can best be used.

7.5 Summary

This chapter analyzes the size, contents, semantics, and usage of modules on four benchmark problems. First, the characteristics of modules, such as depth, phenotypic contents, and semantics, discovered by the various module identification methods are examined. Next, the usage of modules discovered by the best and worst approach for finding modules on each problem is discussed by showing how modules are used by different percentages of the population over time as well as the average fitness of individuals that use them.

The results of the analysis presented in this chapter can be related to two questions proposed in Section 1.2.1:

Research Question 3 - What are the characteristics of the modules themselves that makes them useful? At the various stages in evolution, a particular size of module is likely to be more helpful than others. This size may vary across problems. A mechanism to discover modules of the appropriate size at a given generation may be beneficial. All of the approaches for identifying modules under examination find relatively similar proportions of the terminal symbols that comprise modules. However, specifically on the $x^5 - 2x^3 + x$ Symbolic Regression problem (Section 7.3.1), modules may often

contain symbols that are not very useful. This raises the issue of bloat in the sense that the modules being found may not be particularly useful. Studying the semantics of the discovered modules revealed that ensuring modules adhere to a certain fitness distribution in terms of their behavior may be a valuable feature to add to the module replacement operation.

Research Question 4 - Are modules discovered by various identification methods used differently by evolution? Looking at how modules are used in the population shows that the best approaches on two of the benchmark problems are able to easily replace modules. This suggests that the ability to replace old modules with newer and more useful modules is important for improving GE's performance. The worst performing approaches to discovering modules often had difficulty discovering and replacing existing modules.

The results of these studies point towards a number of features that should be taken into consideration when designing a module identification operation. The experiments carried out for this work placed no restrictions on the size, contents, or semantics of modules. Based on the results shown in this chapter, the development of future module identification methods should attempt to determine what sizes, contents, and semantic values of modules are useful and use this information to identify new modules with even greater utility. Another conclusion from the results presented in this chapter show that it is sometimes difficult for old modules to be replaced with newer modules. A potentially beneficial vein for future work could be to ensure that a reasonable amount of modules are being replaced with newer modules. Old modules could be removed based on how many individuals use then, the fitness of individuals using them, the number of generations they have been available for use, or any combination of these values.

Part IV

Conclusions and Future Work

Chapter 8

Conclusions and Questions Raised

After examining various methods for incorporating modules into GE's evolving population and different approaches for identifying modules, this chapter presents the conclusions of this thesis. First, a summary of the conclusions taken from this work is given in Section 8.1. Next, Section 8.2 covers the limitations of this thesis. Finally, Section 8.3 presents opportunities for future work based on the findings of this research.

8.1 Summary of Thesis

The aim of this thesis is to determine the most important questions that should be considered when incorporating modularity into GE. It explores different approaches to modularity in GE, with the ultimate goal of examining how these approaches alter GE's performance. Understanding how the characteristics of the approach to modularity affect GE's performance will allow for the development of more effective mechanisms for integrating modularity into, not just GE, but other EAs as well. To accomplish this goal of understanding modularity in GE, a number of questions are asked:

Module Identification

Research Question 1: *Should modules be identified using a fitness-based or a usage-based approach?*

Research Question 2: *How much computational power is reasonable to use in identifying modules?*

Research Question 3: *What are the characteristics of the modules themselves that makes them useful?*

Research Question 4: *Are modules discovered by various identification methods used differently by evolution?*

Module Usage

Research Question 5: *How should modules be made available to GE's population during evolution?*

Research Question 6: *Does updating the set of modules available to individuals more or less frequently change the performance of GE?*

Research Question 7: *Does the number of modules made available to GE during evolution affect GE's performance?*

The first experimental chapter of this thesis, Chapter 4, introduces a novel method for identifying modules. It then compares various methods adding modules to GE's grammar during an evolutionary run. It also compares variations for how frequently modules are added to GE's grammar and how many modules are added to GE's grammar per grammar modification step. The results of Chapter 4 show that modules should be added to GE's grammar in a way that causes minimal disturbance in the fitness of the population. It

also shows that there should be a balance in how often modules should be identified and how many modules should be added to GE's grammar when it is modified. Discovering modules too frequently under the **remap** approach causes disruptions in performance, uses computational resources, and does not allow evolution enough time to use the modules it has previously identified before they are replaced. Adding too many modules to GE's grammar dilutes the possibilities for individuals to create new information from the original grammar or to use previously discovered modules.

The following experimental chapter, Chapter 5, introduces the genotype repair operator which is used to ensure the addition of modules into GE's grammar does not present any adverse effects in terms of the fitness of the population. In addition to the genotype repair operator, a module expansion operator is also introduced. The results from Chapter 5 show that both the genotype repair and module expansion operations give a performance increase over the previous method of simply adding modules to GE's grammar.

Next, Chapter 6 defines four methods for identifying modules in GE. It then covers various parameters and variations of these operators in an attempt to improve their performance and understand what features are most desirable in a module identification method. The particular parameters examined include determining which portion of the population from which modules should be selected, how many fitness evaluation should be used to identify modules, the level of rigor of the module identification methods, and if modules should be identified after the initial generation of evolution. The results of this chapter show that there is no single-best approach for identifying modules. One general trend in the results was that the best performing approaches tended to use few or no fitness evaluations and identified modules after the initial generation.

The final experimental chapter, Chapter 7, goes on to analyze the features of the modules themselves. It shows how different sizes of modules are discovered at different generations of evolution and suggests that discovering modules of the appropriate size at

a given generation may be beneficial. It also reports how all the approaches for identifying modules find modules with similar contents. A third conclusion from Chapter 7 is that maintaining a certain distribution of fitness values amongst modules may also be beneficial. The final result from Chapter 7 shows how modules are spread throughout, added to, and removed from the population during evolution.

8.1.1 Contributions

Aside from the literature review and compilation of approaches to modularity in GP, the contributions of this thesis fall under two categories: integrating modules into the population and identifying modules. The following contributions address the questions presented earlier in this section.

Module Identification

Development of new module identification methods: To answer Question 8.1, two new novel module identification operators were developed and presented in Section 6.4.4. A comparison of the new operators to random and frequency-based module identification operators shows that operators which use less fitness evaluations tend to perform better.

Comparison of module identification approaches and their parameters: Section 6.4 answers Question 8.1. In that section, many variations of parameters for the modules identification methods are examined, showing that any method of reducing the fitness evaluations needed to identify modules is beneficial. Identifying modules immediately after the first generation may also be useful.

Analysis of module characteristics: Next, Question 8.1 is answered in Sections 7.2 and 7.3. These sections show that the best performing approaches to module discovery

find modules within certain size and fitness diversity ranges.

Analysis of module usage during evolution: The final question, Question 8.1, is answered in Section 7.4. The results of Section 7.4 show that the best approaches for identifying modules are able to more easily replace older modules with newer modules.

Incorporating Modules into GE

Analysis of methods for modifying GE's grammar: In order to answer Question 8.1, an examination of two mechanisms for incorporating modules into GE's grammar are examined in Section 4.4.1. This examination shows that using a module library production in the grammar allows modules to be added to GE's grammar while still allowing a reasonable probability for individuals to pick productions from the original grammar during their genotype-to-phenotype mapping process.

Analysis of the frequency at which the grammar is modified: Question 8.1 is answered in Section 4.4.2, where it is shown that allowing GE more generations before modifying its grammar gives a performance increase over modifying the grammar more frequently.

Analysis of the size of grammar modification: The answer to Question 8.1 also comes from Section 4.4.1. The results in this section show that keeping every module discovered often lead to poor performance compared to keeping a smaller selection of modules

Genotype repair operator: Although there was no initial research question which the genotype repair operator addressed, it has been a useful tool in enhancing GE's performance. This operator ensures modifying GE's grammar causes no change in the phenotypes of individuals after the grammar has changed during evolution.

8.2 Limitations of Thesis

As is mentioned in Chapter 1, the possibilities for work in regards to modularity and GP and/or GE are vast, and this thesis is only able to cover a limited amount of these avenues of research.

One of the first limitations is the set of parameters used in the experiments presented in Chapters 4-7. Varying basic EA parameters such as population size, individual size, crossover rate, mutation rate, etc., will change what individuals are present in the population and what modules can be selected from them. However, only a single set of parameters was used for these settings. Along these lines, only a single grammar was used for each benchmark problem. However, Nicolau [103] and Harper [53] show how grammars capable of expressing the same language, but with different numbers and layouts of terminal and non-terminal symbols, can have different performance in GE. No attempt was made to use any grammars other than those given in Chapter 2. This leaves a potential to research of new grammar designs to be used by GE with modules for future work.

Also mentioned in Chapter 1 was the number of parameters needed by the modularity methods themselves. An attempt was made to test these values within a reasonable range, but there are still many variations of these parameters which were unable to be examined. Making these parameters self-adaptive may prove beneficial in future work. Additionally, there are many other variations of the operators themselves which were unable to be examined. In particular, only one module replacement strategy was reported. In previous work, there has been a number of methods to update the set of modules available for use during evolution. No attempt was made to examine and compare these approaches to module replacement method currently in use.

Another limitation is the representation of a module in this thesis. As stated in Chapter 1, for this thesis, the definition of a module is **an encapsulated sub-derivation**

tree from a GE individual. Also, in the experiments carried out, modules are non-parameterized and may not be changed by evolutionary operators. However, there are a number of definitions of modules and how they are represented and evolved. No attempt was made to investigate these variations on modules' representations.

8.3 Opportunities for Future Research

The work presented in this thesis also presents a number of avenues for future work. This section outlines the most promising veins for new experiments.

One promising set of future experiments is implementing new module replacement operators. Currently, modules are assigned a fitness value upon their creation, and this value is the basis for which modules may remain available to the population during evolution. However, this value may not be an accurate representation of the utility of a module in later generations. There are many possibilities for updating this value based on which and how many individuals use a module. It is also possible to implement an operator equalization-like [35, 147] replacement operator where modules are replaced based on how they fit a size distribution. This could also be altered so modules are forced to fit a distribution of semantics as well.

Another interesting possibility for future work is the creation of new module identification operators. This thesis introduces two methods for sampling the semantics of a candidate module in GE and compares their performance to random and frequency-based approaches for identifying modules. In the current modularity implementation, only one of these methods may be used at a time. It may be beneficial to create a hybrid identification method that is able to leverage methods for estimating a module's worth based on the fitness of individuals in which it appears as well as using the frequency with which a module appears in the population.

Looking at how the modules themselves are represented also raises another potential extension of this work. Restricting modules to static, encapsulated elements is one approach to looking at modules, but allowing them to be parameterized and/or evolvable could increase their potential for improving GE's performance. This could be done using a number of methods, such as randomly selecting portions of the module to convert to a parameter or analyzing the population to look for patterns which could be used to determine which elements of a module should turn into a parameter. Allowing modules to be parameterized and evolvable would further increase the search space but could offer a more efficient route to a solution.

A final vein of future research that would prove valuable is applying the approaches to module identification and usage detailed in this thesis to GP. This could bring the work carried out in this thesis to a broader audience and allow for easier comparisons against past approaches to modularity.

Bibliography

[1] M. Ahluwalia and L. Bull. Co-evolving functions in genetic programming: Dynamic ADF creation using GLiB. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, volume 1447 of *LNCS*, pages 809–818, Mission Valley Marriott, San Diego, California, USA, 25-27 Mar. 1998. Springer-Verlag.

[2] M. Ahluwalia and L. Bull. Coevolving functions in genetic programming: Classification using K-nearest-neighbour. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 947–952, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[3] M. Ahluwalia and L. Bull. Coevolving functions in genetic programming. *Journal of Systems Architecture*, 47(7):573 – 585, 2001. Evolutionary computing.

[4] M. Ahluwalia, L. Bull, and T. C. Fogarty. Co-evolving functions in genetic programming: A comparison in ADF selection strategies. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 3–8, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[5] M. Ahluwalia and T. C. Fogarty. Co-evolving hierarchical programs using genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming*, GECCO '96, pages 419–419, Cambridge, MA, USA, 1996. MIT Press.

[6] L. Altenberg. The evolution of evolvability in genetic programming. *Advances in genetic programming*, pages 47–74, 1994.

[7] P. Angeline. Genetic programming and emergent intelligence. *Advances in genetic programming*, 1:75–98, 1994.

[8] P. J. Angeline and J. Pollack. Evolutionary module acquisition. In D. Fogel and W. Atmar, editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 154–163, La Jolla, CA, USA, 25-26 Feb. 1993.

[9] P. J. Angeline and J. B. Pollack. The evolutionary induction of subroutines. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 236–241, Bloomington, Indiana, USA, 1992. Lawrence Erlbaum.

[10] P. J. Angeline and J. B. Pollack. *Coevolving High-Level Representations*, pages 55–72. Addison-Wesley, 1994.

[11] D. Baltimore. Our genome unveiled. *Nature*, 409(6822):814–816, 02 2001.

[12] W. Banzhaf, D. Banscherus, and P. Dittrich. Hierarchical genetic programming using local modules. *InterJournal Complex Systems*, 228, 2000.

[13] A. Brabazon and M. O'Neill. *Biologically Inspired Algorithms for Financial Modelling*. Springer, 2006.

[14] O. Brock. Evolving reusable subroutines for genetic programming. *Artificial Life at Stanford*, pages 11–19, 1994.

[15] W. Bruce. *The application of genetic programming to the automatic generation of object-oriented programs*. PhD thesis, Nova Southeastern University, 1995.

[16] W. S. Bruce. Automatic generation of object-oriented programs using genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming*, GECCO '96, pages 267–272, Cambridge, MA, USA, 1996. MIT Press.

[17] W. S. Bruce. The lawnmower problem revisited: Stack-based genetic programming and automatically defined functions. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 52–57. Morgan Kaufmann, 1997.

[18] R. Calabretta, S. Nolfi, D. Parisi, and G. Wagner. A case study of the evolution of modularity: towards a bridge between evolutionary biology, artificial life, neuro-and cognitive science. *Artificial life six*, 6:275, 1998.

[19] R. Calabretta, S. Nolfi, D. Parisi, and G. P. Wagner. Emergence of functional modularity in robots. *From animals to animats*, 5:497–504, 1998.

[20] R. Calabretta, S. Nolfi, D. Parisi, and G. P. Wagner. Duplication of modules facilitates the evolution of functional specialization. *Artificial Life*, 6(1):69–84, 2012/08/06 2000.

[21] M. Cebrian, M. Alfonseca, and A. Ortega. Towards the validation of plagiarism detection tools by means of grammar evolution. *Evolutionary Computation, IEEE Transactions on*, 13(3):477 –485, june 2009.

[22] J. Clune, B. E. Beckmann, P. K. McKinley, and C. Ofria. Investigating whether hyperneat produces modular neural networks. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 635–642, New York, NY, USA, 2010. ACM.

[23] D. Cook and L. Holder. Graph-based data mining. *Intelligent Systems and their Applications, IEEE*, 15(2):32 –41, mar/apr 2000.

[24] N. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the First International Conference on Genetic Algorithms*, volume 183, page 187, 1985.

[25] S. Şen and J. A. Clark. A grammatical evolution approach to intrusion detection on mobile ad hoc networks. In *Proceedings of the second ACM conference on Wireless network security*, WiSec '09, pages 95–102, New York, NY, USA, 2009. ACM.

[26] J. Cullen. Evolutionary meta compilation: Evolving programs using real world engineering tools. In *Proceedings of the 8th international conference on Evolvable Systems: From Biology to Hardware*, ICES '08, pages 414–419, Berlin, Heidelberg, 2008. Springer-Verlag.

[27] J. Cullen. Evolving digital circuits in an industry standard hardware description language. In X. Li, M. Kirley, M. Zhang, D. Green, V. Ciesielski, H. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. Tan, J. Branke, and Y. Shi, editors, *Simulated Evolution and Learning*, volume 5361 of *Lecture Notes in Computer Science*, pages 514–523. Springer Berlin / Heidelberg, 2008.

[28] C. Darwin. *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life.* John Murray, 1859.

[29] E. de Jong, D. Thierens, and R. Watson. Hierarchical genetic algorithms. In X. Yao, E. Burke, J. Lozano, J. Smith, J. Merelo-Guervs, J. Bullinaria, J. Rowe, P. Tino, A. Kabn, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 232–241. Springer Berlin / Heidelberg, 2004.

[30] E. D. de Jong and D. Thierens. Exploiting modularity, hierarchy, and repetition in variable-length problems. *Genetic and Evolutionary Computation –GECCO 2004*, pages 1030–1041, 2004.

[31] E. D. de Jong, D. Thierens, and R. A. Watson. Defining modularity, hierarchy, and repetition. In R. Poli, S. Cagnoni, and et al., editors, *GECCO 2004 Workshop Proceedings*, Seattle, Washington, USA, June 2004.

[32] E. D. de Jong, R. A. Watson, and D. Thierens. On the complexity of hierarchical problem solving. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1201–1208, New York, NY, USA, 2005. ACM.

[33] I. Dempsey, M. O'Neill, and A. Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments*. Springer, 2009.

[34] A. Dessi, A. Giani, and A. Starita. An analysis of automatic subroutine discovery in genetic programming. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 996–1001, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[35] S. Dignum and R. Poli. Operator equalisation and bloat free gp. In *Proceedings of the 11th European conference on Genetic programming*, EuroGP'08, pages 110–121, Berlin, Heidelberg, 2008. Springer-Verlag.

[36] J. Drchal and M. Šnorek. Tree-based indirect encodings for evolutionary development of neural networks. In V. Kurková, R. Neruda, and J. Koutník, editors, *Artificial Neural Networks - ICANN 2008*, volume 5164 of *Lecture Notes in Computer Science*, pages 839–848. Springer Berlin / Heidelberg, 2008.

[37] G. Escuela, G. Ochoa, and N. Krasnogor. Evolving l-systems to capture protein structure native conformations. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, editors, *Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 142–142. Springer Berlin / Heidelberg, 2005.

[38] D. Fagan, E. Hemberg, M. Nicolau, M. O'Neill, and S. McGarraghy. Towards adaptive mutation in grammatical evolution. In T. Soule, A. Auger, J. Moore, D. Pelta, C. Solnon, M. Preuss, A. Dorin, Y.-S. Ong, C. Blum, D. L. Silva, F. Neumann, T. Yu, A. Ekart, W. Browne, T. Kovacs, M.-L. Wong, C. Pizzuti, J. Rowe, T. Friedrich, G. Squillero, N. Bredeche, S. Smith, A. Motsinger-Rei, J. Lozano, M. Pelikan, S. Meyer-Nienber, C. Igel, G. Hornby, R. Doursat, S. Gustafson, G. Olague, S. Yoo, J. Clark, G. Ochoa, G. Pappa, F. Lobo, D. Tauritz, J. Branke, and K. Deb, editors, *GECCO Companion '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 1481–1482, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM.

[39] D. Fagan, E. Hemberg, M. O'Neill, and S. McGarraghy. Fitness reactive mutation in grammatical evolution. In R. Matousek, editor, *18th International Conference on Soft Computing, MENDEL 2012*, pages 144–149, Brno, Czech Republic, 27-29 June 2012. Brno University of Technology.

[40] D. Fagan, M. Nicolau, E. Hemberg, M. O'Neill, A. Brabazon, and S. McGarraghy. Investigation of the performance of different mapping orders for GE on the max problem. In S. Silva, J. A. Foster, M. Nicolau, M. Giacobini, and P. Machado, editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 286–297, Turin, Italy, 27-29 Apr. 2011. Springer Verlag.

[41] D. Fagan, M. O'Neill, E. G. López, A. Brabazon, and S. McGarraghy. An analysis of genotype-phenotype maps in grammatical evolution. In A. I. Esparcia-Alcazar, A. Ekart, S. Silva, S. Dignum, and A. S. Uyar, editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 62–73, Istanbul, 7-9 Apr. 2010. Springer.

[42] I. I. Garibay, O. O. Garibay, and A. S. Wu. Effects of module encapsulation in repetitively modular genotypes on the search space. *Genetic and Evolutionary Computation –GECCO 2004*, pages 1125–1137, 2004.

[43] O. Garibay, I. Garibay, and A. Wu. The modular genetic algorithm: Exploiting regularities in the problem space. *Computer and Information Sciences - ISCIS 2003*, pages 584–591, 2003.

[44] O. O. Garibay. *Analyzing the Effects of Modularity on Search Spaces*. PhD thesis, University of Central Florida, 2008.

[45] O. O. Garibay and A. S. Wu. Analyzing the effects of module encapsulation on search space bias. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1234–1241, New York, NY, USA, 2007. ACM.

[46] L. Georgiou and W. J. Teahan. Constituent grammatical evolution. In T. Walsh, editor, *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 1261–1268, Barcelona, Spain, 16-22 July 2011. AAAI Press.

[47] G. Georgoulas, D. Gavrilis, I. G. Tsoulos, C. Stylios, J. Bernardes, and P. P. Groumpos. Novel approach for fetal heart rate classification introducing grammatical evolution. *Biomedical Signal Processing and Control*, 2(2):69 – 79, 2007.

[48] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[49] F. Gruau. Genetic synthesis of modular neural networks. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 318–325, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[50] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2):151–183, Sept. 1994.

[51] G. R. Harik and D. E. Goldberg. Learning linkage. In R. K. Belew and M. D. Vose, editors, *Proceedings of the 4zh Workshop on Foundations of Genetic Algorithms*, pages 247–262, San Francisco, Aug. 5 1997. Morgan Kaufman.

[52] R. Harper. *Enhancing Grammatical Evolution*. PhD thesis, The University of New South Wales, 2009.

[53] R. Harper. Ge, explosive grammars and the lasting legacy of bad initialisation. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1 –8, july 2010.

[54] R. Harper and A. Blair. A structure preserving crossover in grammatical evolution. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume Volume 3, pages 2537–2544. IEEE Press, 2005.

[55] R. Harper and A. Blair. Dynamically defined functions in grammatical evolution. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 9188–9195, Vancouver, 6-21 July 2006. IEEE Press.

[56] L. Hartwell, J. Hopfield, S. Leibler, A. Murray, et al. From molecular to modular cell biology. *Nature*, 402(6761):47–52, 1999.

[57] E. Hemberg. *An Exploration of Grammars in Grammatical Evolution*. PhD thesis, University College Dublin, 2010.

[58] E. Hemberg, C. Gilligan, M. O'Neill, and A. Brabazon. A grammatical genetic programming approach to modularity in genetic algorithms. In M. Ebner et al., editors, *EuroGP 2007: Proceedings of the 10th European Conference on Genetic Programming*, number 4445 in LNCS, Valencia, Spain, 2007. Springer.

[59] E. Hemberg, M. O'Neill, and A. Brabazon. An investigation into automatically defined function representations in grammatical evolution. In R. Matousek and L. Nolle, editors, *15th International Conference on Soft Computing, Mendel'09*, Brno, Czech Republic, 24-26 June 2009.

[60] M. Hemberg and U.-M. O'Reilly. Extending grammatical evolution to evolve digital surfaces with Genr8. In M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 299–308, Coimbra, Portugal, April 2004. Springer-Verlag.

[61] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975.

[62] J. H. Holland. *Hidden Order: How Adaptation Builds Complexity.* Perseus Books, 1995.

[63] G. Hornby, H. Lipson, and J. Pollack. Generative representations for the automated design of modular physical robots. *Robotics and Automation, IEEE Transactions on*, 19(4):703 – 719, aug. 2003.

[64] G. S. Hornby. Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. In H.-G. Beyer and U.-M. O. et al., editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1729–1736, Washington DC, USA, 25-29 June 2005. ACM Press.

[65] C.-C. Huang and A. Kusiak. Modularity in design of products and systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 28(1):66–77, Jan 1998.

[66] C. Jassadapakorn and P. Chongstitvatana. Reduction of computational effort in genetic programming by subroutines. *Chulalongkorn University, Bangkok, Thailand*, 1998.

[67] I. Jonyer and A. Himes. Improving modularity in genetic programming using graph-based data mining. In G. C. J. Sutcliffe and R. G. Goebel, editors, *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*, pages 556–561, Melbourne Beach, Florida, USA, May 11-13 2006. American Association for Artificial Intelligence.

[68] P. Kaufmann and M. Platzner. Advanced techniques for the creation and propagation of modules in cartesian genetic programming. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1219–1226, New York, NY, USA, 2008. ACM.

[69] M. Keijzer, C. Ryan, and M. Cattolico. Run transferable libraries —learning functional bias in problem domains. *Genetic and Evolutionary Computation –GECCO 2004*, pages 531–542, 2004.

[70] M. Keijzer, C. Ryan, G. Murphy, and M. Cattolico. Undirected training of run transferable libraries. *Genetic Programming*, pages 361–370, 2005.

[71] K. Kinnear Jr. Alternatives in automatic function definition: A comparison of performance. *Advances in Genetic Programming*, pages 119–141, 1994.

[72] J. Koza. *Genetic programming III: darwinian invention and problem solving*. Morgan Kaufmann Pub, 1999.

[73] J. R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[74] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, USA, 1994.

[75] K. Krawiec. On relationships between semantic diversity, complexity and modularity of programming tasks. In T. Soule and J. H. Moore, editors, *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012*, pages 783–790. ACM, 2012.

[76] K. Krawiec and T. Pawlak. Locally geometric semantic crossover. In T. Soule, A. Auger, J. Moore, D. Pelta, C. Solnon, M. Preuss, A. Dorin, Y.-S. Ong, C. Blum, D. L. Silva, F. Neumann, T. Yu, A. Ekart, W. Browne, T. Kovacs, M.-L. Wong, C. Pizzuti, J. Rowe, T. Friedrich, G. Squillero, N. Bredeche, S. Smith, A. Motsinger-Rei, J. Lozano, M. Pelikan, S. Meyer-Nienber, C. Igel, G. Hornby, R. Doursat, S. Gustafson, G. Olague, S. Yoo, J. Clark, G. Ochoa, G. Pappa, F. Lobo, D. Tauritz, J. Branke, and K. Deb, editors, *GECCO Companion '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 1487–1488, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM.

[77] K. Krawiec and B. Wieloch. Analysis of semantic modularity for genetic programming. *Foundations of Computing and Decision Sciences*, 34(4):265–285, 2009.

[78] K. Krawiec and B. Wieloch. Functional modularity for genetic programming. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 995–1002, New York, NY, USA, 2009. ACM.

[79] X. Li, C. Zhou, W. Xiao, and P. Nelson. Direct evolution of hierarchical solutions with self-emergent substructures. In *Machine Learning and Applications, 2005. Proceedings. Fourth International Conference on*, page 6 pp., December 2005.

[80] X. Li, C. Zhou, W. Xiao, and P. C. Nelson. Prefix gene expression programming. In F. Rothlauf, editor, *Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO'2005)*, Washington, D.C., USA, 25-29 June 2005.

[81] X. Li, C. Zhou, W. Xiao, and P. C. Nelson. Introducing emergent loose modules into the learning process of a linear genetic programming system. *Machine Learning and Applications, Fourth International Conference on*, 0:219–224, 2006.

[82] H. Lipson, J. B. Pollack, and N. P. Suh. Promoting modularity in evolutionary design. In *Proceedings of DETC'01: 2001 ASME Design Engineering Technical Conferences*, 2001.

[83] H. Lipson, J. B. Pollack, and N. P. Suh. On the origin of modular variation. *Evolution*, 56(8):1549–1556, 2002.

[84] O. Litvin, H. C. Causton, B.-J. Chen, and D. Pe'er. Modularity and interactions in the genetics of gene expression. *Proceedings of the National Academy of Sciences*, 106(16):6441–6446, 2009.

[85] E. G. López, D. Fagan, E. Murphy, J. M. Swafford, A. Agapitos, M. O'Neill, and A. Brabazon. Comparing the performance of the evolvable pigrammatical evolution genotype-phenotype map to grammatical evolution in the dynamic Ms. Pac-Man environment. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*, pages 1–8. IEEE, 2010.

[86] E. G. López, J. M. Swafford, M. O'Neill, and A. Brabazon. Evolving a Ms. Pac-Man controller using grammatical evolution. In C. D. Chio, S. Cagnoni, C. Cotta,

M. Ebner, A. Ekárt, A. Esparcia-Alcázar, C. K. Goh, J. J. M. Guervós, F. Neri, M. Preuss, J. Togelius, and G. N. Yannakakis, editors, *EvoApplications (1)*, volume 6024 of *Lecture Notes in Computer Science*, pages 161–170. Springer, 2010.

[87] M. Luerssen and D. Powers. Evolving encapsulated programs as shared grammars. *Genetic Programming and Evolvable Machines*, 9(3):203–228, 09 2008.

[88] H. Majeed and C. Ryan. Context-aware mutation: a modular, context aware mutation operator for genetic programming. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1651–1658, New York, NY, USA, 2007. ACM.

[89] H. Majeed, C. Ryan, and R. M. Atif Azad. Evaluating gp schema in context. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1773–1774, New York, NY, USA, 2005. ACM.

[90] J. McDermott, J. Byrne, J. M. Swafford, M. Hemberg, C. McNally, E. Shotton, E. Hemberg, M. Fenton, and M. O'Neill. String-rewriting grammars for evolutionary architectural design. *Environment and Planning B: Planning and Design*, 39(4):713–731, 2012.

[91] J. McDermott, J. Byrne, J. M. Swafford, M. O'Neill, and A. Brabazon. Higher-order functions in aesthetic EC encodings. In *CEC 2010. Proceedings of the 12th IEEE Conference on Evolutionary Computation*, Jul 2010.

[92] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3/4):365–396, Sept. 2010. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.

[93] N. F. McPhee, B. Ohs, and T. Hutchison. Semantic building blocks in genetic programming. In *Proceedings of the 11th European conference on Genetic programming*, EuroGP'08, pages 134–145, Berlin, Heidelberg, 2008. Springer-Verlag.

[94] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, 1988.

[95] J. F. Miller and P. Thomson. Cartesian genetic programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, and T. C. Fogarty, editors, *Proceedings of the Third European Conference on Genetic Programming (EuroGP-2000)*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, Scotland, 2000. Springer Verlag.

[96] A. Moraglio, K. Krawiec, and C. G. Johnson. Geometric semantic genetic programming. In C. A. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature, PPSN XII (part 1)*, volume 7491 of *Lecture Notes in Computer Science*, pages 21–31, Taormina, Italy, Sep 2012. Springer.

[97] E. Murphy. Examining grammars and grammatical evolution in dynamic environments. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 779–782, Dublin, Ireland, 2011. ACM.

[98] E. Murphy, M. O'Neill, and A. Brabazon. A comparison of ge and tage in dynamic environments. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1387–1394, New York, NY, USA, 2011. ACM.

[99] E. Murphy, M. O'Neill, and A. Brabazon. Examining mutation landscapes in grammar based genetic programming. In S. Silva, J. A. Foster, M. Nicolau, M. Giacobini, and P. Machado, editors, *Proceedings of the 14th European Conference on Genetic*

Programming, EuroGP 2011, volume 6621 of *LNCS*, pages 130–141, Turin, Italy, 27-29 Apr. 2011. Springer Verlag. Best paper.

[100] E. Murphy, M. O'Neill, E. G. López, and A. Brabazon. Tree-adjunct grammatical evolution. In *2010 IEEE World Congress on Computational Intelligence*, pages 4449–4456, Barcelona, Spain, 18-23 July 2010. IEEE Computational Intelligence Society, IEEE Press.

[101] J. Murphy, M. O'Neill, and H. Carr. Exploring grammatical evolution for horse gait optimisation. In L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, and M. Ebner, editors, *Genetic Programming*, volume 5481 of *Lecture Notes in Computer Science*, pages 183–194. Springer Berlin / Heidelberg, 2009.

[102] Q. U. Nguyen, X. H. Nguyen, and M. O'Neill. Semantic aware crossover for genetic programming: The case for real-valued function regression. In L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, and M. Ebner, editors, *Genetic Programming*, volume 5481 of *Lecture Notes in Computer Science*, pages 292–302. Springer Berlin / Heidelberg, 2009.

[103] M. Nicolau. Automatic grammar complexity reduction in grammatical evolution. In R. Poli, S. Cagnoni, M. Keijzer, E. Costa, F. Pereira, G. Raidl, S. C. Upton, D. Goldberg, H. Lipson, E. de Jong, J. Koza, H. Suzuki, H. Sawai, I. Parmee, M. Pelikan, K. Sastry, D. Thierens, W. Stolzmann, P. L. Lanzi, S. W. Wilson, M. O'Neill, C. Ryan, T. Yu, J. F. Miller, I. Garibay, G. Holifield, A. S. Wu, T. Riopka, M. M. Meysenburg, A. W. Wright, N. Richter, J. H. Moore, M. D. Ritchie, L. Davis, R. Roy, and M. Jakiela, editors, *GECCO 2004 Workshop Proceedings*, Seattle, Washington, USA, 26-30 June 2004.

[104] M. Olmer, P. Nordin, and W. Banzhaf. Evolving real-time behavior modules for a real robot with GP. In M. Jamshidi, F. Pin, and P. Dauchez, editors, *Proceedings of the 6th international symposium on robotics and manufacturing, ISRAM-96*, pages 675–680, Montpellier, France, May 1996. Asme Press.

[105] J. Olsson. How to invent functions. In R. Poli, P. Nordin, W. Langdon, and T. Fogarty, editors, *Genetic Programming*, volume 1598 of *Lecture Notes in Computer Science*, pages 652–652. Springer Berlin / Heidelberg, 1999.

[106] J. R. Olsson. The art of writing specifications for the ADATE automatic programming system. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 278–283, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[107] R. Olsson. Inductive functional programming using incremental program transformation. *Artificial Intelligence*, 74(1):55–81, Mar. 1995.

[108] R. Olsson. Population management for automatic design of algorithms through evolution. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 592 –597, may 1998.

[109] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, and A. Brabazon. GEVA: grammatical evolution in Java. *ACM SIGEVOlution*, 3(2):17–22, 2008.

[110] M. O'Neill, J. McDermott, J. M. Swafford, J. Byrne, E. Hemberg, E. Shotton, C. McNally, A. Brabazon, and M. Hemberg. Evolutionary design using grammatical evolution and shape grammars: Designing a shelter. *International Journal of Design Engineering*, 3, 2010.

[111] M. O'Neill, M. Nicolau, and A. Brabazon. Dynamic environments can speed up evolution with genetic programming. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 191–192, New York, NY, USA, 2011. ACM.

[112] M. O'Neill and C. Ryan. Under the hood of grammatical evolution. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic & Evolutionary Computation Conference*, volume 2, pages 1143–1148, July 1999.

[113] M. O'Neill and C. Ryan. Grammar based function definition in grammatical evolution. In *GECCO '00: Proceedings of the Genetic and evolutionary computation Conference*, pages 485–490, 2000.

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

[115] M. O'Neill, C. Ryan, M. Keijzer, and M. Cattolico. Crossover in grammatical evolution. *Genetic Programming and Evolvable Machines*, 4:67–93, 2003.

[116] M. O'Neill, J. M. Swafford, J. McDermott, J. Byrne, A. Brabazon, E. Shotton, C. McNally, and M. Hemberg. Shape grammars and grammatical evolution for evolutionary design. In G. Raidl, F. Rothlauf, and et al., editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1035–1042, Montreal, 8-12 July 2009. ACM.

[117] M. O'Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11:339–363, 2010. 10.1007/s10710-010-9113-2.

[118] U.-M. O'Reilly. Investigating the generality of automatically defined functions. In *GECCO '96: Proceedings of the First Annual Conference on Genetic Programming*, pages 351–356, Cambridge, MA, USA, 1996. MIT Press.

[119] A. Ortega, A. A. Dalhoum, and M. Alfonseca. Grammatical evolution to design fractal curves with a given dimension. *IBM Journal of Research and Development*, 47(4):483 –493, july 2003.

[120] M. ONeill and C. Ryan. Crossover in grammatical evolution: A smooth operator? In R. Poli, W. Banzhaf, W. Langdon, J. Miller, P. Nordin, and T. Fogarty, editors, *Genetic Programming*, volume 1802 of *Lecture Notes in Computer Science*, pages 149–162. Springer Berlin Heidelberg, 2000.

[121] M. ONeill and C. Ryan. Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. In M. Keijzer, U.-M. OReilly, S. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming*, volume 3003 of *Lecture Notes in Computer Science*, pages 138–149. Springer Berlin Heidelberg, 2004.

[122] M. ONeill, C. Ryan, M. Keijzer, and M. Cattolico. Crossover in grammatical evolution: The search continues. In J. Miller, M. Tomassini, P. Lanzi, C. Ryan, A. Tettamanzi, and W. Langdon, editors, *Genetic Programming*, volume 2038 of *Lecture Notes in Computer Science*, pages 337–347. Springer Berlin Heidelberg, 2001.

[123] J. Parent, A. Nowe, K. Steenhaut, and A. Defaweux. Linear genetic programming using a compressed genotype representation. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1164 – 1171 Vol. 2, sept. 2005.

[124] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz. Boa: The bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary*

Computation Conference, GECCO '99, pages 525–532, San Fransisco, CA, USA, 1999. Morgan Kaufmann Publishers.

[125] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon. Evolving behavior trees for the mario AI competition using grammatical evolution. In C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekart, A. I. Esparcia-Alcazar, J. J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. N. Yannakakis, editors, *Applications of Evolutionary Computing, EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, EvoSTOC*, volume 6624 of *LNCS*, pages 121–130, Turin, Italy, 27-29 Apr. 2011. Springer Verlag.

[126] T. Perkis. Stack-based genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 148–153, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[127] R. Poli. Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, University of Birmingham, B15 2TT, UK, September 1996.

[128] The R Project for Statistical Computing. `http://www.r-project.com`. Accessed: 27/09/2012.

[129] A. Racine, M. Schoenauer, and P. Dague. A dynamic lattice to envolve hierarchically shared subroutines. In W. Banzhaf, R. Poli, M. Schoenauer, and T. Fogarty, editors, *Genetic Programming*, volume 1391 of *Lecture Notes in Computer Science*, pages 220–232. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0055941.

[130] J. Reddin, J. McDermott, and M. O'Neill. Elevated pitch: Automated grammatical evolution of short compositions. In M. Giacobini et al., editors, *Applications of*

Evolutionary Computing: EvoWorkshops 2004, number 5484 in LNCS, pages 579–584. Springer-Verlag, 2009.

[131] E. Rich and K. Knight. *Artificial Intelligence*. McGraw Hill, 1991.

[132] S. Roberts, D. Howard, and J. Koza. Evolving modules in genetic programming by subtree encapsulation. In J. Miller, M. Tomassini, P. Lanzi, C. Ryan, A. Tettamanzi, and W. Langdon, editors, *Genetic Programming*, volume 2038 of *Lecture Notes in Computer Science*, pages 160–175. Springer Berlin / Heidelberg, 2001.

[133] E. Rodrigues and A. Pozo. Grammar-guided genetic programming and automatically defined functions. In G. Bittencourt and G. Ramalho, editors, *Advances in Artificial Intelligence: 16th Brazilian Symposium on Artificial Intelligence, SBIA 2002, Proceedings*, volume 2507 of *LNAI*, pages 324–333, Porto de Galinhas/Recife, Brazil, 11-14 Nov. 2002. Springer.

[134] J. Rosca. *Hierarchical Learning with Procedural Abstraction Mechanisms*. PhD thesis, University of Rochester, 1997.

[135] J. Rosca and D. Ballard. Hierarchical self-organization in genetic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 251–258. Morgan Kaufmann, 1994.

[136] J. Rosca and D. Ballard. Learning by adapting representations in genetic programming. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 407–412 vol.1, Jun 1994.

[137] J. Rosca and D. H. Ballard. Evolution-based discovery of hierarchical behaviors. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*. AAAI / The MIT Press, 1996.

[138] J. P. Rosca. Genetic programming exploratory power and the discovery of functions. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Evolutionary Programming IV Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 719–736, San Diego, CA, USA, 1-3 Mar. 1995. MIT Press.

[139] J. P. Rosca. Generality versus size in genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming*, GECCO '96, pages 381–387, Cambridge, MA, USA, 1996. MIT Press.

[140] J. P. Rosca and D. H. Ballard. *Discovery of subroutines in genetic programming*, pages 177–201. MIT Press, Cambridge, MA, USA, 1996.

[141] D. E. Rumelhart, G. E. Hinton, and R. Willians. Learning internal representations by error propagation. In *Parallel Distributed Processing*. MIT Press, 1986.

[142] C. Ryan, M. Keijzer, and M. Cattolico. Favourable biasing of function sets using run transferable libraries. *Genetic Programming Theory and Practice II*, pages 103–120, 2004.

[143] G. Saunders, J. Kolen, P. Angeline, and J. Pollack. Additive modular learning in preemptrons. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 1098–1103, 1992.

[144] G. Schlosser and G. Wagner. *Modularity in development and evolution*. University of Chicago Press, 2004.

[145] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[146] J. Shao, J. McDermott, M. O'Neill, and A. Brabazon. Jive: A generative, interactive, virtual, evolutionary music system. In C. Di Chio, A. Brabazon, G. Di Caro,

M. Ebner, M. Farooq, A. Fink, J. Grahl, G. Greenfield, P. Machado, M. O'Neill, E. Tarantino, and N. Urquhart, editors, *Applications of Evolutionary Computation*, volume 6025 of *Lecture Notes in Computer Science*, pages 341–350. Springer Berlin / Heidelberg, 2010.

[147] S. Silva and S. Dignum. Extending operator equalisation: Fitness based self adaptive length distribution for bloat free gp. In L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, and M. Ebner, editors, *Genetic Programming*, volume 5481 of *Lecture Notes in Computer Science*, pages 159–170. Springer Berlin / Heidelberg, 2009.

[148] H. A. Simon. *The sciences of the artificial*. MIT Press, 3 edition, 1996.

[149] L. Spector. Simultaneous evolution of programs and their control structures. *Advances in genetic programming*, 2:137–154, 1996.

[150] L. Spector. Autoconstructive evolution: Push, pushGP, and pushpop. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 137–146, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[151] L. Spector, K. I. Harrington, and T. Helmuth. Tag-based modularity in tree-based genetic programming. In T. Soule and J. H. Moore, editors, *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012*, pages 815–822. ACM, 2012.

[152] L. Spector, B. Martin, K. Harrington, and T. Helmuth. Tag-based modules in genetic programming. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1419–1426, New York, NY, USA, 2011. ACM.

[153] L. Spector and A. Robinson. Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40, Mar. 2002.

[154] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, April 2009.

[155] J. M. Swafford, E. Hemberg, M. O'Neill, and A. Brabazon. Analyzing module usage in grammatical evolution. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Proceedings of the 12th Int. Conf. on Parallel Problem Solving From Nature*, volume 7491 of *Lecture Notes in Computer Science*. Springer, 2012.

[156] J. M. Swafford, E. Hemberg, M. O'Neill, M. Nicolau, and A. Brabazon. A non-destructive grammar modification approach to modularity in grammatical evolution. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1411–1418, Dublin, Ireland, 2011. ACM.

[157] J. M. Swafford, M. Nicolau, E. Hemberg, M. O'Neill, and A. Brabazon. Comparing methods for module identification in grammatical evolution. In T. Soule and J. H. Moore, editors, *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012*, pages 823–830. ACM, 2012.

[158] J. M. Swafford and M. O'Neill. An examination on the modularity of grammars in grammatical evolutionary design. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*. IEEE, Jul 2010.

[159] J. M. Swafford, M. O'Neill, M. Nicolau, and A. Brabazon. Exploring grammatical modification with modules in grammatical evolution. In S. Silva, J. A. Foster,

M. Nicolau, P. Machado, and M. Giacobini, editors, *EuroGP*, volume 6621 of *Lecture Notes in Computer Science*, pages 310–321. Springer, 2011.

[160] W. Teahan. *Artificial Intelligence–Agents and Environments*. Ventus Publishing ApS, 2010.

[161] P. Verbancsics and K. O. Stanley. Constraining connectivity to encourage modularity in HyperNEAT. Technical Report CS-TR-10-10, Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816, USA, November 2010.

[162] G. Wagner. Adaptation and the modular design of organisms. In F. Morán, A. Moreno, J. Merelo, and P. Chacón, editors, *Advances in Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 315–328. Springer Berlin / Heidelberg, 1995.

[163] G. P. Wagner. Homologues, natural kinds and the evolution of modularity. *American Zoologist*, 36(1):36–43, 1996.

[164] J. Walker and J. Miller. Evolution and acquisition of modules in cartesian genetic programming. In M. Keijzer, U.-M. O'Reilly, S. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming*, volume 3003 of *Lecture Notes in Computer Science*, pages 187–197. Springer Berlin / Heidelberg, 2004.

[165] J. Walker and J. Miller. The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. *Evolutionary Computation, IEEE Transactions on*, 12(4):397 –417, August 2008.

[166] J. A. Walker and J. F. Miller. Investigating the performance of module acquisition in cartesian genetic programming. In *GECCO '05: Proceedings of the 2005 conference*

on Genetic and evolutionary computation, pages 1649–1656, New York, NY, USA, 2005. ACM.

[167] J. A. Walker and J. F. Miller. Embedded cartesian genetic programming and the lawnmower and hierarchical-if-and-only-if problems. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 911–918, New York, NY, USA, 2006. ACM.

[168] J. A. Walker, J. F. Miller, and R. Cavill. A multi-chromosome approach to standard and embedded cartesian genetic programming. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 903–910, New York, NY, USA, 2006. ACM.

[169] A. R. Wallace. *The Malay Archipelago*. Harper, 1869.

[170] P. Whigham. Inductive bias and genetic programming. In *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414)*, pages 461 –466, Sept. 1995.

[171] P. Whigham. *Grammatical Bias for Evolutionary Learning*. PhD thesis, University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 1996.

[172] D. Wilson and D. Kaur. Using grammatical evolution for evolving intrusion detection rules. In *Proceedings of the 5th WSEAS International Conference on Information Security and Privacy*, pages 183–188, Stevens Point, Wisconsin, USA, 2006. World Scientific and Engineering Academy and Society (WSEAS).

[173] J. Woodward. Modularity in genetic programming. In *In Genetic Programming, Proceedings of EuroGP 2003*, pages 14–16. Springer-Verlag, 2003.

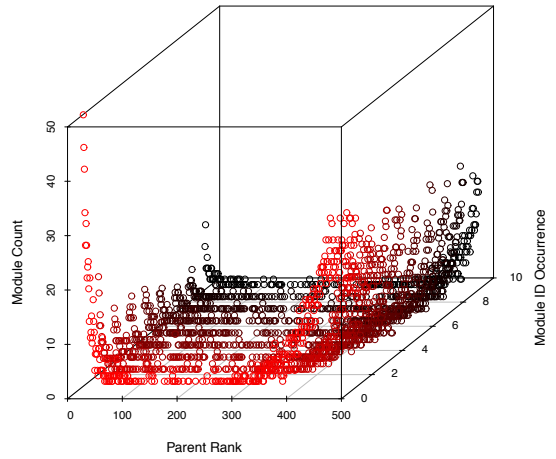
[174] T. Yu. Hierarchical processing for evolving recursive and modular programs using higher-order functions and lambda abstraction. *Genetic Programming and Evolvable Machines*, 2:345–380, 2001. 10.1023/A:1012926821302.

Appendices

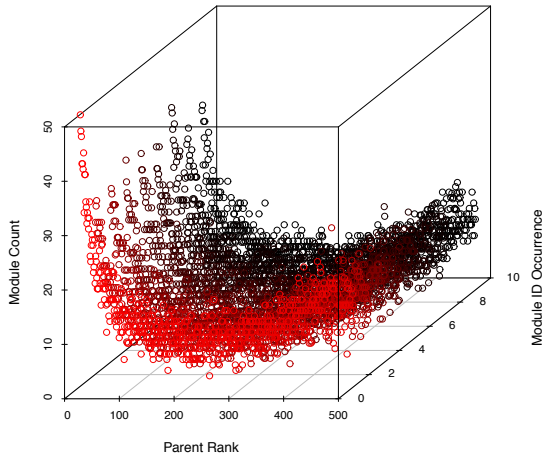
Appendix A

Module Identification (Chapter 6)

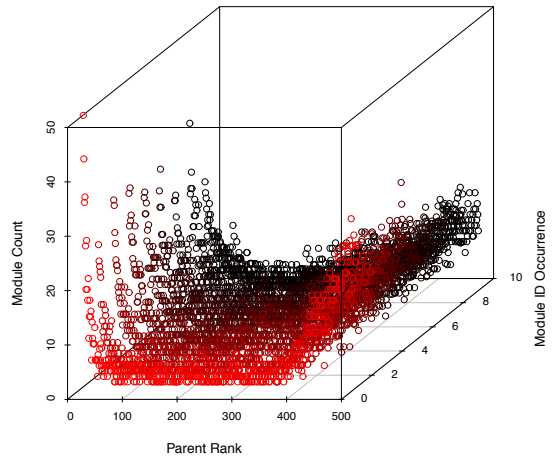
A.1 Initial Results with Module Identification



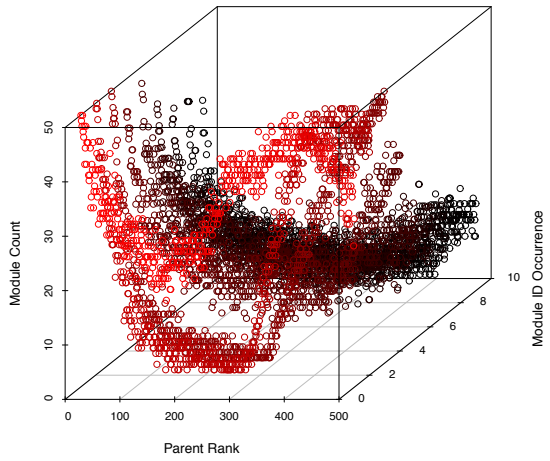
(a) Santa Fe Ant Trail



(b) Even 7 Parity

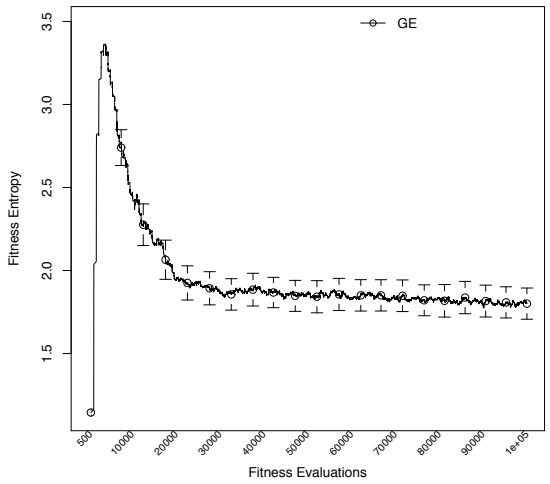


(c) $x^5 - 2x^3 + x$ Symbolic Regression

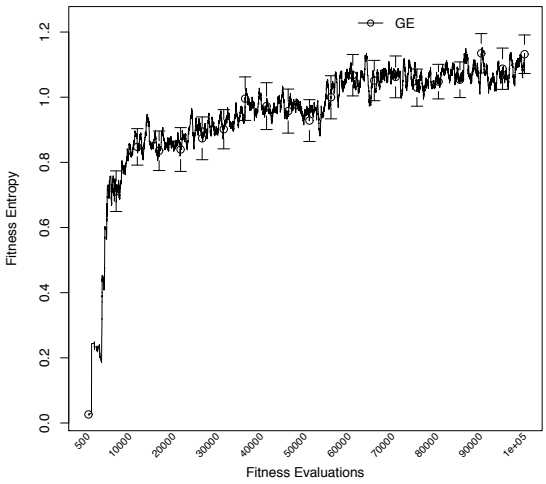


(d) 8×8 Lawn Mower

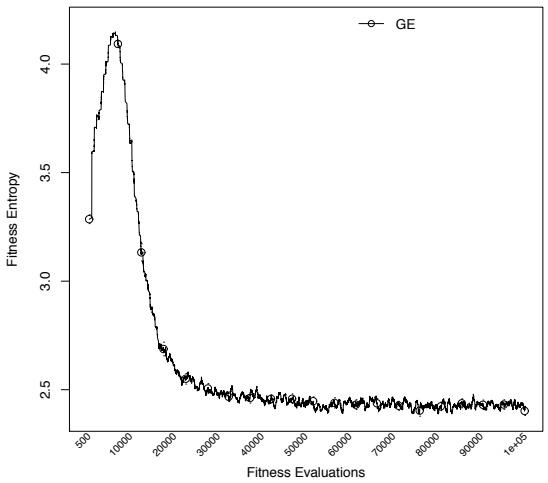
Figure A.1: These figures show which section of the population contributes modules during the identification process. The x-axis represents an individual's rank in the population based on fitness, 1 being the best and 500 being the worst. The y-axis (height) represents how many times individuals with a given rank contributed a module over the course of 50 runs. This value does not take into account the quality of the module discovered. The z-axis (depth) denotes how many times modules have been identified. The R-ID approach was used for these figures.



(a) Even 7 Parity



(b) $x^5 - 2x^3 + x$ Symbolic Regression



(c) 8×8 Lawn Mower

Figure A.2: These figures show how the diversity of the population's fitness values changes over time during standard GE runs. The data is averaged across 50 runs. Diversity is measured using Shannon entropy [145]

A.2 Reducing Module Selection Pressure

Table A.1: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the Santa Fe Ant Trail problem after 100000 fitness evaluations.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
I-ID ρ-50	17.220	13.630	1.928	7
I-ID ρ-25	18.860	13.391	1.894	9
I-ID ρ-75	19.900	13.142	1.859	6
M-ID ρ-75	23.120	12.953	1.832	5
M-ID ρ-50	23.340	12.920	1.827	5
M-ID ρ-25	23.360	12.912	1.826	5

Table A.2: This table reports the p-value of Wilcoxon rank-sum tests performed on the average best fitness values of each approach after 100000 fitness evaluations on the Santa Fe Ant Trail problem. The p-values reported are calculated with a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	I-ID ρ-50	I-ID ρ-25	I-ID ρ-75	M-ID ρ-75	M-ID ρ-50	M-ID ρ-25
I-ID ρ-50		0.593	0.240	0.017*	0.023*	0.010*
I-ID ρ-25	0.593		0.663	0.109	0.062	0.125
I-ID ρ-75	0.240	0.663		0.196	0.206	0.093
M-ID ρ-75	0.017*	0.109	0.196		0.844	0.918
M-ID ρ-50	0.023*	0.062	0.206	0.844		0.918
M-ID ρ-25	0.010*	0.123	0.093	0.918	0.918	

Table A.3: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the Even 7 Parity problem after 100000 fitness evaluations.

Approach	Best Fitness	Std. Dev.	Std. Err.	Number Solved
M-ID ρ-75	5.260	8.710	1.232	34
M-ID ρ-50	5.320	8.759	1.239	34
M-ID ρ-25	5.880	8.859	1.253	32
I-ID ρ-75	6.440	9.752	1.379	30
I-ID ρ-50	6.440	9.752	1.379	30
I-ID ρ-25	6.440	9.752	1.379	30

Table A.4: This table reports the p-value of Wilcoxon rank-sum tests performed on the average best fitness values of each approach after 100000 fitness evaluations on the Even 7 Parity problem. The p-values reported are calculated with a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	M-ID ρ-75	M-ID ρ-50	M-ID ρ-25	I-ID ρ-75	I-ID ρ-50	I-ID ρ-25
M-ID ρ-75		1.000	0.696	0.522	0.501	0.500
M-ID ρ-50	1.000		0.922	0.500	0.551	0.413
M-ID ρ-25	0.686	0.922		0.637	0.641	0.535
I-ID ρ-75	0.522	0.500	0.637		0.891	0.964
I-ID ρ-50	0.501	0.551	0.641	0.891		0.981
I-ID ρ-25	0.500	0.413	0.535	0.964	0.981	

Table A.5: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the $x^5 - 2x^3 + x$ Symbolic Regression problem after 100000 fitness evaluations.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
I-ID ρ-75	0.510	0.407	0.058	4
I-ID ρ-25	0.511	0.347	0.049	4
I-ID ρ-50	0.544	0.445	0.063	4
M-ID ρ-75	0.704	0.530	0.075	2
M-ID ρ-25	0.720	0.603	0.085	5
M-ID ρ-50	0.785	0.633	0.090	5

Table A.6: This table reports the p-value of Wilcoxon rank-sum tests performed on the average best fitness values of each approach after 100000 fitness evaluations on the $x^5 - 2x^3 + x$ Symbolic Regression problem. The p-values reported are calculated with a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	I-ID ρ-75	I-ID ρ-25	I-ID ρ-50	M-ID ρ-75	M-ID ρ-25	M-ID ρ-50
I-ID ρ-75		0.903	0.636	0.005*	0.030*	0.017*
I-ID ρ-25	0.903		0.750	0.071	0.074	0.010*
I-ID ρ-50	0.636	0.750		0.144	0.041*	0.032*
M-ID ρ-75	0.005*	0.071	0.144		0.882	0.592
M-ID ρ-25	0.030*	0.074	0.041	0.882		0.627
M-ID ρ-50	0.017*	0.010*	0.032*	0.592	0.627	

Table A.7: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the 8×8 Lawn Mower problem after 100000 fitness evaluations.

	BestFit	StdDev.	StdErr.	NumSolved
M-ID ρ-75	3.27018	5.58502	0.78984	12.00000
M-ID ρ-50	3.27018	5.58502	0.78984	12.00000
M-ID ρ-25	3.27018	5.58502	0.78984	12.00000
I-ID ρ-25	28.41600	1.10774	0.15666	0.00000
I-ID ρ-75	29.23800	1.11041	0.15704	0.00000
I-ID ρ-50	29.23800	1.11041	0.15704	0.00000

Table A.8: This table reports the p-value of Wilcoxon rank-sum tests performed on the average best fitness values of each approach after 100000 fitness evaluations on the 8×8 Lawn Mower problem. The p-values reported are calculated with a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	M-ID ρ-75	M-ID ρ-50	M-ID ρ-25	I-ID ρ-25	I-ID ρ-75	I-ID ρ-50
M-ID ρ-75		0.979	0.704	7.8×10^{-10}*	7.7×10^{-10}*	7.8×10^{-10}*
M-ID ρ-50	0.979		0.894	7.8×10^{-10}*	7.7×10^{-10}*	7.7×10^{-10}*
M-ID ρ-25	0.704	0.894		7.7×10^{-10}*	7.7×10^{-10}*	7.7×10^{-10}*
I-ID ρ-25	7.8×10^{-10}*	7.8×10^{-10}*	7.7×10^{-10}*		0.005	0.009
I-ID ρ-75	7.7×10^{-10}*	7.7×10^{-10}*	7.7×10^{-10}*	0.005		0.941
I-ID ρ-50	7.8×10^{-10}*	7.7×10^{-10}*	7.7×10^{-10}*	0.009	0.941	

A.3 Reducing Fitness Evaluations

Table A.9: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the Santa Fe Ant Trail problem after 100000 fitness evaluations.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
I-ID TP n-25	15.720	13.240	1.872	9
I-ID TP n-50	16.380	13.041	1.844	9
F-ID AP	16.760	13.283	1.878	12
R-ID AP	17.240	12.934	1.829	12
I-ID AP n-10	18.100	13.287	1.879	9
M-ID AP n-10	18.300	13.452	1.902	12
I-ID AP n-25	18.300	13.474	1.906	11
I-ID AP n-50	18.860	13.391	1.894	9
M-ID TP n-25	19.220	13.196	1.866	8
M-ID TP n-10	19.260	14.294	2.021	9
F-ID TP	19.700	12.960	1.833	8
I-ID TP n-10	19.800	12.513	1.779	6
M-ID AP n-25	19.940	13.707	1.939	6
M-ID TP n-50	20.060	13.856	1.959	7
R-ID TP	21.300	12.918	1.827	7
M-ID AP n-50	23.120	12.953	1.832	5

Table A.10: This table reports the p-value of Wilcoxon rank-sum tests performed on the average best fitness values of each approach after 100000 fitness evaluations on the Santa Fe Ant Trail problem. The p-values reported are calculated with a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	I-ID TP n-25	I-ID TP n-50	F-ID AP	R-ID AP	I-ID AP n-10	M-ID AP n-10	I-ID AP n-25	I-ID AP n-50
I-ID TP n-25		0.779	0.789	0.698	0.290	0.149	0.231	0.130
I-ID TP n-50	0.779		0.896	0.883	0.395	0.429	0.382	0.350
F-ID AP	0.789	0.896		0.748	0.558	0.562	0.582	0.403
R-ID AP	0.698	0.883	0.748		0.748	0.628	0.684	0.596
I-ID AP n-10	0.290	0.395	0.558	0.748		0.992	1.000	0.813
M-ID AP n-10	0.149	0.429	0.562	0.628	0.992		0.969	0.806
I-ID AP n-25	0.231	0.382	0.582	0.684	1.000	0.969		0.779
I-ID AP n-50	0.130	0.350	0.403	0.596	0.813	0.806	0.779	
M-ID TP n-25	0.141	0.302	0.367	0.343	0.735	0.747	0.695	0.866
M-ID TP n-10	0.230	0.148	0.266	0.318	0.611	0.696	0.701	0.924
F-ID TP	0.161	0.135	0.309	0.359	0.488	0.704	0.559	0.546
I-ID TP n-10	0.119	0.240	0.211	0.382	0.515	0.404	0.479	0.686
M-ID AP n-25	0.119	0.133	0.186	0.274	0.491	0.406	0.420	0.611
M-ID TP n-50	0.107	0.165	0.210	0.274	0.495	0.612	0.192	0.412
R-ID TP	4.431×10^{-2}*	3.823×10^{-2}*	8.794×10^{-2}	4.540×10^{-2}*	0.148	0.209	0.269	0.440
M-ID AP n-50	2.820×10^{-3}*	8.529×10^{-3}*	9.959×10^{-3}*	3.616×10^{-2}*	0.117	4.528×10^{-2}*	3.998×10^{-2}*	0.109

	M-ID TP n-25	M-ID TP n-10	F-ID TP	I-ID TP n-10	M-ID AP n-25	M-ID TP n-50	R-ID TP	M-ID AP n-50
I-ID TP n-25	0.141	0.230	0.161	0.119	0.119	0.107	4.431×10^{-2}*	2.820×10^{-3}*
I-ID TP n-50	0.302	0.148	0.135	0.240	0.133	0.165	3.823×10^{-2}*	8.529×10^{-3}*
F-ID AP	0.367	0.266	0.309	0.211	0.186	0.210	8.794×10^{-2}	9.959×10^{-3}*
R-ID AP	0.343	0.318	0.359	0.382	0.274	0.274	4.540×10^{-2}*	3.616×10^{-2}*
I-ID AP n-10	0.735	0.611	0.488	0.515	0.491	0.495	0.148	0.117
M-ID AP n-10	0.747	0.696	0.704	0.404	0.406	0.612	0.209	4.528×10^{-2}*
I-ID AP n-25	0.695	0.701	0.559	0.479	0.420	0.192	0.269	3.998×10^{-2}*
I-ID AP n-50	0.866	0.924	0.546	0.686	0.611	0.412	0.440	0.109
M-ID TP n-25		0.928	0.789	0.814	0.641	0.526	0.335	0.151
M-ID TP n-10	0.928		0.937	0.879	0.731	0.825	0.461	0.123
F-ID TP	0.789	0.937		0.979	0.886	0.922	0.535	0.153
I-ID TP n-10	0.814	0.879	0.979		0.947	0.932	0.443	0.193
M-ID AP n-25	0.641	0.731	0.886	0.947		0.982	0.567	0.236
M-ID TP n-50	0.526	0.825	0.922	0.932	0.982		0.634	0.307
R-ID TP	0.335	0.461	0.535	0.443	0.567	0.634		0.545
M-ID AP n-50	0.151	0.123	0.153	0.193	0.236	0.307	0.545	

Table A.11: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the Even 7 Parity problem after 100000 fitness evaluations.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
R-ID AP	1.820	4.839	0.684	42
I-ID TP n-10	2.160	5.991	0.847	42
F-ID TP	2.320	6.619	0.936	42
F-ID AP	2.640	6.382	0.903	42
R-ID TP	2.680	6.757	0.956	40
M-ID TP n-10	3.180	6.766	0.957	38
M-ID TP n-25	3.320	6.885	0.974	37
I-ID TP n-50	3.620	6.978	0.987	36
I-ID TP n-25	3.660	7.580	1.072	37
M-ID TP n-50	3.660	7.927	1.121	36
I-ID AP n-10	3.980	8.312	1.175	37
M-ID AP n-10	4.580	8.157	1.154	34
M-ID AP n-25	4.680	8.353	1.181	34
M-ID AP n-50	5.260	8.710	1.232	34
I-ID AP n-25	5.480	8.683	1.228	33
I-ID AP n-50	6.440	9.752	1.379	30

Table A.12: This table reports the p-value of Wilcoxon rank-sum tests performed on the average best fitness values of each approach after 100000 fitness evaluations on the Even 7 Parity problem. The p-values reported are calculated with a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	R-ID AP	I-ID TP n-10	F-ID TP	F-ID AP	R-ID TP	M-ID TP n-10	M-ID TP n-25	I-ID TP n-50
R-ID AP		0.865	0.856	0.426	0.660	0.297	0.227	0.177
I-ID TP n-10	0.865		1.000	0.622	0.551	0.400	0.355	0.234
F-ID TP	0.856	1.000		0.622	0.740	0.437	0.383	0.250
F-ID AP	0.426	0.622	0.622		1.000	0.831	0.711	0.684
R-ID TP	0.660	0.551	0.740	1.000		0.657	0.614	0.410
M-ID TP n-10	0.297	0.400	0.437	0.831	0.657		0.931	0.613
M-ID TP n-25	0.227	0.355	0.383	0.711	0.614	0.931		0.819
I-ID TP n-50	0.177	0.234	0.250	0.684	0.410	0.613	0.819	
I-ID TP n-25	0.139	0.137	0.302	0.680	0.419	0.807	0.726	0.943
M-ID TP n-50	0.204	0.184	0.330	0.546	0.359	0.964	0.988	0.796
I-ID AP n-10	0.130	0.267	0.285	0.465	0.331	0.795	0.764	0.963
M-ID AP n-10	6.130×10^{-2}	0.171	8.513×10^{-2}	0.204	0.166	0.439	0.316	0.709
M-ID AP n-25	1.243×10^{-2}*	0.102	0.102	0.248	0.171	0.265	0.394	0.620
M-ID AP n-50	2.249×10^{-2}*	2.450×10^{-2}*	6.718×10^{-2}	4.244×10^{-2}*	5.496×10^{-2}	0.216	0.213	0.295
I-ID AP n-25	1.004×10^{-2}*	2.288×10^{-2}*	4.233×10^{-2}*	4.451×10^{-2}*	9.812×10^{-2}	9.865×10^{-2}	0.172	0.293
I-ID AP n-50	6.457×10^{-3}*	8.494×10^{-3}*	1.570×10^{-2}*	3.969×10^{-2}*	1.452×10^{-2}*	6.233×10^{-2}	9.177×10^{-2}	0.146

	I-ID TP n-25	M-ID TP n-50	I-ID AP n-10	M-ID AP n-10	M-ID AP n-25	M-ID AP n-50	I-ID AP n-25	I-ID AP n-50
R-ID AP	0.139	0.204	0.130	6.130×10^{-2}	1.243×10^{-2}*	2.249×10^{-2}*	1.004×10^{-2}*	6.457×10^{-3}*
I-ID TP n-10	0.137	0.184	0.267	0.171	0.102	2.450×10^{-2}*	2.288×10^{-2}*	8.494×10^{-3}*
F-ID TP	0.302	0.330	0.285	8.513×10^{-2}	0.102	6.718×10^{-2}	4.233×10^{-2}*	1.570×10^{-2}*
F-ID AP	0.680	0.546	0.465	0.204	0.248	4.244×10^{-2}*	4.451×10^{-2}*	3.969×10^{-2}*
R-ID TP	0.419	0.359	0.331	0.166	0.171	5.496×10^{-2}	9.812×10^{-2}	1.452×10^{-2}*
M-ID TP n-10	0.807	0.964	0.795	0.439	0.265	0.216	9.865×10^{-2}	6.233×10^{-2}
M-ID TP n-25	0.726	0.988	0.764	0.316	0.394	0.213	0.172	9.177×10^{-2}
I-ID TP n-50	0.943	0.796	0.963	0.709	0.620	0.295	0.293	0.146
I-ID TP n-25		0.845	0.939	0.417	0.528	0.331	0.170	9.911×10^{-2}
M-ID TP n-50	0.845		0.819	0.489	0.399	0.240	0.234	8.932×10^{-2}
I-ID AP n-10	0.939	0.819		0.586	0.586	0.388	0.374	0.160
M-ID AP n-10	0.417	0.489	0.586		0.952	0.692	0.499	0.321
M-ID AP n-25	0.528	0.399	0.586	0.952		0.676	0.613	0.343
M-ID AP n-50	0.331	0.240	0.388	0.692	0.676		0.794	0.500
I-ID AP n-25	0.170	0.234	0.374	0.499	0.613	0.794		0.606
I-ID AP n-50	9.911×10^{-2}	8.932×10^{-2}	0.160	0.321	0.343	0.500	0.606	

Table A.13: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the $x^5 - 2x^3 + x$ Symbolic Regression problem after 100000 fitness evaluations.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
R-ID AP	0.284	0.459	0.065	13
F-ID AP	0.349	0.511	0.072	14
F-ID TP	0.410	0.504	0.071	12
I-ID TP n-25	0.417	0.446	0.063	13
I-ID TP n-10	0.429	0.453	0.064	11
I-ID AP n-10	0.479	0.483	0.068	13
R-ID TP	0.481	0.568	0.080	10
I-ID TP n-50	0.488	0.500	0.071	11
I-ID AP n-50	0.511	0.347	0.049	4
M-ID AP n-10	0.541	0.604	0.085	15
I-ID AP n-25	0.542	0.501	0.071	9
M-ID AP n-25	0.565	0.586	0.083	10
M-ID TP n-25	0.591	0.609	0.086	9
M-ID TP n-50	0.640	0.616	0.087	7
M-ID TP n-10	0.657	0.654	0.093	9
M-ID AP n-50	0.704	0.530	0.075	2

Table A.14: This table reports the p-value of Wilcoxon rank-sum tests performed on the average best fitness values of each approach after 100000 fitness evaluations on the $x^5 - 2x^3 + x$ Symbolic Regression problem. The p-values reported are calculated with a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	R-ID AP	F-ID AP	F-ID TP	I-ID TP n-25	I-ID TP n-10	I-ID AP n-10	R-ID TP	I-ID TP n-50
R-ID AP		0.493	0.107	9.208×10^{-2}	4.118×10^{-2}*	1.383×10^{-2}*	4.675×10^{-2}*	2.105×10^{-2}*
F-ID AP	0.493		0.244	0.406	0.107	6.943×10^{-2}	0.180	0.276
F-ID TP	0.107	0.244		0.851	0.985	0.388	0.612	0.348
I-ID TP n-25	9.208×10^{-2}	0.406	0.851		0.946	0.348	0.567	0.296
I-ID TP n-10	4.118×10^{-2}*	0.107	0.985	0.946		0.968	0.743	0.672
I-ID AP n-10	1.383×10^{-2}*	6.943×10^{-2}	0.388	0.348	0.968		0.657	0.765
R-ID TP	4.675×10^{-2}*	0.180	0.612	0.567	0.743	0.657		0.802
I-ID TP n-50	2.105×10^{-2}*	0.276	0.348	0.296	0.672	0.765	0.802	
I-ID AP n-50	8.088×10^{-4}*	5.025×10^{-3}*	4.465×10^{-2}*	6.030×10^{-2}	6.867×10^{-2}	0.125	9.557×10^{-2}	0.524
M-ID AP n-10	4.782×10^{-2}*	1.782×10^{-2}*	0.313	0.320	0.372	0.866	0.691	0.893
I-ID AP n-25	2.295×10^{-3}*	3.083×10^{-2}*	9.489×10^{-2}	0.200	0.153	0.652	0.183	0.800
M-ID AP n-25	1.224×10^{-2}*	1.874×10^{-2}*	0.122	9.762×10^{-2}	0.115	0.148	0.211	0.599
M-ID TP n-25	4.821×10^{-3}*	8.646×10^{-3}*	6.382×10^{-2}	7.266×10^{-2}	0.372	0.469	0.320	0.330
M-ID TP n-50	4.924×10^{-4}*	1.082×10^{-2}*	4.503×10^{-2}*	6.807×10^{-2}	4.944×10^{-2}*	0.194	5.596×10^{-2}	0.385
M-ID TP n-10	3.041×10^{-3}*	1.720×10^{-2}*	3.626×10^{-2}*	5.595×10^{-2}	0.145	0.172	0.193	0.255
M-ID AP n-50	2.138×10^{-6}*	1.633×10^{-4}*	1.373×10^{-3}*	4.335×10^{-4}*	2.874×10^{-5}*	3.316×10^{-3}*	3.339×10^{-3}*	1.049×10^{-2}*
	I-ID AP n-50	M-ID AP n-10	I-ID AP n-25	M-ID AP n-25	M-ID TP n-25	M-ID TP n-50	M-ID TP n-10	M-ID AP n-50
R-ID AP	8.088×10^{-4}*	4.782×10^{-2}*	2.295×10^{-3}*	1.224×10^{-2}*	4.821×10^{-3}*	4.924×10^{-4}*	3.041×10^{-3}*	2.138×10^{-6}*
F-ID AP	5.025×10^{-3}*	1.782×10^{-2}*	3.083×10^{-2}*	1.874×10^{-2}*	8.646×10^{-3}*	1.082×10^{-2}*	1.720×10^{-2}*	1.633×10^{-4}*
F-ID TP	4.465×10^{-2}*	0.313	9.489×10^{-2}	0.122	6.382×10^{-2}	4.503×10^{-2}*	3.626×10^{-2}*	1.373×10^{-3}*
I-ID TP n-25	6.030×10^{-2}	0.320	0.200	9.762×10^{-2}	7.266×10^{-2}	6.807×10^{-2}	5.595×10^{-2}	4.335×10^{-4}*
I-ID TP n-10	6.867×10^{-2}	0.372	0.153	0.115	0.372	4.944×10^{-2}*	0.145	2.874×10^{-5}*
I-ID AP n-10	0.125	0.866	0.652	0.148	0.469	0.194	0.172	3.316×10^{-3}*
R-ID TP	9.557×10^{-2}	0.691	0.183	0.211	0.320	5.596×10^{-2}	0.193	3.339×10^{-3}*
I-ID TP n-50	0.524	0.893	0.800	0.599	0.330	0.385	0.255	1.049×10^{-2}*
I-ID AP n-50		0.676	0.943	0.564	0.717	0.841	0.521	7.134×10^{-2}
M-ID AP n-10	0.676		0.762	0.882	0.885	0.415	0.798	4.447×10^{-2}*
I-ID AP n-25	0.943	0.762		0.963	0.908	0.866	0.913	8.535×10^{-2}
M-ID AP n-25	0.564	0.882	0.963		0.735	0.521	0.554	8.201×10^{-2}
M-ID TP n-25	0.717	0.885	0.908	0.735		0.689	0.901	6.889×10^{-2}
M-ID TP n-50	0.841	0.415	0.866	0.521	0.689		0.866	0.127
M-ID TP n-10	0.521	0.798	0.913	0.554	0.901	0.866		7.944×10^{-2}
M-ID AP n-50	7.134×10^{-2}	4.447×10^{-2}*	8.535×10^{-2}	8.201×10^{-2}	6.889×10^{-2}	0.127	7.944×10^{-2}	

Table A.15: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the 8×8 Lawn Mower problem after 100000 fitness evaluations.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
M-ID AP n-10	0.224	1.584	0.224	19
R-ID AP	0.462	2.287	0.323	20
M-ID TP n-10	0.464	2.299	0.325	21
M-ID AP n-25	0.492	2.435	0.344	19
M-ID TP n-50	0.506	2.510	0.355	17
M-ID TP n-25	1.188	3.608	0.510	21
R-ID TP	1.736	4.354	0.616	24
M-ID AP n-50	3.270	5.585	0.790	12
F-ID TP	5.966	6.894	0.975	15
F-ID AP	9.634	6.510	0.921	3
I-ID AP n-10	28.288	1.323	0.187	0
I-ID AP n-50	28.416	1.108	0.157	0
I-ID AP n-25	28.532	1.426	0.202	0
I-ID TP n-10	28.556	1.332	0.188	0
I-ID TP n-50	28.718	1.229	0.174	0
I-ID TP n-25	28.914	1.403	0.198	0

Table A.16: This table reports the p-value of Wilcoxon rank-sum tests performed on the average best fitness values of each approach after 100000 fitness evaluations on the 8×8 Lawn Mower problem. The p-values reported are calculated with a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	M-ID AP n-10	R-ID AP	M-ID TP n-10	M-ID AP n-25	M-ID TP n-50	M-ID TP n-25	R-ID TP	M-ID AP n-50
M-ID AP n-10		0.596	0.380	0.732	0.839	0.370	0.389	4.675×10^{-4}*
R-ID AP	0.596		0.705	0.431	0.290	0.964	0.658	5.771×10^{-3}*
M-ID TP n-10	0.380	0.705		0.391	0.401	0.414	0.402	4.367×10^{-3}*
M-ID AP n-25	0.732	0.431	0.391		0.694	0.406	0.426	4.438×10^{-4}*
M-ID TP n-50	0.839	0.289	0.403	0.694		0.305	0.398	1.635×10^{-3}*
M-ID TP n-25	0.370	0.964	0.414	0.406	0.305		0.757	7.178×10^{-3}*
R-ID TP	0.389	0.658	0.402	0.426	0.398	0.757		3.545×10^{-2}*
M-ID AP n-50	4.674×10^{-4}*	5.771×10^{-3}*	4.367×10^{-3}*	4.438×10^{-4}*	1.635×10^{-3}*	7.178×10^{-3}*	3.545×10^{-2}*	
F-ID TP	3.599×10^{-5}*	3.028×10^{-4}*	6.065×10^{-5}*	2.712×10^{-4}*	1.293×10^{-5}*	9.708×10^{-6}*	3.237×10^{-4}*	3.020×10^{-2}*
F-ID AP	1.140×10^{-8}*	9.083×10^{-8}*	1.205×10^{-8}*	7.244×10^{-8}*	2.421×10^{-8}*	1.466×10^{-8}*	1.816×10^{-7}*	3.272×10^{-5}*
I-ID AP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-50	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.800×10^{-10}*
I-ID AP n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID TP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID TP n-50	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID TP n-25	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*

	F-ID TP	F-ID AP	I-ID AP n-10	I-ID AP n-50	I-ID AP n-25	I-ID TP n-10	I-ID TP n-50	I-ID TP n-25
M-ID AP n-10	3.600×10^{-5}*	1.140×10^{-8}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
R-ID AP	3.028×10^{-4}*	9.083×10^{-8}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
M-ID TP n-10	6.065×10^{-5}*	1.205×10^{-8}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
M-ID AP n-25	2.712×10^{-4}*	7.244×10^{-8}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID TP n-50	1.293×10^{-5}*	2.421×10^{-8}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID TP n-25	9.708×10^{-6}*	1.466×10^{-8}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
R-ID TP	3.237×10^{-4}*	1.816×10^{-7}*	7.800×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
M-ID AP n-50	3.020×10^{-2}*	3.272×10^{-5}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID TP		2.221×10^{-3}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID AP	2.221×10^{-3}*		7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-10	7.800×10^{-10}*	7.800×10^{-10}*		0.515	0.487	0.434	7.045×10^{-2}	3.452×10^{-2}*
I-ID AP n-50	7.800×10^{-10}*	7.800×10^{-10}*	0.515		0.764	0.674	0.263	8.444×10^{-2}
I-ID AP n-25	7.800×10^{-10}*	7.800×10^{-10}*	0.487	0.764		0.891	0.518	0.312
I-ID TP n-10	7.800×10^{-10}*	7.800×10^{-10}*	0.434	0.674	0.891		0.616	0.224
I-ID TP n-50	7.800×10^{-10}*	7.800×10^{-10}*	7.045×10^{-2}	0.263	0.518	0.616		0.433
I-ID TP n-25	7.800×10^{-10}*	7.800×10^{-10}*	3.452×10^{-2}*	8.444×10^{-2}	0.312	0.224	0.433	

A.4 Initial Generation Identification

Table A.17: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the Santa Fe Ant Trail problem after 100000 fitness evaluations.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
F-ID TP G1	9.520	12.011	1.699	18
F-ID AP G1	11.480	13.278	1.878	21
R-ID TP G1	15.420	13.527	1.913	13
I-ID TP n-25	15.720	13.240	1.872	9
I-ID AP G1 n-10	15.820	14.951	2.114	10
I-ID AP G1 n-25	16.360	14.525	2.054	10
I-ID TP n-50	16.380	13.041	1.844	9
F-ID AP	16.760	13.283	1.878	12
R-ID AP	17.240	12.934	1.829	12
M-ID TP G1 n-50	17.340	14.662	2.073	12
M-ID TP G1 n-25	17.580	13.256	1.875	7
M-ID TP G1 n-10	17.840	12.921	1.827	11
M-ID AP G1 n-10	17.900	13.347	1.888	8
I-ID AP n-10	18.100	13.287	1.879	9
M-ID AP n-10	18.300	13.452	1.902	12
I-ID AP n-25	18.300	13.474	1.906	11
R-ID AP G1	18.600	12.779	1.807	8
I-ID TP G1 n-25	18.980	13.202	1.867	5
I-ID TP G1 n-10	19.140	13.398	1.895	6
M-ID TP n-25	19.220	13.196	1.866	8
M-ID TP n-10	19.260	14.294	2.021	9
I-ID TP G1 n-50	19.680	12.306	1.740	5
F-ID TP	19.700	12.960	1.833	8
I-ID TP n-10	19.800	12.51285	1.770	6
M-ID AP n-25	19.940	13.70745	1.939	6
M-ID TP n-50	20.060	13.85554	1.959	7
R-ID TP	21.300	12.91771	1.827	7
M-ID AP G1 n-25	21.320	14.764	2.088	7

Table A.18: This table reports the p-value of Wilcoxon rank-sum tests performed on the best fitness values of each approach after 100000 fitness evaluations on the Santa Fe A problem. The p-values reported are calculated with a confidence interval of 0.05. Values with an asterisk (*) are significant.

	F-ID TP G1	F-ID AP G1	R-ID TP G1	I-ID TP n-25	I-ID AP G1 n-10
F-ID TP G1		0.472	4.052×10^{-2}*	1.659×10^{-2}*	1.864×10^{-2}*
F-ID AP G1	0.472		7.572×10^{-2}	0.137	0.181
R-ID TP G1	4.052×10^{-2}*	7.572×10^{-2}		0.845	0.799
I-ID TP n-25	1.659×10^{-2}*	0.137	0.845		0.991
I-ID AP G1 n-10	1.864×10^{-2}*	0.181	0.799	0.991	
I-ID AP G1 n-25	3.307×10^{-2}*	2.807×10^{-2}*	0.784	0.891	0.886
I-ID TP n-50	9.065×10^{-3}*	7.263×10^{-2}	0.763	0.779	0.723
F-ID AP	7.638×10^{-3}*	8.108×10^{-2}	0.593	0.789	0.633
R-ID AP	5.060×10^{-3}*	2.351×10^{-2}*	0.439	0.698	0.582
M-ID TP G1 n-50	6.410×10^{-3}*	5.081×10^{-2}	0.564	0.604	0.579
M-ID TP G1 n-25	2.339×10^{-3}*	7.660×10^{-3}*	0.409	0.512	0.654
M-ID TP G1 n-10	1.396×10^{-3}*	2.465×10^{-2}*	0.337	0.465	0.455
M-ID AP G1 n-10	1.803×10^{-3}*	2.390×10^{-2}*	0.286	0.421	0.531
I-ID AP n-10	1.151×10^{-3}*	1.667×10^{-2}*	0.307	0.290	0.370
M-ID AP n-10	4.518×10^{-3}*	1.470×10^{-2}*	0.202	0.149	0.429
I-ID AP n-25	1.220×10^{-3}*	1.021×10^{-2}*	0.289	0.231	0.518
R-ID AP G1	1.761×10^{-3}*	5.455×10^{-3}*	0.183	0.264	0.259
I-ID TP G1 n-25	3.229×10^{-4}*	7.406×10^{-3}*	0.194	0.137	0.179
I-ID TP G1 n-10	1.766×10^{-4}*	6.503×10^{-3}*	7.025×10^{-2}	0.186	0.320
M-ID TP n-25	7.051×10^{-4}*	6.402×10^{-3}*	0.179	0.141	0.239
M-ID TP n-10	1.070×10^{-3}*	1.626×10^{-2}*	0.184	0.230	0.131
I-ID TP G1 n-50	1.227×10^{-5}*	5.263×10^{-3}*	0.122	0.182	0.162
F-ID TP	3.034×10^{-4}*	1.504×10^{-2}*	6.469×10^{-2}	0.161	0.218
I-ID TP n-10	6.909×10^{-4}*	3.518×10^{-3}*	0.110	0.119	0.299
M-ID AP n-25	1.354×10^{-5}*	2.687×10^{-3}*	8.256×10^{-2}	0.119	0.184
M-ID TP n-50	2.511×10^{-4}*	3.337×10^{-3}*	5.812×10^{-2}	0.107	0.168
R-ID TP	5.633×10^{-5}*	1.044×10^{-3}*	3.042×10^{-2}*	4.431×10^{-2}*	4.380×10^{-2}*
M-ID AP G1 n-25	6.475×10^{-5}*	3.347×10^{-3}*	3.540×10^{-2}*	3.490×10^{-2}*	7.625×10^{-2}

	I-ID AP G1 n-25	I-ID TP n-50	F-ID AP	R-ID AP	M-ID TP G1 n-50
F-ID TP G1	3.307×10^{-2}*	9.065×10^{-3}*	7.638×10^{-3}*	5.060×10^{-3}*	6.410×10^{-3}*
F-ID AP G1	2.807×10^{-2}*	7.263×10^{-2}	8.108×10^{-2}	2.351×10^{-2}*	5.081×10^{-2}
R-ID TP G1	0.784	0.763	0.593	0.439	0.564
I-ID TP n-25	0.891	0.779	0.789	0.698	0.604
I-ID AP G1 n-10	0.886	0.723	0.633	0.582	0.579
I-ID AP G1 n-25		0.952	0.804	0.611	0.731
I-ID TP n-50	0.952		0.896	0.883	0.693
F-ID AP	0.804	0.896		0.748	1.000
R-ID AP	0.611	0.883	0.748		0.996
M-ID TP G1 n-50	0.731	0.693	1.000	0.996	
M-ID TP G1 n-25	0.667	0.569	0.893	0.901	0.958
M-ID TP G1 n-10	0.481	0.638	0.658	0.806	0.881
M-ID AP G1 n-10	0.608	0.562	0.644	0.662	0.834
I-ID AP n-10	0.550	0.395	0.558	0.748	0.698
M-ID AP n-10	0.498	0.429	0.562	0.628	0.522
I-ID AP n-25	0.388	0.382	0.582	0.684	0.639
R-ID AP G1	0.418	0.377	0.436	0.899	0.435
I-ID TP G1 n-25	0.440	0.290	0.403	0.496	0.608
I-ID TP G1 n-10	0.253	0.155	0.472	0.471	0.662
M-ID TP n-25	0.207	0.302	0.367	0.343	0.508

Santa Fe Ant Trail Wilcoxon rank-sun test results continued.

M-ID TP n-10	0.317	0.148	0.266	0.318	0.382
I-ID TP G1 n-50	0.313	0.130	0.297	0.322	0.462
F-ID TP	0.253	0.135	0.309	0.359	0.360
I-ID TP n-10	0.273	0.240	0.211	0.382	0.395
M-ID AP n-25	0.156	0.133	0.186	0.274	0.340
M-ID TP n-50	7.797×10^{-2}	0.165	0.210	0.274	0.424
R-ID TP	8.795×10^{-2}	3.823×10^{-2}*	8.794×10^{-2}	4.540×10^{-2}*	0.229
M-ID AP G1 n-25	7.011×10^{-2}	6.002×10^{-2}	0.105	0.194	0.168

	M-ID TP G1 n-25	M-ID TP G1 n-10	M-ID AP G1 n-10	I-ID AP n-10	M-ID AP n-10
F-ID TP G1	2.339×10^{-3}*	1.396×10^{-3}*	1.803×10^{-3}*	1.151×10^{-3}*	4.518×10^{-3}*
F-ID AP G1	7.660×10^{-3}*	2.465×10^{-2}*	2.390×10^{-2}*	1.667×10^{-2}*	1.470×10^{-2}*
R-ID TP G1	0.409	0.337	0.286	0.307	0.202
I-ID TP n-25	0.512	0.465	0.421	0.290	0.149
I-ID AP G1 n-10	0.654	0.455	0.531	0.370	0.429
I-ID AP G1 n-25	0.667	0.481	0.608	0.550	0.498
I-ID TP n-50	0.569	0.638	0.562	0.395	0.429
F-ID AP	0.893	0.658	0.644	0.558	0.562
R-ID AP	0.901	0.806	0.662	0.748	0.628
M-ID TP G1 n-50	0.958	0.881	0.834	0.698	0.522
M-ID TP G1 n-25		0.906	0.770	0.750	0.715
M-ID TP G1 n-10	0.906		0.939	0.856	0.875
M-ID AP G1 n-10	0.770	0.939		0.902	0.992
I-ID AP n-10	0.750	0.856	0.902		0.992
M-ID AP n-10	0.715	0.875	0.992	0.992	
I-ID AP n-25	0.996	1.000	1.000	1.000	0.969
R-ID AP G1	0.548	0.797	0.839	0.973	0.861
I-ID TP G1 n-25	0.564	0.777	0.704	0.750	0.791
I-ID TP G1 n-10	0.522	0.312	0.648	0.727	0.804
M-ID TP n-25	0.458	0.530	0.607	0.735	0.747
M-ID TP n-10	0.597	0.529	0.708	0.611	0.696
I-ID TP G1 n-50	0.401	0.488	0.633	0.508	0.641
F-ID TP	0.440	0.460	0.473	0.488	0.704
I-ID TP n-10	0.294	0.409	0.484	0.515	0.404
M-ID AP n-25	0.114	0.299	0.372	0.491	0.406
M-ID TP n-50	0.355	0.351	0.505	0.495	0.612
R-ID TP	6.316×10^{-2}	4.159×10^{-2}*	0.191	0.148	0.209
M-ID AP G1 n-25	0.218	0.145	0.242	0.352	0.336

	I-ID AP n-25	R-ID AP G1	I-ID TP G1 n-25	I-ID TP G1 n-10	M-ID TP n-25
F-ID TP G1	1.220×10^{-3}*	1.761×10^{-3}*	3.229×10^{-4}*	1.766×10^{-4}*	7.051×10^{-4}*
F-ID AP G1	1.021×10^{-2}*	5.455×10^{-3}*	7.406×10^{-3}*	6.503×10^{-3}*	6.402×10^{-3}*
R-ID TP G1	0.289	0.183	0.194	7.025×10^{-2}	0.179
I-ID TP n-25	0.231	0.264	0.137	0.186	0.141
I-ID AP G1 n-10	0.518	0.259	0.179	0.320	0.239
I-ID AP G1 n-25	0.388	0.418	0.440	0.253	0.207
I-ID TP n-50	0.382	0.377	0.290	0.155	0.302
F-ID AP	0.582	0.436	0.403	0.472	0.367
R-ID AP	0.684	0.899	0.496	0.471	0.343
M-ID TP G1 n-50	0.639	0.435	0.608	0.662	0.508
M-ID TP G1 n-25	0.996	0.548	0.564	0.522	0.458
M-ID TP G1 n-10	1.000	0.797	0.777	0.312	0.530
M-ID AP G1 n-10	1.000	0.839	0.704	0.648	0.607
I-ID AP n-10	1.000	0.973	0.750	0.727	0.735
M-ID AP n-10	0.969	0.861	0.791	0.804	0.747
I-ID AP n-25		0.794	0.932	0.538	0.695

Santa Fe Ant Trail Wilcoxon rank-sun test results continued.

R-ID AP G1	0.794		0.955	0.913	0.858
I-ID TP G1 n-25	0.932	0.955		0.987	0.960
I-ID TP G1 n-10	0.538	0.913	0.987		0.739
M-ID TP n-25	0.695	0.858	0.960	0.739	
M-ID TP n-10	0.701	0.799	0.918	0.862	0.928
I-ID TP G1 n-50	0.487	0.672	0.548	0.812	0.913
F-ID TP	0.559	0.750	0.691	0.750	0.789
I-ID TP n-10	0.479	0.805	0.696	0.772	0.814
M-ID AP n-25	0.420	0.636	0.605	0.739	0.641
M-ID TP n-50	0.192	0.449	0.604	0.773	0.526
R-ID TP	0.269	0.178	0.322	0.478	0.335
M-ID AP G1 n-25	0.234	0.269	0.415	0.385	0.381

	M-ID TP n-10	I-ID TP G1 n-50	F-ID TP	I-ID TP n-10	M-ID AP n-25
F-ID TP G1	1.070×10^{-3}*	1.227×10^{-5}*	3.034×10^{-4}*	6.909×10^{-4}*	1.354×10^{-5}*
F-ID AP G1	1.626×10^{-2}*	5.263×10^{-3}*	1.504×10^{-2}*	3.518×10^{-3}*	2.687×10^{-3}*
R-ID TP G1	0.184	0.122	6.469×10^{-2}	0.110	8.256×10^{-2}
I-ID TP n-25	0.230	0.182	0.161	0.119	0.119
I-ID AP G1 n-10	0.131	0.162	0.218	0.299	0.184
I-ID AP G1 n-25	0.317	0.313	0.253	0.273	0.156
I-ID TP n-50	0.148	0.130	0.135	0.240	0.133
F-ID AP	0.266	0.297	0.309	0.211	0.186
R-ID AP	0.318	0.322	0.359	0.382	0.274
M-ID TP G1 n-50	0.382	0.462	0.360	0.395	0.340
M-ID TP G1 n-25	0.597	0.401	0.440	0.294	0.114
M-ID TP G1 n-10	0.529	0.488	0.460	0.409	0.299
M-ID AP G1 n-10	0.708	0.633	0.473	0.484	0.372
I-ID AP n-10	0.611	0.508	0.488	0.515	0.491
M-ID AP n-10	0.696	0.641	0.704	0.404	0.406
I-ID AP n-25	0.701	0.487	0.559	0.479	0.420
R-ID AP G1	0.799	0.672	0.750	0.805	0.636
I-ID TP G1 n-25	0.918	0.548	0.691	0.696	0.605
I-ID TP G1 n-10	0.862	0.812	0.750	0.772	0.739
M-ID TP n-25	0.928	0.913	0.789	0.814	0.641
M-ID TP n-10		0.894	0.937	0.879	0.731
I-ID TP G1 n-50	0.894		0.585	0.898	0.800
F-ID TP	0.937	0.585		0.979	0.886
I-ID TP n-10	0.879	0.898	0.979		0.947
M-ID AP n-25	0.731	0.800	0.886	0.947	
M-ID TP n-50	0.825	0.978	0.922	0.932	0.982
R-ID TP	0.461	0.459	0.535	0.443	0.567
M-ID AP G1 n-25	0.448	0.546	0.481	0.629	0.380

	M-ID TP n-50	R-ID TP	M-ID AP G1 n-25
F-ID TP G1	2.511×10^{-4}*	5.633×10^{-5}*	6.475×10^{-5}*
F-ID AP G1	3.337×10^{-3}*	1.044×10^{-3}*	3.347×10^{-3}*
R-ID TP G1	5.812×10^{-2}	3.042×10^{-2}*	3.540×10^{-2}*
I-ID TP n-25	0.107	4.431×10^{-2}*	3.490×10^{-2}*
I-ID AP G1 n-10	0.168	4.380×10^{-2}*	7.625×10^{-2}
I-ID AP G1 n-25	7.797×10^{-2}	8.795×10^{-2}	7.011×10^{-2}
I-ID TP n-50	0.165	3.823×10^{-2}*	6.002×10^{-2}
F-ID AP	0.210	8.794×10^{-2}	0.105
R-ID AP	0.274	4.540×10^{-2}*	0.194
M-ID TP G1 n-50	0.424	0.229	0.168
M-ID TP G1 n-25	0.355	6.316×10^{-2}	0.218
M-ID TP G1 n-10	0.351	4.159×10^{-2}*	0.145

Santa Fe Ant Trail Wilcoxon rank-sun test results continued.

M-ID AP G1 n-10	0.505	0.191	0.242
I-ID AP n-10	0.495	0.148	0.352
M-ID AP n-10	0.612	0.209	0.336
I-ID AP n-25	0.192	0.269	0.234
R-ID AP G1	0.449	0.178	0.269
I-ID TP G1 n-25	0.604	0.322	0.415
I-ID TP G1 n-10	0.773	0.478	0.385
M-ID TP n-25	0.526	0.335	0.381
M-ID TP n-10	0.825	0.461	0.448
I-ID TP G1 n-50	0.978	0.459	0.546
F-ID TP	0.922	0.535	0.481
I-ID TP n-10	0.932	0.443	0.629
M-ID AP n-25	0.982	0.567	0.380
M-ID TP n-50		0.634	0.573
R-ID TP	0.634		0.781
M-ID AP G1 n-25	0.573	0.781	

Table A.19: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the Even 7 Parity problem after 100000 fitness evaluations.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
F-ID AP G1	0.480	2.082	0.294	47
I-ID TP G1 n-50	0.640	2.724	0.385	47
F-ID TP G1	0.720	2.990	0.423	47
R-ID AP G1	0.880	2.715	0.384	45
M-ID TP G1 n-10	1.040	3.664	0.518	45
M-ID TP G1 n-25	1.460	3.871	0.548	42
I-ID TP G1 n-25	1.600	6.465	0.914	46
M-ID TP G1 n-50	1.760	5.723	0.809	44
I-ID AP G1 n-10	1.760	5.947	0.841	44
R-ID AP	1.820	4.839	0.684	42
R-ID TP G1	2.000	6.168	0.872	44
M-ID AP G1 n-10	2.140	5.292	0.748	41
I-ID TP n-10	2.160	5.991	0.847	42
F-ID TP	2.320	6.619	0.936	42
M-ID AP G1 n-25	2.420	6.101	0.863	42
F-ID AP	2.640	6.382	0.903	42
R-ID TP	2.680	6.757	0.956	40
I-ID AP G1 n-25	2.760	6.675	0.944	41
I-ID TP G1 n-10	2.780	6.287	0.889	39
M-ID TP n-10	3.180	6.766	0.957	38
M-ID TP n-25	3.320	6.885	0.974	37
I-ID TP n-50	3.620	6.978	0.987	36
I-ID TP n-25	3.660	7.580	1.072	37
M-ID TP n-50	3.660	7.927	1.121	36
I-ID AP n-10	3.980	8.312	1.175	37
M-ID AP n-10	4.580	8.157	1.154	34
M-ID AP n-25	4.680	8.353	1.181	34
I-ID AP n-25	5.480	8.683	1.228	33

Table A.20: This table reports the p-value of Wilcoxon rank-sum tests performed on the av
best fitness values of each approach after 100000 fitness evaluations on the Even 7 Parity pro
The p-values reported are calculated with a confidence interval of 0.05. Values marked w
asterisk (*) are significant.

	F-ID AP G1	I-ID TP G1 n-50	F-ID TP G1	R-ID AP G1	M-ID TP G1 n-10
F-ID AP G1		0.832	0.673	0.395	0.352
I-ID TP G1 n-50	0.832		1.000	0.714	0.606
F-ID TP G1	0.673	1.000		1.000	0.670
R-ID AP G1	0.395	0.714	1.000		0.903
M-ID TP G1 n-10	0.352	0.606	0.670	0.903	
M-ID TP G1 n-25	0.153	0.303	0.284	0.548	0.554
I-ID TP G1 n-25	0.443	0.396	0.670	0.903	0.905
M-ID TP G1 n-50	0.172	0.202	0.259	0.440	0.574
I-ID AP G1 n-10	0.234	0.291	0.349	0.587	0.609
R-ID AP	0.109	0.181	0.262	0.308	0.346
R-ID TP G1	7.636×10^{-2}	0.184	0.185	0.189	0.500
M-ID AP G1 n-10	6.371×10^{-2}	0.132	0.108	0.227	0.324
I-ID TP n-10	7.479×10^{-2}	0.166	0.196	0.246	0.401
F-ID TP	8.924×10^{-2}	0.151	0.210	0.270	0.342
M-ID AP G1 n-25	3.977×10^{-2}*	5.627×10^{-2}	7.913×10^{-2}	0.123	0.141
F-ID AP	1.207×10^{-2}*	5.851×10^{-2}	6.685×10^{-2}	6.767×10^{-2}	0.134
R-ID TP	4.580×10^{-2}*	8.570×10^{-2}	0.107	0.138	0.190
I-ID AP G1 n-25	2.579×10^{-2}*	1.353×10^{-2}*	7.585×10^{-2}	5.639×10^{-2}	0.145
I-ID TP G1 n-10	2.012×10^{-2}*	2.601×10^{-2}*	6.711×10^{-2}	6.028×10^{-2}	9.990×10^{-2}
M-ID TP n-10	1.229×10^{-2}*	2.615×10^{-2}*	2.090×10^{-2}*	3.744×10^{-2}*	6.967×10^{-2}
M-ID TP n-25	9.604×10^{-3}*	2.425×10^{-2}*	2.960×10^{-2}*	2.946×10^{-2}*	5.790×10^{-2}
I-ID TP n-50	5.432×10^{-3}*	1.161×10^{-2}*	1.016×10^{-2}*	1.526×10^{-2}*	4.126×10^{-2}*
I-ID TP n-25	5.038×10^{-3}*	1.860×10^{-2}*	1.499×10^{-2}*	2.077×10^{-2}*	1.576×10^{-2}*
M-ID TP n-50	1.107×10^{-2}*	1.103×10^{-2}*	8.059×10^{-3}*	4.410×10^{-2}*	5.034×10^{-2}
I-ID AP n-10	6.877×10^{-3}*	1.434×10^{-2}*	1.814×10^{-2}*	2.738×10^{-2}*	3.684×10^{-2}*
M-ID AP n-10	1.984×10^{-3}*	2.749×10^{-3}*	1.239×10^{-3}*	5.795×10^{-3}*	1.847×10^{-2}*
M-ID AP n-25	7.835×10^{-4}*	4.754×10^{-3}*	5.622×10^{-3}*	5.306×10^{-3}*	8.307×10^{-3}*
I-ID AP n-25	4.910×10^{-4}*	3.931×10^{-4}*	5.906×10^{-4}*	1.028×10^{-3}*	1.230×10^{-3}*

	M-ID TP G1 n-25	I-ID TP G1 n-25	M-ID TP G1 n-50	I-ID AP G1 n-10	R-ID AP
F-ID AP G1	0.153	0.443	0.172	0.234	0.109
I-ID TP G1 n-50	0.303	0.396	0.202	0.291	0.181
F-ID TP G1	0.284	0.670	0.259	0.349	0.262
R-ID AP G1	0.548	0.903	0.440	0.587	0.308
M-ID TP G1 n-10	0.554	0.905	0.574	0.609	0.346
M-ID TP G1 n-25		0.723	0.972	0.916	0.711
I-ID TP G1 n-25	0.723		0.758	0.837	0.646
M-ID TP G1 n-50	0.972	0.758		0.937	0.925
I-ID AP G1 n-10	0.916	0.837	0.937		0.777
R-ID AP	0.711	0.646	0.925	0.777	
R-ID TP G1	0.725	0.681	0.781	0.783	0.844
M-ID AP G1 n-10	0.528	0.481	0.623	0.592	0.836
I-ID TP n-10	0.431	0.475	0.753	0.674	0.865
F-ID TP	0.551	0.501	0.725	0.636	0.856
M-ID AP G1 n-25	0.295	0.474	0.503	0.488	0.508
F-ID AP	0.319	0.284	0.455	0.401	0.426
R-ID TP	0.419	0.381	0.314	0.378	0.660
I-ID AP G1 n-25	0.252	0.269	0.194	0.347	0.484
I-ID TP G1 n-10	0.200	0.198	0.311	0.298	0.420
M-ID TP n-10	0.204	0.131	0.234	0.218	0.297

Even 7 Parity Wilcoxon rank-sun test results continued.

M-ID TP n-25	5.772×10^{-2}	0.129	0.184	9.298×10^{-2}	0.227
I-ID TP n-50	8.936×10^{-2}	6.729×10^{-2}	0.184	0.116	0.177
I-ID TP n-25	9.607×10^{-2}	0.103	0.161	6.447×10^{-2}	0.139
M-ID TP n-50	0.158	8.225×10^{-2}	0.184	9.263×10^{-2}	0.204
I-ID AP n-10	8.651×10^{-2}	9.671×10^{-2}	0.111	0.112	0.130
M-ID AP n-10	7.934×10^{-3}*	4.197×10^{-2}*	3.214×10^{-2}*	3.576×10^{-2}*	6.130×10^{-2}
M-ID AP n-25	1.310×10^{-2}*	3.793×10^{-2}*	4.964×10^{-2}*	4.731×10^{-2}*	1.243×10^{-2}*
I-ID AP n-25	8.320×10^{-3}*	2.192×10^{-2}*	1.564×10^{-2}*	1.359×10^{-2}*	1.004×10^{-2}*

	R-ID TP G1	M-ID AP G1 n-10	I-ID TP n-10	F-ID TP	M-ID AP G1 n-25
F-ID AP G1	7.636×10^{-2}	6.371×10^{-2}	7.479×10^{-2}	8.924×10^{-2}	3.977×10^{-2}*
I-ID TP G1 n-50	0.184	0.132	0.166	0.151	5.627×10^{-2}
F-ID TP G1	0.185	0.108	0.196	0.210	7.913×10^{-2}
R-ID AP G1	0.189	0.227	0.246	0.270	0.123
M-ID TP G1 n-10	0.500	0.324	0.401	0.342	0.141
M-ID TP G1 n-25	0.725	0.528	0.431	0.551	0.295
I-ID TP G1 n-25	0.681	0.481	0.475	0.501	0.474
M-ID TP G1 n-50	0.781	0.623	0.753	0.725	0.503
I-ID AP G1 n-10	0.783	0.592	0.674	0.636	0.488
R-ID AP	0.844	0.836	0.865	0.856	0.508
R-ID TP G1		0.863	1.000	0.950	0.758
M-ID AP G1 n-10	0.863		0.955	0.864	0.796
I-ID TP n-10	1.000	0.955		1.000	0.776
F-ID TP	0.950	0.864	1.000		0.864
M-ID AP G1 n-25	0.758	0.796	0.776	0.864	
F-ID AP	0.752	0.641	0.622	0.622	0.917
R-ID TP	0.614	0.831	0.551	0.740	0.981
I-ID AP G1 n-25	0.590	0.568	0.550	0.640	0.695
I-ID TP G1 n-10	0.755	0.568	0.521	0.515	0.754
M-ID TP n-10	0.324	0.367	0.400	0.437	0.526
M-ID TP n-25	0.275	0.327	0.355	0.383	0.492
I-ID TP n-50	0.196	0.168	0.234	0.250	0.347
I-ID TP n-25	0.380	0.358	0.137	0.302	0.547
M-ID TP n-50	0.285	0.211	0.184	0.330	0.456
I-ID AP n-10	0.182	0.227	0.267	0.285	0.373
M-ID AP n-10	0.116	9.023×10^{-2}	0.171	8.513×10^{-2}	0.168
M-ID AP n-25	8.080×10^{-2}	7.302×10^{-2}	0.102	0.102	0.117
I-ID AP n-25	3.481×10^{-2}*	2.400×10^{-2}*	2.288×10^{-2}*	4.233×10^{-2}*	3.205×10^{-2}*

	F-ID AP	R-ID TP	I-ID AP G1 n-25	I-ID TP G1 n-10	M-ID TP n-10
F-ID AP G1	1.207×10^{-2}*	4.580×10^{-2}*	2.579×10^{-2}*	2.012×10^{-2}*	1.229×10^{-2}*
I-ID TP G1 n-50	5.851×10^{-2}	8.570×10^{-2}	1.353×10^{-2}*	2.601×10^{-2}*	2.615×10^{-2}*
F-ID TP G1	6.685×10^{-2}	0.107	7.585×10^{-2}	6.711×10^{-2}	2.090×10^{-2}*
R-ID AP G1	6.767×10^{-2}	0.138	5.639×10^{-2}	6.028×10^{-2}	3.744×10^{-2}*
M-ID TP G1 n-10	0.134	0.190	0.145	9.990×10^{-2}	6.967×10^{-2}
M-ID TP G1 n-25	0.319	0.419	0.252	0.200	0.204
I-ID TP G1 n-25	0.284	0.381	0.269	0.198	0.131
M-ID TP G1 n-50	0.455	0.314	0.194	0.311	0.234
I-ID AP G1 n-10	0.401	0.378	0.347	0.298	0.218
R-ID AP	0.426	0.660	0.484	0.420	0.297
R-ID TP G1	0.752	0.614	0.590	0.755	0.324
M-ID AP G1 n-10	0.641	0.831	0.568	0.568	0.367
I-ID TP n-10	0.622	0.551	0.550	0.521	0.400
F-ID TP	0.622	0.740	0.640	0.515	0.437
M-ID AP G1 n-25	0.917	0.981	0.695	0.754	0.526
F-ID AP		1.000	0.975	0.981	0.831

Even 7 Parity Wilcoxon rank-sun test results continued.

R-ID TP	1.000		0.955	0.777	0.657
I-ID AP G1 n-25	0.975	0.955		0.981	0.887
I-ID TP G1 n-10	0.981	0.777	0.981		0.762
M-ID TP n-10	0.831	0.657	0.887	0.762	
M-ID TP n-25	0.711	0.614	0.745	0.626	0.931
I-ID TP n-50	0.684	0.410	0.517	0.473	0.613
I-ID TP n-25	0.680	0.419	0.726	0.668	0.807
M-ID TP n-50	0.546	0.359	0.485	0.625	0.964
I-ID AP n-10	0.465	0.331	0.467	0.513	0.795
M-ID AP n-10	0.204	0.166	0.285	0.315	0.439
M-ID AP n-25	0.248	0.171	0.296	0.280	0.265
I-ID AP n-25	4.451×10^{-2}*	9.812×10^{-2}	0.106	8.085×10^{-2}	9.865×10^{-2}

	M-ID TP n-25	I-ID TP n-50	I-ID TP n-25	M-ID TP n-50	I-ID AP n-10
F-ID AP G1	9.604×10^{-3}*	5.432×10^{-3}*	5.038×10^{-3}*	1.107×10^{-2}*	6.877×10^{-3}*
I-ID TP G1 n-50	2.425×10^{-2}*	1.161×10^{-2}*	1.860×10^{-2}*	1.103×10^{-2}*	1.434×10^{-2}*
F-ID TP G1	2.960×10^{-2}*	1.016×10^{-2}*	1.499×10^{-2}*	8.059×10^{-3}*	1.814×10^{-2}*
R-ID AP G1	2.946×10^{-2}*	1.526×10^{-2}*	2.077×10^{-2}*	4.410×10^{-2}*	2.738×10^{-2}*
M-ID TP G1 n-10	5.790×10^{-2}	4.126×10^{-2}*	1.576×10^{-2}*	5.034×10^{-2}	3.684×10^{-2}*
M-ID TP G1 n-25	5.772×10^{-2}	8.936×10^{-2}	9.607×10^{-2}	0.158	8.651×10^{-2}
I-ID TP G1 n-25	0.129	6.729×10^{-2}	0.103	8.225×10^{-2}	9.671×10^{-2}
M-ID TP G1 n-50	0.184	0.184	0.161	0.184	0.111
I-ID AP G1 n-10	9.298×10^{-2}	0.116	6.447×10^{-2}	9.263×10^{-2}	0.112
R-ID AP	0.227	0.177	0.139	0.204	0.130
R-ID TP G1	0.275	0.196	0.380	0.285	0.182
M-ID AP G1 n-10	0.327	0.168	0.358	0.211	0.227
I-ID TP n-10	0.355	0.234	0.137	0.184	0.267
F-ID TP	0.383	0.250	0.302	0.330	0.285
M-ID AP G1 n-25	0.492	0.347	0.547	0.456	0.373
F-ID AP	0.711	0.684	0.680	0.546	0.465
R-ID TP	0.614	0.410	0.419	0.359	0.331
I-ID AP G1 n-25	0.745	0.517	0.726	0.485	0.467
I-ID TP G1 n-10	0.626	0.473	0.668	0.625	0.513
M-ID TP n-10	0.931	0.613	0.807	0.964	0.795
M-ID TP n-25		0.819	0.726	0.988	0.764
I-ID TP n-50	0.819		0.943	0.796	0.963
I-ID TP n-25	0.726	0.943		0.845	0.939
M-ID TP n-50	0.988	0.796	0.845		0.819
I-ID AP n-10	0.764	0.963	0.939	0.819	
M-ID AP n-10	0.316	0.709	0.417	0.489	0.586
M-ID AP n-25	0.394	0.620	0.528	0.399	0.586
I-ID AP n-25	0.172	0.293	0.170	0.234	0.374

	M-ID AP n-10	M-ID AP n-25	I-ID AP n-25
F-ID AP G1	1.984×10^{-3}*	7.835×10^{-4}*	4.910×10^{-4}*
I-ID TP G1 n-50	2.749×10^{-3}*	4.754×10^{-3}*	3.931×10^{-4}*
F-ID TP G1	1.239×10^{-3}*	5.622×10^{-3}*	5.906×10^{-4}*
R-ID AP G1	5.795×10^{-3}*	5.306×10^{-3}*	1.028×10^{-3}*
M-ID TP G1 n-10	1.847×10^{-2}*	8.307×10^{-3}*	1.230×10^{-3}*
M-ID TP G1 n-25	7.934×10^{-3}*	1.310×10^{-2}*	8.320×10^{-3}*
I-ID TP G1 n-25	4.197×10^{-2}*	3.793×10^{-2}*	2.192×10^{-2}*
M-ID TP G1 n-50	3.214×10^{-2}*	4.964×10^{-2}*	1.564×10^{-2}*
I-ID AP G1 n-10	3.576×10^{-2}*	4.731×10^{-2}*	1.359×10^{-2}*
R-ID AP	6.130×10^{-2}	1.243×10^{-2}*	1.004×10^{-2}*
R-ID TP G1	0.116	8.080×10^{-2}	3.481×10^{-2}*
M-ID AP G1 n-10	9.023×10^{-2}	7.302×10^{-2}	2.400×10^{-2}*

Even 7 Parity Wilcoxon rank-sun test results continued.

I-ID TP n-10	0.171	0.102	2.288×10^{-2}*
F-ID TP	8.513×10^{-2}	0.102	4.233×10^{-2}*
M-ID AP G1 n-25	0.168	0.117	3.205×10^{-2}*
F-ID AP	0.204	0.248	4.451×10^{-2}*
R-ID TP	0.166	0.171	9.812×10^{-2}
I-ID AP G1 n-25	0.285	0.296	0.106
I-ID TP G1 n-10	0.315	0.280	8.085×10^{-2}
M-ID TP n-10	0.439	0.265	9.865×10^{-2}
M-ID TP n-25	0.316	0.394	0.172
I-ID TP n-50	0.709	0.620	0.293
I-ID TP n-25	0.417	0.528	0.170
M-ID TP n-50	0.489	0.399	0.234
I-ID AP n-10	0.586	0.586	0.374
M-ID AP n-10		0.952	0.499
M-ID AP n-25	0.952		0.613
I-ID AP n-25	0.499	0.613	

Table A.21: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the $x^5 - 2x^3 + x$ Symbolic Regression problem after 100000 fitness evaluations.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
R-ID AP	0.284	0.459	0.065	13
R-ID TP G1	0.293	0.433	0.061	13
F-ID AP	0.349	0.511	0.072	14
F-ID TP	0.410	0.504	0.071	12
I-ID TP n-25	0.417	0.446	0.063	13
F-ID TP G1	0.425	0.494	0.070	8
I-ID TP n-10	0.429	0.453	0.064	11
I-ID TP G1 n-50	0.471	0.375	0.053	8
F-ID AP G1	0.473	0.474	0.067	5
I-ID AP n-10	0.479	0.483	0.068	13
R-ID TP	0.481	0.568	0.080	10
R-ID AP G1	0.483	0.681	0.096	12
I-ID TP n-50	0.488	0.500	0.071	11
M-ID TP G1 n-10	0.509	0.485	0.069	9
M-ID TP G1 n-25	0.534	0.506	0.072	4
I-ID TP G1 n-25	0.538	0.453	0.064	7
I-ID AP G1 n-25	0.540	0.448	0.063	6
M-ID AP n-10	0.541	0.604	0.085	15
I-ID AP n-25	0.542	0.501	0.071	9
M-ID AP n-25	0.565	0.586	0.083	10
M-ID TP G1 n-50	0.588	0.591	0.084	10
M-ID TP n-25	0.591	0.609	0.086	9
I-ID TP G1 n-10	0.620	0.546	0.077	5
I-ID AP G1 n-10	0.627	0.587	0.083	6
M-ID TP n-50	0.640	0.616	0.087	7
M-ID TP n-10	0.657	0.654	0.093	9
M-ID AP G1 n-10	0.721	0.629	0.089	5
M-ID AP G1 n-25	0.937	0.640	0.090	1

Table A.22: This table reports the p-value of Wilcoxon rank-sum tests performe
best fitness values of each approach after 100000 fitness evaluations on the Even
The p-values reported are calculated with a confidence interval of 0.05. Values
asterisk (*) are significant.

	R-ID AP	R-ID TP G1	F-ID AP	F-ID TP	I-ID TP n-25
R-ID AP		0.900	0.493	0.107	9.208×10^{-2}
R-ID TP G1	0.900		0.931	0.235	7.412×10^{-2}
F-ID AP	0.493	0.931		0.244	0.406
F-ID TP	0.107	0.235	0.244		0.851
I-ID TP n-25	9.208×10^{-2}	7.412×10^{-2}	0.406	0.851	
F-ID TP G1	3.211×10^{-2}*	0.215	0.179	0.640	0.969
I-ID TP n-10	4.118×10^{-2}*	5.784×10^{-2}	0.107	0.985	0.946
I-ID TP G1 n-50	1.613×10^{-3}*	1.097×10^{-2}*	3.827×10^{-2}*	0.108	0.246
F-ID AP G1	9.410×10^{-3}*	1.941×10^{-2}*	2.441×10^{-2}*	0.133	0.271
I-ID AP n-10	1.383×10^{-2}*	9.301×10^{-2}	6.943×10^{-2}	0.388	0.348
R-ID TP	4.675×10^{-2}*	1.310×10^{-2}*	0.180	0.612	0.567
R-ID AP G1	0.208	0.395	0.662	0.892	0.585
I-ID TP n-50	2.105×10^{-2}*	3.410×10^{-2}*	0.276	0.348	0.296
M-ID TP G1 n-10	2.007×10^{-3}*	4.042×10^{-3}*	1.966×10^{-2}*	0.113	0.275
M-ID TP G1 n-25	1.291×10^{-3}*	5.463×10^{-4}*	2.480×10^{-2}*	7.413×10^{-2}	4.502×10^{-2}*
I-ID TP G1 n-25	5.684×10^{-4}*	3.352×10^{-4}*	5.178×10^{-3}*	4.501×10^{-2}*	0.129
I-ID AP G1 n-25	2.795×10^{-5}*	1.030×10^{-3}*	1.328×10^{-3}*	4.651×10^{-3}*	4.291×10^{-2}*
M-ID AP n-10	4.782×10^{-2}*	6.579×10^{-3}*	1.782×10^{-2}*	0.313	0.320
I-ID AP n-25	2.295×10^{-3}*	2.113×10^{-3}*	3.083×10^{-2}*	9.489×10^{-2}	0.200
M-ID AP n-25	1.224×10^{-2}*	6.482×10^{-3}*	1.874×10^{-2}*	0.122	9.762×10^{-2}
M-ID TP G1 n-50	1.912×10^{-3}*	3.853×10^{-3}*	5.911×10^{-2}	6.449×10^{-2}	0.106
M-ID TP n-25	4.821×10^{-3}*	1.374×10^{-3}*	8.646×10^{-3}*	6.382×10^{-2}	7.266×10^{-2}
I-ID TP G1 n-10	5.022×10^{-5}*	7.153×10^{-4}*	2.453×10^{-3}*	5.600×10^{-3}*	1.134×10^{-2}*
I-ID AP G1 n-10	1.001×10^{-4}*	1.028×10^{-3}*	2.637×10^{-3}*	1.401×10^{-2}*	1.898×10^{-2}*
M-ID TP n-50	4.924×10^{-4}*	4.456×10^{-4}*	1.082×10^{-2}*	4.503×10^{-2}*	6.807×10^{-2}
M-ID TP n-10	3.041×10^{-3}*	4.713×10^{-4}*	1.720×10^{-2}*	3.626×10^{-2}*	5.595×10^{-2}
M-ID AP G1 n-10	6.363×10^{-5}*	1.936×10^{-5}*	3.000×10^{-4}*	1.482×10^{-3}*	2.639×10^{-2}*
M-ID AP G1 n-25	2.660×10^{-6}*	4.811×10^{-7}*	1.980×10^{-5}*	2.853×10^{-5}*	1.544×10^{-6}*

	F-ID TP G1	I-ID TP n-10	I-ID TP G1 n-50	F-ID AP G1	I-ID AP n-10
R-ID AP	3.211×10^{-2}*	4.118×10^{-2}*	1.613×10^{-3}*	9.410×10^{-3}*	1.383×10^{-2}*
R-ID TP G1	0.215	5.784×10^{-2}	1.097×10^{-2}*	1.941×10^{-2}*	9.301×10^{-2}
F-ID AP	0.179	0.107	3.827×10^{-2}*	2.441×10^{-2}*	6.943×10^{-2}
F-ID TP	0.640	0.985	0.108	0.133	0.388
I-ID TP n-25	0.969	0.946	0.246	0.271	0.348
F-ID TP G1		0.772	0.111	0.385	0.598
I-ID TP n-10	0.772		0.283	0.398	0.968
I-ID TP G1 n-50	0.111	0.283		0.900	0.345
F-ID AP G1	0.385	0.398	0.900		1.000
I-ID AP n-10	0.598	0.968	0.345	1.000	
R-ID TP	0.931	0.743	0.354	0.724	0.657
R-ID AP G1	0.889	0.996	0.155	0.382	0.499
I-ID TP n-50	0.315	0.672	0.996	0.877	0.765
M-ID TP G1 n-10	0.299	0.438	1.000	0.728	0.597
M-ID TP G1 n-25	5.875×10^{-2}	7.827×10^{-2}	0.427	0.441	0.322
I-ID TP G1 n-25	0.119	6.009×10^{-2}	0.663	0.330	0.278
I-ID AP G1 n-25	1.440×10^{-2}*	4.212×10^{-2}*	0.598	0.181	0.260
M-ID AP n-10	0.489	0.372	0.720	0.841	0.866
I-ID AP n-25	7.497×10^{-2}	0.153	0.743	0.532	0.652
M-ID AP n-25	9.970×10^{-2}	0.115	0.412	0.548	0.148

$x^5 - 2x^3 + x$ Symbolic Regression Wilcoxon rank-sun test results continued.

M-ID TP G1 n-50	0.236	0.113	0.505	0.294	0.238
M-ID TP n-25	0.255	0.372	0.531	0.553	0.469
I-ID TP G1 n-10	4.720×10^{-2}*	2.707×10^{-2}*	0.106	0.128	2.637×10^{-2}*
I-ID AP G1 n-10	9.273×10^{-2}	4.249×10^{-2}*	0.341	8.985×10^{-2}	4.163×10^{-2}*
M-ID TP n-50	7.730×10^{-2}	4.944×10^{-2}*	0.648	0.231	0.194
M-ID TP n-10	3.534×10^{-2}*	0.145	0.294	0.249	0.172
M-ID AP G1 n-10	7.074×10^{-3}*	2.695×10^{-3}*	0.116	4.311×10^{-2}*	1.649×10^{-2}*
M-ID AP G1 n-25	1.124×10^{-5}*	4.076×10^{-5}*	9.650×10^{-6}*	1.147×10^{-4}*	6.092×10^{-5}*

	R-ID TP	R-ID AP G1	I-ID TP n-50	M-ID TP G1 n-10	M-ID TP G1 n-25
R-ID AP	4.675×10^{-2}*	0.208	2.105×10^{-2}*	2.007×10^{-3}*	1.291×10^{-3}*
R-ID TP G1	1.310×10^{-2}*	0.395	3.410×10^{-2}*	4.042×10^{-3}*	5.463×10^{-4}*
F-ID AP	0.180	0.662	0.276	1.966×10^{-2}*	2.480×10^{-2}*
F-ID TP	0.612	0.892	0.348	0.113	7.413×10^{-2}
I-ID TP n-25	0.567	0.585	0.296	0.275	4.502×10^{-2}*
F-ID TP G1	0.931	0.889	0.315	0.299	5.875×10^{-2}
I-ID TP n-10	0.743	0.996	0.672	0.438	7.827×10^{-2}
I-ID TP G1 n-50	0.354	0.155	0.996	1.000	0.427
F-ID AP G1	0.724	0.382	0.877	0.728	0.441
I-ID AP n-10	0.657	0.499	0.765	0.597	0.322
R-ID TP		0.667	0.802	0.462	0.233
R-ID AP G1	0.667		0.537	0.263	0.137
I-ID TP n-50	0.802	0.537		0.950	0.527
M-ID TP G1 n-10	0.462	0.263	0.950		0.303
M-ID TP G1 n-25	0.233	0.137	0.527	0.303	
I-ID TP G1 n-25	4.664×10^{-2}*	9.056×10^{-2}	0.479	0.581	0.789
I-ID AP G1 n-25	0.125	0.101	7.806×10^{-2}	0.317	0.509
M-ID AP n-10	0.691	0.415	0.893	0.758	0.313
I-ID AP n-25	0.183	0.118	0.800	0.489	0.980
M-ID AP n-25	0.211	0.149	0.599	0.787	0.858
M-ID TP G1 n-50	0.357	0.111	0.571	0.281	0.779
M-ID TP n-25	0.320	0.306	0.330	0.839	0.717
I-ID TP G1 n-10	6.216×10^{-2}	3.618×10^{-2}*	0.259	0.248	0.313
I-ID AP G1 n-10	1.402×10^{-2}*	8.747×10^{-2}	0.210	0.125	0.257
M-ID TP n-50	5.596×10^{-2}	9.178×10^{-2}	0.385	0.431	0.835
M-ID TP n-10	0.193	0.122	0.255	0.505	0.674
M-ID AP G1 n-10	8.635×10^{-3}*	1.991×10^{-2}*	0.103	4.833×10^{-2}*	0.174
M-ID AP G1 n-25	1.689×10^{-5}*	3.049×10^{-4}*	5.840×10^{-5}*	3.342×10^{-5}*	7.052×10^{-4}*

	I-ID TP G1 n-25	I-ID AP G1 n-25	M-ID AP n-10	I-ID AP n-25	M-ID AP n-25
R-ID AP	5.684×10^{-4}*	2.795×10^{-5}*	4.782×10^{-2}*	2.295×10^{-3}*	1.224×10^{-2}*
R-ID TP G1	3.352×10^{-4}*	1.030×10^{-3}*	6.579×10^{-3}*	2.113×10^{-3}*	6.482×10^{-3}*
F-ID AP	5.178×10^{-3}*	1.328×10^{-3}*	1.782×10^{-2}*	3.083×10^{-2}*	1.874×10^{-2}*
F-ID TP	4.501×10^{-2}*	4.651×10^{-3}*	0.313	9.489×10^{-2}	0.122
I-ID TP n-25	0.129	4.291×10^{-2}*	0.320	0.200	9.762×10^{-2}
F-ID TP G1	0.119	1.440×10^{-2}*	0.489	7.497×10^{-2}	9.970×10^{-2}
I-ID TP n-10	6.009×10^{-2}	4.212×10^{-2}*	0.372	0.153	0.115
I-ID TP G1 n-50	0.663	0.598	0.720	0.743	0.412
F-ID AP G1	0.330	0.181	0.841	0.532	0.548
I-ID AP n-10	0.278	0.260	0.866	0.652	0.148
R-ID TP	4.664×10^{-2}*	0.125	0.691	0.183	0.211
R-ID AP G1	9.056×10^{-2}	0.101	0.415	0.118	0.149
I-ID TP n-50	0.479	7.806×10^{-2}	0.893	0.800	0.599
M-ID TP G1 n-10	0.581	0.317	0.758	0.489	0.787
M-ID TP G1 n-25	0.789	0.509	0.313	0.980	0.858
I-ID TP G1 n-25		0.926	0.605	0.365	0.662

$x^5 - 2x^3 + x$ Symbolic Regression Wilcoxon rank-sun test results continued.

I-ID AP G1 n-25	0.926		0.735	1.000	0.555
M-ID AP n-10	0.605	0.735		0.762	0.882
I-ID AP n-25	0.365	1.000	0.762		0.963
M-ID AP n-25	0.662	0.555	0.882	0.963	
M-ID TP G1 n-50	0.813	0.848	0.255	0.685	0.694
M-ID TP n-25	0.717	0.601	0.885	0.908	0.735
I-ID TP G1 n-10	0.680	0.454	0.317	0.515	0.295
I-ID AP G1 n-10	0.406	0.650	6.643×10^{-2}	0.454	0.532
M-ID TP n-50	0.591	0.858	0.415	0.866	0.521
M-ID TP n-10	0.909	0.671	0.798	0.913	0.554
M-ID AP G1 n-10	0.731	0.198	4.292×10^{-2}*	0.292	9.469×10^{-2}
M-ID AP G1 n-25	3.314×10^{-5}*	9.557×10^{-4}*	2.233×10^{-3}*	2.897×10^{-5}*	1.044×10^{-3}*

	M-ID TP G1 n-50	M-ID TP n-25	I-ID TP G1 n-10	I-ID AP G1 n-10	M-ID TP n-50
R-ID AP	1.912×10^{-3}*	4.821×10^{-3}*	5.022×10^{-5}*	1.001×10^{-4}*	4.924×10^{-4}*
R-ID TP G1	3.853×10^{-3}*	1.374×10^{-3}*	7.153×10^{-4}*	1.028×10^{-3}*	4.456×10^{-4}*
F-ID AP	5.911×10^{-2}	8.646×10^{-3}*	2.453×10^{-3}*	2.637×10^{-3}*	1.082×10^{-2}*
F-ID TP	6.449×10^{-2}	6.382×10^{-2}	5.600×10^{-3}*	1.401×10^{-2}*	4.503×10^{-2}*
I-ID TP n-25	0.106	7.266×10^{-2}	1.134×10^{-2}*	1.898×10^{-2}*	6.807×10^{-2}
F-ID TP G1	0.236	0.255	4.720×10^{-2}*	9.273×10^{-2}	7.730×10^{-2}
I-ID TP n-10	0.113	0.372	2.707×10^{-2}*	4.249×10^{-2}*	4.944×10^{-2}*
I-ID TP G1 n-50	0.505	0.531	0.106	0.341	0.648
F-ID AP G1	0.294	0.553	0.128	8.985×10^{-2}	0.231
I-ID AP n-10	0.238	0.469	2.637×10^{-2}*	4.163×10^{-2}*	0.194
R-ID TP	0.357	0.320	6.216×10^{-2}	1.402×10^{-2}*	5.596×10^{-2}
R-ID AP G1	0.111	0.306	3.618×10^{-2}*	8.747×10^{-2}	9.178×10^{-2}
I-ID TP n-50	0.571	0.330	0.259	0.210	0.385
M-ID TP G1 n-10	0.281	0.839	0.248	0.125	0.431
M-ID TP G1 n-25	0.779	0.717	0.313	0.257	0.835
I-ID TP G1 n-25	0.813	0.717	0.680	0.406	0.591
I-ID AP G1 n-25	0.848	0.601	0.454	0.650	0.858
M-ID AP n-10	0.255	0.885	0.317	6.643×10^{-2}	0.415
I-ID AP n-25	0.685	0.908	0.515	0.454	0.866
M-ID AP n-25	0.694	0.735	0.295	0.532	0.521
M-ID TP G1 n-50		0.823	0.552	0.719	0.858
M-ID TP n-25	0.823		0.225	0.471	0.689
I-ID TP G1 n-10	0.552	0.225		0.796	0.637
I-ID AP G1 n-10	0.719	0.471	0.796		0.992
M-ID TP n-50	0.858	0.689	0.637	0.992	
M-ID TP n-10	0.477	0.901	0.426	0.739	0.866
M-ID AP G1 n-10	0.310	0.197	0.525	0.633	0.835
M-ID AP G1 n-25	4.335×10^{-3}*	7.511×10^{-5}*	4.414×10^{-3}*	6.948×10^{-3}*	4.565×10^{-3}*

	M-ID TP n-10	M-ID AP G1 n-10	M-ID AP G1 n-25
R-ID AP	3.041×10^{-3}*	6.363×10^{-5}*	2.660×10^{-6}*
R-ID TP G1	4.713×10^{-4}*	1.936×10^{-5}*	4.811×10^{-7}*
F-ID AP	1.720×10^{-2}*	3.000×10^{-4}*	1.980×10^{-5}*
F-ID TP	3.626×10^{-2}*	1.482×10^{-3}*	2.853×10^{-5}*
I-ID TP n-25	5.595×10^{-2}	2.639×10^{-2}*	1.544×10^{-6}*
F-ID TP G1	3.534×10^{-2}*	7.074×10^{-3}*	1.124×10^{-5}*
I-ID TP n-10	0.145	2.695×10^{-3}*	4.076×10^{-5}*
I-ID TP G1 n-50	0.294	0.116	9.650×10^{-6}*
F-ID AP G1	0.249	4.311×10^{-2}*	1.147×10^{-4}*
I-ID AP n-10	0.172	1.649×10^{-2}*	6.092×10^{-5}*
R-ID TP	0.193	8.635×10^{-3}*	1.689×10^{-5}*
R-ID AP G1	0.122	1.991×10^{-2}*	3.049×10^{-4}*

$x^5 - 2x^3 + x$ Symbolic Regression Wilcoxon rank-sun test results continued.

I-ID TP n-50	0.255	0.103	5.840×10^{-5}*
M-ID TP G1 n-10	0.505	4.833×10^{-2}*	3.342×10^{-5}*
M-ID TP G1 n-25	0.674	0.174	7.052×10^{-4}*
I-ID TP G1 n-25	0.909	0.731	3.314×10^{-5}*
I-ID AP G1 n-25	0.671	0.198	9.557×10^{-4}*
M-ID AP n-10	0.798	4.292×10^{-2}*	2.233×10^{-3}*
I-ID AP n-25	0.913	0.292	2.897×10^{-5}*
M-ID AP n-25	0.554	9.469×10^{-2}	1.044×10^{-3}*
M-ID TP G1 n-50	0.477	0.310	4.335×10^{-3}*
M-ID TP n-25	0.901	0.197	7.511×10^{-5}*
I-ID TP G1 n-10	0.426	0.525	4.414×10^{-3}*
I-ID AP G1 n-10	0.739	0.633	6.948×10^{-3}*
M-ID TP n-50	0.866	0.835	4.565×10^{-3}*
M-ID TP n-10		0.420	1.113×10^{-2}*
M-ID AP G1 n-10	0.420		2.353×10^{-2}*
M-ID AP G1 n-25	1.113×10^{-2}*	2.353×10^{-2}*	

Table A.23: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the 8×8 Lawn Mower problem after 100000 fitness evaluations.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
M-ID AP n-10	0.224	1.58389	0.22400	19
M-ID TP G1 n-50	0.230	1.62632	0.23000	21
M-ID TP G1 n-25	0.238	1.68290	0.23800	24
M-ID AP G1 n-10	0.260	1.83846	0.26000	18
R-ID AP	0.462	2.287	0.323	20
M-ID TP n-10	0.464	2.299	0.325	21
M-ID AP n-25	0.492	2.435	0.344	19
M-ID TP G1 n-10	0.492	2.435	0.344	22
M-ID TP n-50	0.506	2.510	0.355	17
M-ID TP n-25	1.188	3.608	0.510	21
M-ID AP G1 n-25	1.212	3.674	0.520	14
R-ID TP G1	1.318	3.999	0.566	21
R-ID AP G1	1.376	4.219	0.597	17
R-ID TP	1.736	4.354	0.616	24
F-ID TP	5.966	6.894	0.975	15
F-ID AP	9.634	6.510	0.921	3
F-ID TP G1	10.672	6.339	0.896	2
F-ID AP G1	14.886	4.421	0.625	1
I-ID AP n-10	28.288	1.323	0.187	0
I-ID AP G1 n-10	28.422	1.485	0.210	0
I-ID AP n-25	28.532	1.426	0.202	0
I-ID TP n-10	28.556	1.332	0.188	0
I-ID AP G1 n-25	28.700	1.232	0.174	0
I-ID TP n-50	28.718	1.229	0.174	0
I-ID TP G1 n-25	28.860	1.114	0.158	0
I-ID TP n-25	28.914	1.403	0.198	0
I-ID TP G1 n-50	28.928	1.127	0.159	0
I-ID TP G1 n-10	28.940	1.167	0.165	0

Table A.24: This table reports the p-value of Wilcoxon rank-sum tests performed on th best fitness values of each approach after 100000 fitness evaluations on the Even 7 Parity The p-values reported are calculated with a confidence interval of 0.05. Values marked asterisk (*) are significant.

	M-ID AP n-10	M-ID TP G1 n-50	M-ID TP G1 n-25	M-ID AP G1 n-10	R-ID AP
M-ID AP n-10		0.817	0.188	0.634	0.596
M-ID TP G1 n-50	0.817		0.584	0.984	0.289
M-ID TP G1 n-25	0.188	0.584		0.289	0.198
M-ID AP G1 n-10	0.634	0.984	0.289		0.248
R-ID AP	0.596	0.289	0.198	0.248	
M-ID TP n-10	0.380	0.185	0.159	0.393	0.705
M-ID AP n-25	0.732	0.633	0.369	0.918	0.431
M-ID TP G1 n-10	0.528	0.468	0.143	0.261	0.861
M-ID TP n-50	0.839	0.826	0.328	0.879	0.289
M-ID TP n-25	0.370	0.237	2.356×10^{-2}*	0.221	0.964
M-ID AP G1 n-25	0.201	0.302	3.002×10^{-2}*	0.174	0.581
R-ID TP G1	0.608	0.416	0.184	0.665	0.665
R-ID AP G1	0.165	8.360×10^{-2}	2.036×10^{-2}*	0.114	0.659
R-ID TP	0.389	7.589×10^{-2}	5.750×10^{-2}	0.346	0.658
F-ID TP	3.599×10^{-5}*	1.094×10^{-5}*	1.285×10^{-5}*	5.107×10^{-6}*	3.028×10^{-4}*
F-ID AP	1.140×10^{-8}*	3.280×10^{-9}*	4.910×10^{-9}*	1.252×10^{-8}*	9.083×10^{-8}*
F-ID TP G1	9.520×10^{-9}*	6.210×10^{-9}*	4.700×10^{-9}*	1.741×10^{-8}*	1.952×10^{-8}*
F-ID AP G1	1.300×10^{-9}*	8.500×10^{-10}*	1.220×10^{-9}*	1.300×10^{-9}*	1.220×10^{-9}*
I-ID AP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID TP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-25	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID TP n-50	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID TP G1 n-25	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP n-25	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP G1 n-50	7.700×10^{-10}*	7.600×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP G1 n-10	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*

	M-ID TP n-10	M-ID AP n-25	M-ID TP G1 n-10	M-ID TP n-50	M-ID TP n-25
M-ID AP n-10	0.380	0.732	0.528	0.839	0.370
M-ID TP G1 n-50	0.185	0.633	0.468	0.826	0.237
M-ID TP G1 n-25	0.159	0.369	0.143	0.328	2.356×10^{-2}*
M-ID AP G1 n-10	0.393	0.918	0.261	0.879	0.221
R-ID AP	0.705	0.431	0.861	0.289	0.964
M-ID TP n-10		0.391	0.995	0.403	0.414
M-ID AP n-25	0.391		0.785	0.694	0.406
M-ID TP G1 n-10	0.995	0.785		0.520	0.974
M-ID TP n-50	0.403	0.694	0.520		0.305
M-ID TP n-25	0.414	0.406	0.974	0.305	
M-ID AP G1 n-25	0.647	0.514	0.942	0.305	0.874
R-ID TP G1	0.861	0.855	0.880	0.768	0.890
R-ID AP G1	0.303	6.054×10^{-2}	0.305	5.716×10^{-2}	0.590
R-ID TP	0.402	0.426	0.516	0.398	0.757
F-ID TP	6.065×10^{-5}*	2.712×10^{-4}*	4.414×10^{-5}*	1.293×10^{-5}*	9.708×10^{-6}*
F-ID AP	1.205×10^{-8}*	7.244×10^{-8}*	5.468×10^{-8}*	2.421×10^{-8}*	1.466×10^{-8}*
F-ID TP G1	5.550×10^{-9}*	3.212×10^{-8}*	8.400×10^{-9}*	4.480×10^{-9}*	9.370×10^{-9}*
F-ID AP G1	9.300×10^{-10}*	1.220×10^{-9}*	8.500×10^{-10}*	1.140×10^{-9}*	1.470×10^{-9}*
I-ID AP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID TP n-10	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*

8×8 Lawn Mower Wilcoxon rank-sun test results continued.

I-ID AP G1 n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP n-50	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID TP G1 n-25	7.600×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP n-25	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP G1 n-50	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.600×10^{-10}*
I-ID TP G1 n-10	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*

	M-ID AP G1 n-25	R-ID TP G1	R-ID AP G1	R-ID TP	F-ID TP
M-ID AP n-10	0.201	0.608	0.165	0.389	3.599×10^{-5}*
M-ID TP G1 n-50	0.302	0.416	8.360×10^{-2}	7.589×10^{-2}	1.094×10^{-5}*
M-ID TP G1 n-25	3.002×10^{-2}*	0.184	2.036×10^{-2}*	5.750×10^{-2}	1.285×10^{-5}*
M-ID AP G1 n-10	0.174	0.665	0.114	0.346	5.107×10^{-6}*
R-ID AP	0.581	0.665	0.659	0.658	3.028×10^{-4}*
M-ID TP n-10	0.647	0.861	0.303	0.402	6.065×10^{-5}*
M-ID AP n-25	0.514	0.855	6.054×10^{-2}	0.426	2.712×10^{-4}*
M-ID TP G1 n-10	0.942	0.880	0.305	0.516	4.414×10^{-5}*
M-ID TP n-50	0.305	0.768	5.716×10^{-2}	0.398	1.293×10^{-5}*
M-ID TP n-25	0.874	0.890	0.590	0.757	9.708×10^{-6}*
M-ID AP G1 n-25		0.958	0.495	0.933	2.167×10^{-4}*
R-ID TP G1	0.958		9.717×10^{-2}	0.668	4.797×10^{-4}*
R-ID AP G1	0.495	9.717×10^{-2}		0.737	3.328×10^{-3}*
R-ID TP	0.933	0.668	0.737		3.237×10^{-4}*
F-ID TP	2.167×10^{-4}*	4.797×10^{-4}*	3.328×10^{-3}*	3.237×10^{-4}*	
F-ID AP	2.846×10^{-8}*	5.983×10^{-8}*	3.291×10^{-6}*	1.816×10^{-7}*	2.221×10^{-3}*
F-ID TP G1	2.164×10^{-8}*	9.350×10^{-8}*	3.336×10^{-7}*	3.620×10^{-8}*	2.221×10^{-3}*
F-ID AP G1	1.420×10^{-9}*	1.560×10^{-9}*	1.190×10^{-9}*	1.710×10^{-9}*	2.844×10^{-7}*
I-ID AP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID TP n-10	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-25	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP n-50	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID TP G1 n-25	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID TP n-25	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID TP G1 n-50	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP G1 n-10	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*

	F-ID AP	F-ID TP G1	F-ID AP G1	I-ID AP n-10	I-ID AP G1 n-10
M-ID AP n-10	1.140×10^{-8}*	9.520×10^{-9}*	1.300×10^{-9}*	7.800×10^{-10}*	7.800×10^{-10}*
M-ID TP G1 n-50	3.280×10^{-9}*	6.210×10^{-9}*	8.500×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
M-ID TP G1 n-25	4.910×10^{-9}*	4.700×10^{-9}*	1.220×10^{-9}*	7.700×10^{-10}*	7.800×10^{-10}*
M-ID AP G1 n-10	1.252×10^{-8}*	1.741×10^{-8}*	1.300×10^{-9}*	7.800×10^{-10}*	7.800×10^{-10}*
R-ID AP	9.083×10^{-8}*	1.952×10^{-8}*	1.220×10^{-9}*	7.800×10^{-10}*	7.800×10^{-10}*
M-ID TP n-10	1.205×10^{-8}*	5.550×10^{-9}*	9.300×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
M-ID AP n-25	7.244×10^{-8}*	3.212×10^{-8}*	1.220×10^{-9}*	7.800×10^{-10}*	7.800×10^{-10}*
M-ID TP G1 n-10	5.468×10^{-8}*	8.400×10^{-9}*	8.500×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
M-ID TP n-50	2.421×10^{-8}*	4.480×10^{-9}*	1.140×10^{-9}*	7.800×10^{-10}*	7.800×10^{-10}*
M-ID TP n-25	1.466×10^{-8}*	9.370×10^{-9}*	1.470×10^{-9}*	7.800×10^{-10}*	7.800×10^{-10}*
M-ID AP G1 n-25	2.846×10^{-8}*	2.164×10^{-8}*	1.420×10^{-9}*	7.800×10^{-10}*	7.800×10^{-10}*
R-ID TP G1	5.983×10^{-8}*	9.350×10^{-8}*	1.560×10^{-9}*	7.800×10^{-10}*	7.800×10^{-10}*
R-ID AP G1	3.291×10^{-6}*	3.336×10^{-7}*	1.190×10^{-9}*	7.800×10^{-10}*	7.800×10^{-10}*
R-ID TP	1.816×10^{-7}*	3.620×10^{-8}*	1.710×10^{-9}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID TP	2.221×10^{-3}*	2.221×10^{-3}*	2.844×10^{-7}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID AP		0.235	1.477×10^{-4}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID TP G1	0.235		1.802×10^{-4}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID AP G1	1.477×10^{-4}*	1.802×10^{-4}*		7.800×10^{-10}*	7.800×10^{-10}*

8×8 Lawn Mower Wilcoxon rank-sun test results continued.

I-ID AP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*		0.778
I-ID AP G1 n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	0.778	
I-ID AP n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	0.487	0.849
I-ID TP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	0.434	0.659
I-ID AP G1 n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	0.191	0.543
I-ID TP n-50	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.045×10^{-2}	0.240
I-ID TP G1 n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	5.165×10^{-2}	0.177
I-ID TP n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	3.452×10^{-2}*	0.172
I-ID TP G1 n-50	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	2.172×10^{-2}*	9.959×10^{-2}
I-ID TP G1 n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	1.613×10^{-2}*	0.110

	I-ID AP n-25	I-ID TP n-10	I-ID AP G1 n-25	I-ID TP n-50	I-ID TP G1 n-25
M-ID AP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID TP G1 n-50	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID TP G1 n-25	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
M-ID AP G1 n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
R-ID AP	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
M-ID TP n-10	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
M-ID AP n-25	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID TP G1 n-10	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
M-ID TP n-50	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID TP n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
M-ID AP G1 n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
R-ID TP G1	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
R-ID AP G1	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
R-ID TP	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
F-ID TP	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID AP	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID TP G1	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID AP G1	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-10	0.487	0.434	0.191	7.045×10^{-2}	5.165×10^{-2}
I-ID AP G1 n-10	0.849	0.659	0.543	0.240	0.177
I-ID AP n-25		0.891	0.470	0.518	0.332
I-ID TP n-10	0.891		0.750	0.616	0.320
I-ID AP G1 n-25	0.470	0.750		0.887	0.451
I-ID TP n-50	0.518	0.616	0.887		1.000
I-ID TP G1 n-25	0.332	0.320	0.451	1.000	
I-ID TP n-25	0.312	0.224	0.263	0.433	0.611
I-ID TP G1 n-50	0.312	0.173	0.664	0.567	0.689
I-ID TP G1 n-10	0.109	0.279	0.363	0.521	0.533

	I-ID TP n-25	I-ID TP G1 n-50	I-ID TP G1 n-10
M-ID AP n-10	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID TP G1 n-50	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*
M-ID TP G1 n-25	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*
M-ID AP G1 n-10	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
R-ID AP	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID TP n-10	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID AP n-25	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID TP G1 n-10	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
M-ID TP n-50	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*
M-ID TP n-25	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*
M-ID AP G1 n-25	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
R-ID TP G1	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
R-ID AP G1	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
R-ID TP	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*

8×8 Lawn Mower Wilcoxon rank-sun test results continued.

F-ID TP	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
F-ID AP	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID TP G1	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID AP G1	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-10	3.452×10^{-2}*	2.172×10^{-2}*	1.613×10^{-2}*
I-ID AP G1 n-10	0.172	9.959×10^{-2}	0.110
I-ID AP n-25	0.312	0.312	0.109
I-ID TP n-10	0.224	0.173	0.279
I-ID AP G1 n-25	0.263	0.664	0.363
I-ID TP n-50	0.433	0.567	0.521
I-ID TP G1 n-25	0.611	0.689	0.533
I-ID TP n-25		0.627	0.861
I-ID TP G1 n-50	0.627		0.890
I-ID TP G1 n-10	0.861	0.890	

A.5 Comparison to Standard GE

Table A.25: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the Santa Fe Ant Trail problem after 100000 fitness evaluations of each approach to identifying modules, GE, and GE with ADFs.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
F-ID TP G1	9.520	12.011	1.699	18
F-ID AP G1	11.480	13.278	1.878	21
R-ID TP G1	15.420	13.527	1.913	13
I-ID TP n-25	15.720	13.240	1.872	9
I-ID AP G1 n-10	15.820	14.951	2.114	10
GE	16.180	13.970	1.976	10
I-ID AP G1 n-25	16.360	14.525	2.054	10
I-ID TP n-50	16.380	13.041	1.844	9
F-ID AP	16.760	13.283	1.878	12
ADF	17.160	13.219	1.870	9
R-ID AP	17.240	12.934	1.829	12
M-ID TP G1 n-50	17.340	14.662	2.073	12
M-ID TP G1 n-25	17.580	13.256	1.875	7
M-ID TP G1 n-10	17.840	12.921	1.827	11
M-ID AP G1 n-10	17.900	13.347	1.888	8
I-ID AP n-10	18.100	13.287	1.879	9
M-ID AP n-10	18.300	13.452	1.902	12
I-ID AP n-25	18.300	13.474	1.906	11
R-ID AP G1	18.600	12.779	1.807	8
I-ID TP G1 n-25	18.980	13.202	1.867	5
I-ID TP G1 n-10	19.140	13.398	1.895	6
M-ID TP n-25	19.220	13.196	1.866	8
M-ID TP n-10	19.260	14.294	2.021	9
I-ID TP G1 n-50	19.680	12.306	1.740	5
F-ID TP	19.700	12.960	1.833	8
I-ID TP n-10	19.800	12.51285	1.770	6
M-ID AP n-25	19.940	13.70745	1.939	6
M-ID TP n-50	20.060	13.85554	1.959	7
R-ID TP	21.300	12.91771	1.827	7
M-ID AP G1 n-25	21.320	14.764	2.088	7

Table A.26: This table reports the p-value of Wilcoxon rank-sum tests performed on the best fitness values of each approach to identifying modules, GE, and GE with ADFs after fitness evaluations on the Santa Fe Ant Trail problem. The p-values reported are calcula a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	F-ID TP G1	F-ID AP G1	R-ID TP G1	I-ID TP n-25	I-ID AP G1 n-10
F-ID TP G1		0.472	4.052×10^{-2}*	1.659×10^{-2}*	1.864×10^{-2}*
F-ID AP G1	0.472		7.572×10^{-2}	0.137	0.181
R-ID TP G1	4.052×10^{-2}*	7.572×10^{-2}		0.845	0.799
I-ID TP n-25	1.659×10^{-2}*	0.137	0.845		0.991
I-ID AP G1 n-10	1.864×10^{-2}*	0.181	0.799	0.991	
GE	1.620×10^{-2}*	6.109×10^{-2}	0.773	0.820	0.898
I-ID AP G1 n-25	3.307×10^{-2}*	2.807×10^{-2}*	0.784	0.891	0.886
I-ID TP n-50	9.065×10^{-3}*	7.263×10^{-2}	0.763	0.779	0.723
F-ID AP	7.638×10^{-3}*	8.108×10^{-2}	0.593	0.789	0.633
ADF	8.015×10^{-3}*	5.078×10^{-2}	0.545	0.659	0.693
R-ID AP	5.060×10^{-3}*	2.351×10^{-2}*	0.439	0.698	0.582
M-ID TP G1 n-50	6.410×10^{-3}*	5.081×10^{-2}	0.564	0.604	0.579
M-ID TP G1 n-25	2.339×10^{-3}*	7.660×10^{-3}*	0.409	0.512	0.654
M-ID TP G1 n-10	1.396×10^{-3}*	2.465×10^{-2}*	0.337	0.465	0.455
M-ID AP G1 n-10	1.803×10^{-3}*	2.390×10^{-2}*	0.286	0.421	0.531
I-ID AP n-10	1.151×10^{-3}*	1.667×10^{-2}*	0.307	0.290	0.370
M-ID AP n-10	4.518×10^{-3}*	1.470×10^{-2}*	0.202	0.149	0.429
I-ID AP n-25	1.220×10^{-3}*	1.021×10^{-2}*	0.289	0.231	0.518
R-ID AP G1	1.761×10^{-3}*	5.455×10^{-3}*	0.183	0.264	0.259
I-ID TP G1 n-25	3.229×10^{-4}*	7.406×10^{-3}*	0.194	0.137	0.179
I-ID TP G1 n-10	1.766×10^{-4}*	6.503×10^{-3}*	7.025×10^{-2}	0.186	0.320
M-ID TP n-25	7.051×10^{-4}*	6.402×10^{-3}*	0.179	0.141	0.239
M-ID TP n-10	1.070×10^{-3}*	1.626×10^{-2}*	0.184	0.230	0.131
I-ID TP G1 n-50	1.227×10^{-5}*	5.263×10^{-3}*	0.122	0.182	0.162
F-ID TP	3.034×10^{-4}*	1.504×10^{-2}*	6.469×10^{-2}	0.161	0.218
I-ID TP n-10	6.909×10^{-4}*	3.518×10^{-3}*	0.110	0.119	0.299
M-ID AP n-25	1.354×10^{-5}*	2.687×10^{-3}*	8.256×10^{-2}	0.119	0.184
M-ID TP n-50	2.511×10^{-4}*	3.337×10^{-3}*	5.812×10^{-2}	0.107	0.168
R-ID TP	5.633×10^{-5}*	1.044×10^{-3}*	3.042×10^{-2}*	4.431×10^{-2}*	4.380×10^{-2}*
M-ID AP G1 n-25	6.475×10^{-5}*	3.347×10^{-3}*	3.540×10^{-2}*	3.490×10^{-2}*	7.625×10^{-2}

	GE	I-ID AP G1 n-25	I-ID TP n-50	F-ID AP	ADF
F-ID TP G1	1.620×10^{-2}*	3.307×10^{-2}*	9.065×10^{-3}*	7.638×10^{-3}*	8.015×10^{-3}*
F-ID AP G1	6.109×10^{-2}	2.807×10^{-2}*	7.263×10^{-2}	8.108×10^{-2}	5.078×10^{-2}
R-ID TP G1	0.773	0.784	0.763	0.593	0.545
I-ID TP n-25	0.820	0.891	0.779	0.789	0.659
I-ID AP G1 n-10	0.898	0.886	0.723	0.633	0.693
GE		0.951	0.870	0.879	0.882
I-ID AP G1 n-25	0.951		0.952	0.804	0.678
I-ID TP n-50	0.870	0.952		0.896	0.557
F-ID AP	0.879	0.804	0.896		0.899
ADF	0.882	0.678	0.557	0.899	
R-ID AP	0.674	0.611	0.883	0.748	1.000
M-ID TP G1 n-50	0.519	0.731	0.693	1.000	0.982
M-ID TP G1 n-25	0.641	0.667	0.569	0.893	0.992
M-ID TP G1 n-10	0.588	0.481	0.638	0.658	0.695
M-ID AP G1 n-10	0.623	0.608	0.562	0.644	0.735
I-ID AP n-10	0.559	0.550	0.395	0.558	0.785
M-ID AP n-10	0.656	0.498	0.429	0.562	0.591
I-ID AP n-25	0.492	0.388	0.382	0.582	0.595

Santa Fe Ant Trail Wilcoxon rank-sun test results continued.

R-ID AP G1	0.320	0.418	0.377	0.436	0.533
I-ID TP G1 n-25	0.320	0.440	0.290	0.403	0.505
I-ID TP G1 n-10	0.292	0.253	0.155	0.472	0.462
M-ID TP n-25	0.297	0.207	0.302	0.367	0.580
M-ID TP n-10	0.363	0.317	0.148	0.266	0.275
I-ID TP G1 n-50	0.174	0.313	0.130	0.297	0.354
F-ID TP	0.308	0.253	0.135	0.309	0.325
I-ID TP n-10	0.175	0.273	0.240	0.211	0.272
M-ID AP n-25	0.200	0.156	0.133	0.186	0.373
M-ID TP n-50	0.193	7.797×10^{-2}	0.165	0.210	0.212
R-ID TP	0.104	8.795×10^{-2}	$3.823 \times 10^{-2}*$	8.794×10^{-2}	0.160
M-ID AP G1 n-25	6.172×10^{-2}	7.011×10^{-2}	6.002×10^{-2}	0.105	0.133

	R-ID AP	M-ID TP G1 n-50	M-ID TP G1 n-25	M-ID TP G1 n-10	M-ID AP G1 n-10
F-ID TP G1	$5.060 \times 10^{-3}*$	$6.410 \times 10^{-3}*$	$2.339 \times 10^{-3}*$	$1.396 \times 10^{-3}*$	$1.803 \times 10^{-3}*$
F-ID AP G1	$2.351 \times 10^{-2}*$	5.081×10^{-2}	$7.660 \times 10^{-3}*$	$2.465 \times 10^{-2}*$	$2.390 \times 10^{-2}*$
R-ID TP G1	0.439	0.564	0.409	0.337	0.286
I-ID TP n-25	0.698	0.604	0.512	0.465	0.421
I-ID AP G1 n-10	0.582	0.579	0.654	0.455	0.531
GE	0.674	0.519	0.641	0.588	0.623
I-ID AP G1 n-25	0.611	0.731	0.667	0.481	0.608
I-ID TP n-50	0.883	0.693	0.569	0.638	0.562
F-ID AP	0.748	1.000	0.893	0.658	0.644
ADF	1.000	0.982	0.992	0.695	0.735
R-ID AP		0.996	0.901	0.806	0.662
M-ID TP G1 n-50	0.996		0.958	0.881	0.834
M-ID TP G1 n-25	0.901	0.958		0.906	0.770
M-ID TP G1 n-10	0.806	0.881	0.906		0.939
M-ID AP G1 n-10	0.662	0.834	0.770	0.939	
I-ID AP n-10	0.748	0.698	0.750	0.856	0.902
M-ID AP n-10	0.628	0.522	0.715	0.875	0.992
I-ID AP n-25	0.684	0.639	0.996	1.000	1.000
R-ID AP G1	0.899	0.435	0.548	0.797	0.839
I-ID TP G1 n-25	0.496	0.608	0.564	0.777	0.704
I-ID TP G1 n-10	0.471	0.662	0.522	0.312	0.648
M-ID TP n-25	0.343	0.508	0.458	0.530	0.607
M-ID TP n-10	0.318	0.382	0.597	0.529	0.708
I-ID TP G1 n-50	0.322	0.462	0.401	0.488	0.633
F-ID TP	0.359	0.360	0.440	0.460	0.473
I-ID TP n-10	0.382	0.395	0.294	0.409	0.484
M-ID AP n-25	0.274	0.340	0.114	0.299	0.372
M-ID TP n-50	0.274	0.424	0.355	0.351	0.505
R-ID TP	$4.540 \times 10^{-2}*$	0.229	6.316×10^{-2}	$4.159 \times 10^{-2}*$	0.191
M-ID AP G1 n-25	0.194	0.168	0.218	0.145	0.242

	I-ID AP n-10	M-ID AP n-10	I-ID AP n-25	R-ID AP G1	I-ID TP G1 n-25
F-ID TP G1	$1.151 \times 10^{-3}*$	$4.518 \times 10^{-3}*$	$1.220 \times 10^{-3}*$	$1.761 \times 10^{-3}*$	$3.229 \times 10^{-4}*$
F-ID AP G1	$1.667 \times 10^{-2}*$	$1.470 \times 10^{-2}*$	$1.021 \times 10^{-2}*$	$5.455 \times 10^{-3}*$	$7.406 \times 10^{-3}*$
R-ID TP G1	0.307	0.202	0.289	0.183	0.194
I-ID TP n-25	0.290	0.149	0.231	0.264	0.137
I-ID AP G1 n-10	0.370	0.429	0.518	0.259	0.179
GE	0.559	0.656	0.492	0.320	0.320
I-ID AP G1 n-25	0.550	0.498	0.388	0.418	0.440
I-ID TP n-50	0.395	0.429	0.382	0.377	0.290
F-ID AP	0.558	0.562	0.582	0.436	0.403
ADF	0.785	0.591	0.595	0.533	0.505

Santa Fe Ant Trail Wilcoxon rank-sun test results continued.

R-ID AP	0.748	0.628	0.684	0.899	0.496
M-ID TP G1 n-50	0.698	0.522	0.639	0.435	0.608
M-ID TP G1 n-25	0.750	0.715	0.996	0.548	0.564
M-ID TP G1 n-10	0.856	0.875	1.000	0.797	0.777
M-ID AP G1 n-10	0.902	0.992	1.000	0.839	0.704
I-ID AP n-10		0.992	1.000	0.973	0.750
M-ID AP n-10	0.992		0.969	0.861	0.791
I-ID AP n-25	1.000	0.969		0.794	0.932
R-ID AP G1	0.973	0.861	0.794		0.955
I-ID TP G1 n-25	0.750	0.791	0.932	0.955	
I-ID TP G1 n-10	0.727	0.804	0.538	0.913	0.987
M-ID TP n-25	0.735	0.747	0.695	0.858	0.960
M-ID TP n-10	0.611	0.696	0.701	0.799	0.918
I-ID TP G1 n-50	0.508	0.641	0.487	0.672	0.548
F-ID TP	0.488	0.704	0.559	0.750	0.691
I-ID TP n-10	0.515	0.404	0.479	0.805	0.696
M-ID AP n-25	0.491	0.406	0.420	0.636	0.605
M-ID TP n-50	0.495	0.612	0.192	0.449	0.604
R-ID TP	0.148	0.209	0.269	0.178	0.322
M-ID AP G1 n-25	0.352	0.336	0.234	0.269	0.415

	I-ID TP G1 n-10	M-ID TP n-25	M-ID TP n-10	I-ID TP G1 n-50	F-ID TP
F-ID TP G1	1.766×10^{-4}*	7.051×10^{-4}*	1.070×10^{-3}*	1.227×10^{-5}*	3.034×10^{-4}*
F-ID AP G1	6.503×10^{-3}*	6.402×10^{-3}*	1.626×10^{-2}*	5.263×10^{-3}*	1.504×10^{-2}*
R-ID TP G1	7.025×10^{-2}	0.179	0.184	0.122	6.469×10^{-2}
I-ID TP n-25	0.186	0.141	0.230	0.182	0.161
I-ID AP G1 n-10	0.320	0.239	0.131	0.162	0.218
GE	0.292	0.297	0.363	0.174	0.308
I-ID AP G1 n-25	0.253	0.207	0.317	0.313	0.253
I-ID TP n-50	0.155	0.302	0.148	0.130	0.135
F-ID AP	0.472	0.367	0.266	0.297	0.309
ADF	0.462	0.580	0.275	0.354	0.325
R-ID AP	0.471	0.343	0.318	0.322	0.359
M-ID TP G1 n-50	0.662	0.508	0.382	0.462	0.360
M-ID TP G1 n-25	0.522	0.458	0.597	0.401	0.440
M-ID TP G1 n-10	0.312	0.530	0.529	0.488	0.460
M-ID AP G1 n-10	0.648	0.607	0.708	0.633	0.473
I-ID AP n-10	0.727	0.735	0.611	0.508	0.488
M-ID AP n-10	0.804	0.747	0.696	0.641	0.704
I-ID AP n-25	0.538	0.695	0.701	0.487	0.559
R-ID AP G1	0.913	0.858	0.799	0.672	0.750
I-ID TP G1 n-25	0.987	0.960	0.918	0.548	0.691
I-ID TP G1 n-10		0.739	0.862	0.812	0.750
M-ID TP n-25	0.739		0.928	0.913	0.789
M-ID TP n-10	0.862	0.928		0.894	0.937
I-ID TP G1 n-50	0.812	0.913	0.894		0.585
F-ID TP	0.750	0.789	0.937	0.585	
I-ID TP n-10	0.772	0.814	0.879	0.898	0.979
M-ID AP n-25	0.739	0.641	0.731	0.800	0.886
M-ID TP n-50	0.773	0.526	0.825	0.978	0.922
R-ID TP	0.478	0.335	0.461	0.459	0.535
M-ID AP G1 n-25	0.385	0.381	0.448	0.546	0.481

	I-ID TP n-10	M-ID AP n-25	M-ID TP n-50	R-ID TP	M-ID AP G1 n-25
F-ID TP G1	6.909×10^{-4}*	1.354×10^{-5}*	2.511×10^{-4}*	5.633×10^{-5}*	6.475×10^{-5}*
F-ID AP G1	3.518×10^{-3}*	2.687×10^{-3}*	3.337×10^{-3}*	1.044×10^{-3}*	3.347×10^{-3}*

Santa Fe Ant Trail Wilcoxon rank-sun test results continued.

R-ID TP G1	0.110	8.256×10^{-2}	5.812×10^{-2}	3.042×10^{-2}*	3.540×10^{-2}*
I-ID TP n-25	0.119	0.119	0.107	4.431×10^{-2}*	3.490×10^{-2}*
I-ID AP G1 n-10	0.299	0.184	0.168	4.380×10^{-2}*	7.625×10^{-2}
GE	0.175	0.200	0.193	0.104	6.172×10^{-2}
I-ID AP G1 n-25	0.273	0.156	7.797×10^{-2}	8.795×10^{-2}	7.011×10^{-2}
I-ID TP n-50	0.240	0.133	0.165	3.823×10^{-2}*	6.002×10^{-2}
F-ID AP	0.211	0.186	0.210	8.794×10^{-2}	0.105
ADF	0.272	0.373	0.212	0.160	0.133
R-ID AP	0.382	0.274	0.274	4.540×10^{-2}*	0.194
M-ID TP G1 n-50	0.395	0.340	0.424	0.229	0.168
M-ID TP G1 n-25	0.294	0.114	0.355	6.316×10^{-2}	0.218
M-ID TP G1 n-10	0.409	0.299	0.351	4.159×10^{-2}*	0.145
M-ID AP G1 n-10	0.484	0.372	0.505	0.191	0.242
I-ID AP n-10	0.515	0.491	0.495	0.148	0.352
M-ID AP n-10	0.404	0.406	0.612	0.209	0.336
I-ID AP n-25	0.479	0.420	0.192	0.269	0.234
R-ID AP G1	0.805	0.636	0.449	0.178	0.269
I-ID TP G1 n-25	0.696	0.605	0.604	0.322	0.415
I-ID TP G1 n-10	0.772	0.739	0.773	0.478	0.385
M-ID TP n-25	0.814	0.641	0.526	0.335	0.381
M-ID TP n-10	0.879	0.731	0.825	0.461	0.448
I-ID TP G1 n-50	0.898	0.800	0.978	0.459	0.546
F-ID TP	0.979	0.886	0.922	0.535	0.481
I-ID TP n-10		0.947	0.932	0.443	0.629
M-ID AP n-25	0.947		0.982	0.567	0.380
M-ID TP n-50	0.932	0.982		0.634	0.573
R-ID TP	0.443	0.567	0.634		0.781
M-ID AP G1 n-25	0.629	0.380	0.573	0.781	

Table A.27: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the Even 7 Parity problem after 100000 fitness evaluations of each approach to identifying modules, GE, and GE with ADFs.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
F-ID AP G1	0.480	2.082	0.294	47
I-ID TP G1 n-50	0.640	2.724	0.385	47
F-ID TP G1	0.720	2.990	0.423	47
R-ID AP G1	0.880	2.715	0.384	45
M-ID TP G1 n-10	1.040	3.664	0.518	45
M-ID TP G1 n-25	1.460	3.871	0.548	42
I-ID TP G1 n-25	1.600	6.465	0.914	46
M-ID TP G1 n-50	1.760	5.723	0.809	44
I-ID AP G1 n-10	1.760	5.947	0.841	44
R-ID AP	1.820	4.839	0.684	42
ADF	1.980	6.702	0.948	43
R-ID TP G1	2.000	6.168	0.872	44
M-ID AP G1 n-10	2.140	5.292	0.748	41
I-ID TP n-10	2.160	5.991	0.847	42
F-ID TP	2.320	6.619	0.936	42
M-ID AP G1 n-25	2.420	6.101	0.863	42
F-ID AP	2.640	6.382	0.903	42
R-ID TP	2.680	6.757	0.956	40
I-ID AP G1 n-25	2.760	6.675	0.944	41
I-ID TP G1 n-10	2.780	6.287	0.889	39
M-ID TP n-10	3.180	6.766	0.957	38
M-ID TP n-25	3.320	6.885	0.974	37
GE	3.600	8.381	1.185	40
I-ID TP n-50	3.620	6.978	0.987	36
I-ID TP n-25	3.660	7.580	1.072	37
M-ID TP n-50	3.660	7.927	1.121	36
I-ID AP n-10	3.980	8.312	1.175	37
M-ID AP n-10	4.580	8.157	1.154	34
M-ID AP n-25	4.680	8.353	1.181	34
I-ID AP n-25	5.480	8.683	1.228	33

Table A.28: This table reports the p-value of Wilcoxon rank-sum tests performed on the av
best fitness values of each approach to identifying modules, GE, and GE with ADFs after 1
fitness evaluations on the Even 7 Parity problem. The p-values reported are calculated w
confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	F-ID AP G1	I-ID TP G1 n-50	F-ID TP G1	R-ID AP G1	M-ID TP G1 n-10
F-ID AP G1		0.832	0.673	0.395	0.352
I-ID TP G1 n-50	0.832		1.000	0.714	0.606
F-ID TP G1	0.673	1.000		1.000	0.670
R-ID AP G1	0.395	0.714	1.000		0.903
M-ID TP G1 n-10	0.352	0.606	0.670	0.903	
M-ID TP G1 n-25	0.153	0.303	0.284	0.548	0.554
I-ID TP G1 n-25	0.443	0.396	0.670	0.903	0.905
M-ID TP G1 n-50	0.172	0.202	0.259	0.440	0.574
I-ID AP G1 n-10	0.234	0.291	0.349	0.587	0.609
R-ID AP	0.109	0.181	0.262	0.308	0.346
ADF	0.172	0.307	0.259	0.532	0.593
R-ID TP G1	7.636×10^{-2}	0.184	0.185	0.189	0.500
M-ID AP G1 n-10	6.371×10^{-2}	0.132	0.108	0.227	0.324
I-ID TP n-10	7.479×10^{-2}	0.166	0.196	0.246	0.401
F-ID TP	8.924×10^{-2}	0.151	0.210	0.270	0.342
M-ID AP G1 n-25	3.977×10^{-2}*	5.627×10^{-2}	7.913×10^{-2}	0.123	0.141
F-ID AP	1.207×10^{-2}*	5.851×10^{-2}	6.685×10^{-2}	6.767×10^{-2}	0.134
R-ID TP	4.580×10^{-2}*	8.570×10^{-2}	0.107	0.138	0.190
I-ID AP G1 n-25	2.579×10^{-2}*	1.353×10^{-2}*	7.585×10^{-2}	5.639×10^{-2}	0.145
I-ID TP G1 n-10	2.012×10^{-2}*	2.601×10^{-2}*	6.711×10^{-2}	6.028×10^{-2}	9.990×10^{-2}
M-ID TP n-10	1.229×10^{-2}*	2.615×10^{-2}*	2.090×10^{-2}*	3.744×10^{-2}*	6.967×10^{-2}
M-ID TP n-25	9.604×10^{-3}*	2.425×10^{-2}*	2.960×10^{-2}*	2.946×10^{-2}*	5.790×10^{-2}
GE	1.648×10^{-2}*	2.738×10^{-2}*	3.894×10^{-2}*	4.673×10^{-2}*	7.787×10^{-2}
I-ID TP n-50	5.432×10^{-3}*	1.161×10^{-2}*	1.016×10^{-2}*	1.526×10^{-2}*	4.126×10^{-2}*
I-ID TP n-25	5.038×10^{-3}*	1.860×10^{-2}*	1.499×10^{-2}*	2.077×10^{-2}*	1.576×10^{-2}*
M-ID TP n-50	1.107×10^{-2}*	1.103×10^{-2}*	8.059×10^{-3}*	4.410×10^{-2}*	5.034×10^{-2}
I-ID AP n-10	6.877×10^{-3}*	1.434×10^{-2}*	1.814×10^{-2}*	2.738×10^{-2}*	3.684×10^{-2}*
M-ID AP n-10	1.984×10^{-3}*	2.749×10^{-3}*	1.239×10^{-3}*	5.795×10^{-3}*	1.847×10^{-2}*
M-ID AP n-25	7.835×10^{-4}*	4.754×10^{-3}*	5.622×10^{-3}*	5.306×10^{-3}*	8.307×10^{-3}*
I-ID AP n-25	4.910×10^{-4}*	3.931×10^{-4}*	5.906×10^{-4}*	1.028×10^{-3}*	1.230×10^{-3}*

	M-ID TP G1 n-25	I-ID TP G1 n-25	M-ID TP G1 n-50	I-ID AP G1 n-10	R-ID AP
F-ID AP G1	0.153	0.443	0.172	0.234	0.109
I-ID TP G1 n-50	0.303	0.396	0.202	0.291	0.181
F-ID TP G1	0.284	0.670	0.259	0.349	0.262
R-ID AP G1	0.548	0.903	0.440	0.587	0.308
M-ID TP G1 n-10	0.554	0.905	0.574	0.609	0.346
M-ID TP G1 n-25		0.723	0.972	0.916	0.711
I-ID TP G1 n-25	0.723		0.758	0.837	0.646
M-ID TP G1 n-50	0.972	0.758		0.937	0.925
I-ID AP G1 n-10	0.916	0.837	0.937		0.777
R-ID AP	0.711	0.646	0.925	0.777	
ADF	1.000	0.607	0.813	0.916	0.826
R-ID TP G1	0.725	0.681	0.781	0.783	0.844
M-ID AP G1 n-10	0.528	0.481	0.623	0.592	0.836
I-ID TP n-10	0.431	0.475	0.753	0.674	0.865
F-ID TP	0.551	0.501	0.725	0.636	0.856
M-ID AP G1 n-25	0.295	0.474	0.503	0.488	0.508
F-ID AP	0.319	0.284	0.455	0.401	0.426
R-ID TP	0.419	0.381	0.314	0.378	0.660

Even 7 Parity Wilcoxon rank-sun test results continued.

I-ID AP G1 n-25	0.252	0.269	0.194	0.347	0.484
I-ID TP G1 n-10	0.200	0.198	0.311	0.298	0.420
M-ID TP n-10	0.204	0.131	0.234	0.218	0.297
M-ID TP n-25	5.772×10^{-2}	0.129	0.184	9.298×10^{-2}	0.227
GE	0.102	0.186	0.221	0.223	0.193
I-ID TP n-50	8.936×10^{-2}	6.729×10^{-2}	0.184	0.116	0.177
I-ID TP n-25	9.607×10^{-2}	0.103	0.161	6.447×10^{-2}	0.139
M-ID TP n-50	0.158	8.225×10^{-2}	0.184	9.263×10^{-2}	0.204
I-ID AP n-10	8.651×10^{-2}	9.671×10^{-2}	0.111	0.112	0.130
M-ID AP n-10	7.934×10^{-3}*	4.197×10^{-2}*	3.214×10^{-2}*	3.576×10^{-2}*	6.130×10^{-2}
M-ID AP n-25	1.310×10^{-2}*	3.793×10^{-2}*	4.964×10^{-2}*	4.731×10^{-2}*	1.243×10^{-2}*
I-ID AP n-25	8.320×10^{-3}*	2.192×10^{-2}*	1.564×10^{-2}*	1.359×10^{-2}*	1.004×10^{-2}*

	ADF	R-ID TP G1	M-ID AP G1 n-10	I-ID TP n-10	F-ID TP
F-ID AP G1	0.172	7.636×10^{-2}	6.371×10^{-2}	7.479×10^{-2}	8.924×10^{-2}
I-ID TP G1 n-50	0.307	0.184	0.132	0.166	0.151
F-ID TP G1	0.259	0.185	0.108	0.196	0.210
R-ID AP G1	0.532	0.189	0.227	0.246	0.270
M-ID TP G1 n-10	0.593	0.500	0.324	0.401	0.342
M-ID TP G1 n-25	1.000	0.725	0.528	0.431	0.551
I-ID TP G1 n-25	0.607	0.681	0.481	0.475	0.501
M-ID TP G1 n-50	0.813	0.781	0.623	0.753	0.725
I-ID AP G1 n-10	0.916	0.783	0.592	0.674	0.636
R-ID AP	0.826	0.844	0.836	0.865	0.856
ADF		0.844	0.589	0.624	0.801
R-ID TP G1	0.844		0.863	1.000	0.950
M-ID AP G1 n-10	0.589	0.863		0.955	0.864
I-ID TP n-10	0.624	1.000	0.955		1.000
F-ID TP	0.801	0.950	0.864	1.000	
M-ID AP G1 n-25	0.483	0.758	0.796	0.776	0.864
F-ID AP	0.531	0.752	0.641	0.622	0.622
R-ID TP	0.362	0.614	0.831	0.551	0.740
I-ID AP G1 n-25	0.509	0.590	0.568	0.550	0.640
I-ID TP G1 n-10	0.330	0.755	0.568	0.521	0.515
M-ID TP n-10	0.266	0.324	0.367	0.400	0.437
M-ID TP n-25	0.178	0.275	0.327	0.355	0.383
GE	0.293	0.299	0.371	0.293	0.319
I-ID TP n-50	0.116	0.196	0.168	0.234	0.250
I-ID TP n-25	0.154	0.380	0.358	0.137	0.302
M-ID TP n-50	0.230	0.285	0.211	0.184	0.330
I-ID AP n-10	0.138	0.182	0.227	0.267	0.285
M-ID AP n-10	3.437×10^{-2}*	0.116	9.023×10^{-2}	0.171	8.513×10^{-2}
M-ID AP n-25	5.429×10^{-2}	8.080×10^{-2}	7.302×10^{-2}	0.102	0.102
I-ID AP n-25	1.667×10^{-2}*	3.481×10^{-2}*	2.400×10^{-2}*	2.288×10^{-2}*	4.233×10^{-2}*

	M-ID AP G1 n-25	F-ID AP	R-ID TP	I-ID AP G1 n-25	I-ID TP G1 n-10
F-ID AP G1	3.977×10^{-2}*	1.207×10^{-2}*	4.580×10^{-2}*	2.579×10^{-2}*	2.012×10^{-2}*
I-ID TP G1 n-50	5.627×10^{-2}	5.851×10^{-2}	8.570×10^{-2}	1.353×10^{-2}*	2.601×10^{-2}*
F-ID TP G1	7.913×10^{-2}	6.685×10^{-2}	0.107	7.585×10^{-2}	6.711×10^{-2}
R-ID AP G1	0.123	6.767×10^{-2}	0.138	5.639×10^{-2}	6.028×10^{-2}
M-ID TP G1 n-10	0.141	0.134	0.190	0.145	9.990×10^{-2}
M-ID TP G1 n-25	0.295	0.319	0.419	0.252	0.200
I-ID TP G1 n-25	0.474	0.284	0.381	0.269	0.198
M-ID TP G1 n-50	0.503	0.455	0.314	0.194	0.311
I-ID AP G1 n-10	0.488	0.401	0.378	0.347	0.298
R-ID AP	0.508	0.426	0.660	0.484	0.420

Even 7 Parity Wilcoxon rank-sun test results continued.

ADF	0.483	0.531	0.362	0.509	0.330
R-ID TP G1	0.758	0.752	0.614	0.590	0.755
M-ID AP G1 n-10	0.796	0.641	0.831	0.568	0.568
I-ID TP n-10	0.776	0.622	0.551	0.550	0.521
F-ID TP	0.864	0.622	0.740	0.640	0.515
M-ID AP G1 n-25		0.917	0.981	0.695	0.754
F-ID AP	0.917		1.000	0.975	0.981
R-ID TP	0.981	1.000		0.955	0.777
I-ID AP G1 n-25	0.695	0.975	0.955		0.981
I-ID TP G1 n-10	0.754	0.981	0.777	0.981	
M-ID TP n-10	0.526	0.831	0.657	0.887	0.762
M-ID TP n-25	0.492	0.711	0.614	0.745	0.626
GE	0.442	0.553	0.694	0.604	0.541
I-ID TP n-50	0.347	0.684	0.410	0.517	0.473
I-ID TP n-25	0.547	0.680	0.419	0.726	0.668
M-ID TP n-50	0.456	0.546	0.359	0.485	0.625
I-ID AP n-10	0.373	0.465	0.331	0.467	0.513
M-ID AP n-10	0.168	0.204	0.166	0.285	0.315
M-ID AP n-25	0.117	0.248	0.171	0.296	0.280
I-ID AP n-25	3.205×10^{-2}*	4.451×10^{-2}*	9.812×10^{-2}	0.106	8.085×10^{-2}

	M-ID TP n-10	M-ID TP n-25	GE	I-ID TP n-50	I-ID TP n-25
F-ID AP G1	1.229×10^{-2}*	9.604×10^{-3}*	1.648×10^{-2}*	5.432×10^{-3}*	5.038×10^{-3}*
I-ID TP G1 n-50	2.615×10^{-2}*	2.425×10^{-2}*	2.738×10^{-2}*	1.161×10^{-2}*	1.860×10^{-2}*
F-ID TP G1	2.090×10^{-2}*	2.960×10^{-2}*	3.894×10^{-2}*	1.016×10^{-2}*	1.499×10^{-2}*
R-ID AP G1	3.744×10^{-2}*	2.946×10^{-2}*	4.673×10^{-2}*	1.526×10^{-2}*	2.077×10^{-2}*
M-ID TP G1 n-10	6.967×10^{-2}	5.790×10^{-2}	7.787×10^{-2}	4.126×10^{-2}*	1.576×10^{-2}*
M-ID TP G1 n-25	0.204	5.772×10^{-2}	0.102	8.936×10^{-2}	9.607×10^{-2}
I-ID TP G1 n-25	0.131	0.129	0.186	6.729×10^{-2}	0.103
M-ID TP G1 n-50	0.234	0.184	0.221	0.184	0.161
I-ID AP G1 n-10	0.218	9.298×10^{-2}	0.223	0.116	6.447×10^{-2}
R-ID AP	0.297	0.227	0.193	0.177	0.139
ADF	0.266	0.178	0.293	0.116	0.154
R-ID TP G1	0.324	0.275	0.299	0.196	0.380
M-ID AP G1 n-10	0.367	0.327	0.371	0.168	0.358
I-ID TP n-10	0.400	0.355	0.293	0.234	0.137
F-ID TP	0.437	0.383	0.319	0.250	0.302
M-ID AP G1 n-25	0.526	0.492	0.442	0.347	0.547
F-ID AP	0.831	0.711	0.553	0.684	0.680
R-ID TP	0.657	0.614	0.694	0.410	0.419
I-ID AP G1 n-25	0.887	0.745	0.604	0.517	0.726
I-ID TP G1 n-10	0.762	0.626	0.541	0.473	0.668
M-ID TP n-10		0.931	0.807	0.613	0.807
M-ID TP n-25	0.931		0.952	0.819	0.726
GE	0.807	0.952		1.000	0.917
I-ID TP n-50	0.613	0.819	1.000		0.943
I-ID TP n-25	0.807	0.726	0.917	0.943	
M-ID TP n-50	0.964	0.988	0.856	0.796	0.845
I-ID AP n-10	0.795	0.764	0.777	0.963	0.939
M-ID AP n-10	0.439	0.316	0.428	0.709	0.417
M-ID AP n-25	0.265	0.394	0.410	0.620	0.528
I-ID AP n-25	9.865×10^{-2}	0.172	0.245	0.293	0.170

	M-ID TP n-50	I-ID AP n-10	M-ID AP n-10	M-ID AP n-25	I-ID AP n-25
F-ID AP G1	1.107×10^{-2}*	6.877×10^{-3}*	1.984×10^{-3}*	7.835×10^{-4}*	4.910×10^{-4}*
I-ID TP G1 n-50	1.103×10^{-2}*	1.434×10^{-2}*	2.749×10^{-3}*	4.754×10^{-3}*	3.931×10^{-4}*

Even 7 Parity Wilcoxon rank-sun test results continued.

F-ID TP G1	8.059×10^{-3}*	1.814×10^{-2}*	1.239×10^{-3}*	5.622×10^{-3}*	5.906×10^{-4}*
R-ID AP G1	4.410×10^{-2}*	2.738×10^{-2}*	5.795×10^{-3}*	5.306×10^{-3}*	1.028×10^{-3}*
M-ID TP G1 n-10	5.034×10^{-2}	3.684×10^{-2}*	1.847×10^{-2}*	8.307×10^{-3}*	1.230×10^{-3}*
M-ID TP G1 n-25	0.158	8.651×10^{-2}	7.934×10^{-3}*	1.310×10^{-2}*	8.320×10^{-3}*
I-ID TP G1 n-25	8.225×10^{-2}	9.671×10^{-2}	4.197×10^{-2}*	3.793×10^{-2}*	2.192×10^{-2}*
M-ID TP G1 n-50	0.184	0.111	3.214×10^{-2}*	4.964×10^{-2}*	1.564×10^{-2}*
I-ID AP G1 n-10	9.263×10^{-2}	0.112	3.576×10^{-2}*	4.731×10^{-2}*	1.359×10^{-2}*
R-ID AP	0.204	0.130	6.130×10^{-2}	1.243×10^{-2}*	1.004×10^{-2}*
ADF	0.230	0.138	3.437×10^{-2}*	5.429×10^{-2}	1.667×10^{-2}*
R-ID TP G1	0.285	0.182	0.116	8.080×10^{-2}	3.481×10^{-2}*
M-ID AP G1 n-10	0.211	0.227	9.023×10^{-2}	7.302×10^{-2}	2.400×10^{-2}*
I-ID TP n-10	0.184	0.267	0.171	0.102	2.288×10^{-2}*
F-ID TP	0.330	0.285	8.513×10^{-2}	0.102	4.233×10^{-2}*
M-ID AP G1 n-25	0.456	0.373	0.168	0.117	3.205×10^{-2}*
F-ID AP	0.546	0.465	0.204	0.248	4.451×10^{-2}*
R-ID TP	0.359	0.331	0.166	0.171	9.812×10^{-2}
I-ID AP G1 n-25	0.485	0.467	0.285	0.296	0.106
I-ID TP G1 n-10	0.625	0.513	0.315	0.280	8.085×10^{-2}
M-ID TP n-10	0.964	0.795	0.439	0.265	9.865×10^{-2}
M-ID TP n-25	0.988	0.764	0.316	0.394	0.172
GE	0.856	0.777	0.428	0.410	0.245
I-ID TP n-50	0.796	0.963	0.709	0.620	0.293
I-ID TP n-25	0.845	0.939	0.417	0.528	0.170
M-ID TP n-50		0.819	0.489	0.399	0.234
I-ID AP n-10	0.819		0.586	0.586	0.374
M-ID AP n-10	0.489	0.586		0.952	0.499
M-ID AP n-25	0.399	0.586	0.952		0.613
I-ID AP n-25	0.234	0.374	0.499	0.613	

Table A.29: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the $x^5 - 2x^3 + x$ Symbolic Regression problem after 100000 fitness evaluations of each approach to identifying modules, GE, and GE with ADFs.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
R-ID AP	0.284	0.459	0.065	13
R-ID TP G1	0.293	0.433	0.061	13
F-ID AP	0.349	0.511	0.072	14
GE	0.364	0.244	0.034	4
F-ID TP	0.410	0.504	0.071	12
I-ID TP n-25	0.417	0.446	0.063	13
F-ID TP G1	0.425	0.494	0.070	8
I-ID TP n-10	0.429	0.453	0.064	11
I-ID TP G1 n-50	0.471	0.375	0.053	8
F-ID AP G1	0.473	0.474	0.067	5
I-ID AP n-10	0.479	0.483	0.068	13
R-ID TP	0.481	0.568	0.080	10
R-ID AP G1	0.483	0.681	0.096	12
I-ID TP n-50	0.488	0.500	0.071	11
M-ID TP G1 n-10	0.509	0.485	0.069	9
ADF	0.512	0.564	0.080	9
M-ID TP G1 n-25	0.534	0.506	0.072	4
I-ID TP G1 n-25	0.538	0.453	0.064	7
I-ID AP G1 n-25	0.540	0.448	0.063	6
M-ID AP n-10	0.541	0.604	0.085	15
I-ID AP n-25	0.542	0.501	0.071	9
M-ID AP n-25	0.565	0.586	0.083	10
M-ID TP G1 n-50	0.588	0.591	0.084	10
M-ID TP n-25	0.591	0.609	0.086	9
I-ID TP G1 n-10	0.620	0.546	0.077	5
I-ID AP G1 n-10	0.627	0.587	0.083	6
M-ID TP n-50	0.640	0.616	0.087	7
M-ID TP n-10	0.657	0.654	0.093	9
M-ID AP G1 n-10	0.721	0.629	0.089	5
M-ID AP G1 n-25	0.937	0.640	0.090	1

Table A.30: This table reports the p-value of Wilcoxon rank-sum tests performe best fitness values of each approach to identifying modules, GE, and GE with AI fitness evaluations on the $x^5 - 2x^3 + x$ Symbolic Regression problem. The p-val calculated with a confidence interval of 0.05. Values marked with an asterisk (*

	R-ID AP	R-ID TP G1	F-ID AP	GE	F-ID TP
R-ID AP		0.900	0.493	2.512×10^{-2}*	0.107
R-ID TP G1	0.900		0.931	8.059×10^{-2}	0.235
F-ID AP	0.493	0.931		0.112	0.244
GE	2.512×10^{-2}*	8.059×10^{-2}	0.112		0.664
F-ID TP	0.107	0.235	0.244	0.664	
I-ID TP n-25	9.208×10^{-2}	7.412×10^{-2}	0.406	0.915	0.851
F-ID TP G1	3.211×10^{-2}*	0.215	0.179	0.958	0.640
I-ID TP n-10	4.118×10^{-2}*	5.784×10^{-2}	0.107	0.854	0.985
I-ID TP G1 n-50	1.613×10^{-3}*	1.097×10^{-2}*	3.827×10^{-2}*	0.114	0.108
F-ID AP G1	9.410×10^{-3}*	1.941×10^{-2}*	2.441×10^{-2}*	0.286	0.133
I-ID AP n-10	1.383×10^{-2}*	9.301×10^{-2}	6.943×10^{-2}	0.367	0.388
R-ID TP	4.675×10^{-2}*	1.310×10^{-2}*	0.180	0.935	0.612
R-ID AP G1	0.208	0.395	0.662	0.772	0.892
I-ID TP n-50	2.105×10^{-2}*	3.410×10^{-2}*	0.276	0.357	0.348
M-ID TP G1 n-10	2.007×10^{-3}*	4.042×10^{-3}*	1.966×10^{-2}*	0.164	0.113
ADF	3.917×10^{-3}*	2.187×10^{-2}*	3.662×10^{-2}*	0.216	8.752×10^{-2}
M-ID TP G1 n-25	1.291×10^{-3}*	5.463×10^{-4}*	2.480×10^{-2}*	7.022×10^{-2}	7.413×10^{-2}
I-ID TP G1 n-25	5.684×10^{-4}*	3.352×10^{-4}*	5.178×10^{-3}*	1.563×10^{-2}*	4.501×10^{-2}*
I-ID AP G1 n-25	2.795×10^{-5}*	1.030×10^{-3}*	1.328×10^{-3}*	1.673×10^{-2}*	4.651×10^{-3}*
M-ID AP n-10	4.782×10^{-2}*	6.579×10^{-3}*	1.782×10^{-2}*	0.304	0.313
I-ID AP n-25	2.295×10^{-3}*	2.113×10^{-3}*	3.083×10^{-2}*	0.104	9.489×10^{-2}
M-ID AP n-25	1.224×10^{-2}*	6.482×10^{-3}*	1.874×10^{-2}*	0.131	0.122
M-ID TP G1 n-50	1.912×10^{-3}*	3.853×10^{-3}*	5.911×10^{-2}	8.033×10^{-2}	6.449×10^{-2}
M-ID TP n-25	4.821×10^{-3}*	1.374×10^{-3}*	8.646×10^{-3}*	0.102	6.382×10^{-2}
I-ID TP G1 n-10	5.022×10^{-5}*	7.153×10^{-4}*	2.453×10^{-3}*	8.645×10^{-3}*	5.600×10^{-3}*
I-ID AP G1 n-10	1.001×10^{-4}*	1.028×10^{-3}*	2.637×10^{-3}*	2.300×10^{-2}*	1.401×10^{-2}*
M-ID TP n-50	4.924×10^{-4}*	4.456×10^{-4}*	1.082×10^{-2}*	6.266×10^{-2}	4.503×10^{-2}*
M-ID TP n-10	3.041×10^{-3}*	4.713×10^{-4}*	1.720×10^{-2}*	4.776×10^{-2}*	3.626×10^{-2}*
M-ID AP G1 n-10	6.363×10^{-5}*	1.936×10^{-5}*	3.000×10^{-4}*	2.843×10^{-3}*	1.482×10^{-3}*
M-ID AP G1 n-25	2.660×10^{-6}*	4.811×10^{-7}*	1.980×10^{-5}*	1.402×10^{-6}*	2.853×10^{-5}*

	I-ID TP n-25	F-ID TP G1	I-ID TP n-10	I-ID TP G1 n-50	F-ID AP G1
R-ID AP	9.208×10^{-2}	3.211×10^{-2}*	4.118×10^{-2}*	1.613×10^{-3}*	9.410×10^{-3}*
R-ID TP G1	7.412×10^{-2}	0.215	5.784×10^{-2}	1.097×10^{-2}*	1.941×10^{-2}*
F-ID AP	0.406	0.179	0.107	3.827×10^{-2}*	2.441×10^{-2}*
GE	0.915	0.958	0.854	0.114	0.286
F-ID TP	0.851	0.640	0.985	0.108	0.133
I-ID TP n-25		0.969	0.946	0.246	0.271
F-ID TP G1	0.969		0.772	0.111	0.385
I-ID TP n-10	0.946	0.772		0.283	0.398
I-ID TP G1 n-50	0.246	0.111	0.283		0.900
F-ID AP G1	0.271	0.385	0.398	0.900	
I-ID AP n-10	0.348	0.598	0.968	0.345	1.000
R-ID TP	0.567	0.931	0.743	0.354	0.724
R-ID AP G1	0.585	0.889	0.996	0.155	0.382
I-ID TP n-50	0.296	0.315	0.672	0.996	0.877
M-ID TP G1 n-10	0.275	0.299	0.438	1.000	0.728
ADF	0.330	0.267	0.944	0.870	0.958
M-ID TP G1 n-25	4.502×10^{-2}*	5.875×10^{-2}	7.827×10^{-2}	0.427	0.441
I-ID TP G1 n-25	0.129	0.119	6.009×10^{-2}	0.663	0.330

$x^5 - 2x^3 + x$ Symbolic Regression Wilcoxon rank-sun test results continued.

I-ID AP G1 n-25	4.291×10^{-2}*	1.440×10^{-2}*	4.212×10^{-2}*	0.598	0.181
M-ID AP n-10	0.320	0.489	0.372	0.720	0.841
I-ID AP n-25	0.200	7.497×10^{-2}	0.153	0.743	0.532
M-ID AP n-25	9.762×10^{-2}	9.970×10^{-2}	0.115	0.412	0.548
M-ID TP G1 n-50	0.106	0.236	0.113	0.505	0.294
M-ID TP n-25	7.266×10^{-2}	0.255	0.372	0.531	0.553
I-ID TP G1 n-10	1.134×10^{-2}*	4.720×10^{-2}*	2.707×10^{-2}*	0.106	0.128
I-ID AP G1 n-10	1.898×10^{-2}*	9.273×10^{-2}	4.249×10^{-2}*	0.341	8.985×10^{-2}
M-ID TP n-50	6.807×10^{-2}	7.730×10^{-2}	4.944×10^{-2}*	0.648	0.231
M-ID TP n-10	5.595×10^{-2}	3.534×10^{-2}*	0.145	0.294	0.249
M-ID AP G1 n-10	2.639×10^{-2}*	7.074×10^{-3}*	2.695×10^{-3}*	0.116	4.311×10^{-2}*
M-ID AP G1 n-25	1.544×10^{-6}*	1.124×10^{-5}*	4.076×10^{-5}*	9.650×10^{-6}*	1.147×10^{-4}*

	I-ID AP n-10	R-ID TP	R-ID AP G1	I-ID TP n-50	M-ID TP G1 n-10
R-ID AP	1.383×10^{-2}*	4.675×10^{-2}*	0.208	2.105×10^{-2}*	2.007×10^{-3}*
R-ID TP G1	9.301×10^{-2}	1.310×10^{-2}*	0.395	3.410×10^{-2}*	4.042×10^{-3}*
F-ID AP	6.943×10^{-2}	0.180	0.662	0.276	1.966×10^{-2}*
GE	0.367	0.935	0.772	0.357	0.164
F-ID TP	0.388	0.612	0.892	0.348	0.113
I-ID TP n-25	0.348	0.567	0.585	0.296	0.275
F-ID TP G1	0.598	0.931	0.889	0.315	0.299
I-ID TP n-10	0.968	0.743	0.996	0.672	0.438
I-ID TP G1 n-50	0.345	0.354	0.155	0.996	1.000
F-ID AP G1	1.000	0.724	0.382	0.877	0.728
I-ID AP n-10		0.657	0.499	0.765	0.597
R-ID TP	0.657		0.667	0.802	0.462
R-ID AP G1	0.499	0.667		0.537	0.263
I-ID TP n-50	0.765	0.802	0.537		0.950
M-ID TP G1 n-10	0.597	0.462	0.263	0.950	
ADF	0.889	0.322	0.451	0.889	0.893
M-ID TP G1 n-25	0.322	0.233	0.137	0.527	0.303
I-ID TP G1 n-25	0.278	4.664×10^{-2}*	9.056×10^{-2}	0.479	0.581
I-ID AP G1 n-25	0.260	0.125	0.101	7.806×10^{-2}	0.317
M-ID AP n-10	0.866	0.691	0.415	0.893	0.758
I-ID AP n-25	0.652	0.183	0.118	0.800	0.489
M-ID AP n-25	0.148	0.211	0.149	0.599	0.787
M-ID TP G1 n-50	0.238	0.357	0.111	0.571	0.281
M-ID TP n-25	0.469	0.320	0.306	0.330	0.839
I-ID TP G1 n-10	2.637×10^{-2}*	6.216×10^{-2}	3.618×10^{-2}*	0.259	0.248
I-ID AP G1 n-10	4.163×10^{-2}*	1.402×10^{-2}*	8.747×10^{-2}	0.210	0.125
M-ID TP n-50	0.194	5.596×10^{-2}	9.178×10^{-2}	0.385	0.431
M-ID TP n-10	0.172	0.193	0.122	0.255	0.505
M-ID AP G1 n-10	1.649×10^{-2}*	8.635×10^{-3}*	1.991×10^{-2}*	0.103	4.833×10^{-2}*
M-ID AP G1 n-25	6.092×10^{-5}*	1.689×10^{-5}*	3.049×10^{-4}*	5.840×10^{-5}*	3.342×10^{-5}*

	ADF	M-ID TP G1 n-25	I-ID TP G1 n-25	I-ID AP G1 n-25	M-ID AP n-10
R-ID AP	3.917×10^{-3}*	1.291×10^{-3}*	5.684×10^{-4}*	2.795×10^{-5}*	4.782×10^{-2}*
R-ID TP G1	2.187×10^{-2}*	5.463×10^{-4}*	3.352×10^{-4}*	1.030×10^{-3}*	6.579×10^{-3}*
F-ID AP	3.662×10^{-2}*	2.480×10^{-2}*	5.178×10^{-3}*	1.328×10^{-3}*	1.782×10^{-2}*
GE	0.216	7.022×10^{-2}	1.563×10^{-2}*	1.673×10^{-2}*	0.304
F-ID TP	8.752×10^{-2}	7.413×10^{-2}	4.501×10^{-2}*	4.651×10^{-3}*	0.313
I-ID TP n-25	0.330	4.502×10^{-2}*	0.129	4.291×10^{-2}*	0.320
F-ID TP G1	0.267	5.875×10^{-2}	0.119	1.440×10^{-2}*	0.489
I-ID TP n-10	0.944	7.827×10^{-2}	6.009×10^{-2}	4.212×10^{-2}*	0.372
I-ID TP G1 n-50	0.870	0.427	0.663	0.598	0.720
F-ID AP G1	0.958	0.441	0.330	0.181	0.841

$x^5 - 2x^3 + x$ Symbolic Regression Wilcoxon rank-sun test results continued.

I-ID AP n-10	0.889	0.322	0.278	0.260	0.866
R-ID TP	0.322	0.233	4.664×10^{-2}*	0.125	0.691
R-ID AP G1	0.451	0.137	9.056×10^{-2}	0.101	0.415
I-ID TP n-50	0.889	0.527	0.479	7.806×10^{-2}	0.893
M-ID TP G1 n-10	0.893	0.303	0.581	0.317	0.758
ADF		0.701	0.401	0.394	0.780
M-ID TP G1 n-25	0.701		0.789	0.509	0.313
I-ID TP G1 n-25	0.401	0.789		0.926	0.605
I-ID AP G1 n-25	0.394	0.509	0.926		0.735
M-ID AP n-10	0.780	0.313	0.605	0.735	
I-ID AP n-25	0.588	0.980	0.365	1.000	0.762
M-ID AP n-25	0.404	0.858	0.662	0.555	0.882
M-ID TP G1 n-50	0.435	0.779	0.813	0.848	0.255
M-ID TP n-25	0.773	0.717	0.717	0.601	0.885
I-ID TP G1 n-10	6.497×10^{-2}	0.313	0.680	0.454	0.317
I-ID AP G1 n-10	0.140	0.257	0.406	0.650	6.643×10^{-2}
M-ID TP n-50	0.240	0.835	0.591	0.858	0.415
M-ID TP n-10	0.111	0.674	0.909	0.671	0.798
M-ID AP G1 n-10	0.120	0.174	0.731	0.198	4.292×10^{-2}*
M-ID AP G1 n-25	5.432×10^{-4}*	7.052×10^{-4}*	3.314×10^{-5}*	9.557×10^{-4}*	2.233×10^{-3}*

	I-ID AP n-25	M-ID AP n-25	M-ID TP G1 n-50	M-ID TP n-25	I-ID TP G1 n-10
R-ID AP	2.295×10^{-3}*	1.224×10^{-2}*	1.912×10^{-3}*	4.821×10^{-3}*	5.022×10^{-5}*
R-ID TP G1	2.113×10^{-3}*	6.482×10^{-3}*	3.853×10^{-3}*	1.374×10^{-3}*	7.153×10^{-4}*
F-ID AP	3.083×10^{-2}*	1.874×10^{-2}*	5.911×10^{-2}	8.646×10^{-3}*	2.453×10^{-3}*
GE	0.104	0.131	8.033×10^{-2}	0.102	8.645×10^{-3}*
F-ID TP	9.489×10^{-2}	0.122	6.449×10^{-2}	6.382×10^{-2}	5.600×10^{-3}*
I-ID TP n-25	0.200	9.762×10^{-2}	0.106	7.266×10^{-2}	1.134×10^{-2}*
F-ID TP G1	7.497×10^{-2}	9.970×10^{-2}	0.236	0.255	4.720×10^{-2}*
I-ID TP n-10	0.153	0.115	0.113	0.372	2.707×10^{-2}*
I-ID TP G1 n-50	0.743	0.412	0.505	0.531	0.106
F-ID AP G1	0.532	0.548	0.294	0.553	0.128
I-ID AP n-10	0.652	0.148	0.238	0.469	2.637×10^{-2}*
R-ID TP	0.183	0.211	0.357	0.320	6.216×10^{-2}
R-ID AP G1	0.118	0.149	0.111	0.306	3.618×10^{-2}*
I-ID TP n-50	0.800	0.599	0.571	0.330	0.259
M-ID TP G1 n-10	0.489	0.787	0.281	0.839	0.248
ADF	0.588	0.404	0.435	0.773	6.497×10^{-2}
M-ID TP G1 n-25	0.980	0.858	0.779	0.717	0.313
I-ID TP G1 n-25	0.365	0.662	0.813	0.717	0.680
I-ID AP G1 n-25	1.000	0.555	0.848	0.601	0.454
M-ID AP n-10	0.762	0.882	0.255	0.885	0.317
I-ID AP n-25		0.963	0.685	0.908	0.515
M-ID AP n-25	0.963		0.694	0.735	0.295
M-ID TP G1 n-50	0.685	0.694		0.823	0.552
M-ID TP n-25	0.908	0.735	0.823		0.225
I-ID TP G1 n-10	0.515	0.295	0.552	0.225	
I-ID AP G1 n-10	0.454	0.532	0.719	0.471	0.796
M-ID TP n-50	0.866	0.521	0.858	0.689	0.637
M-ID TP n-10	0.913	0.554	0.477	0.901	0.426
M-ID AP G1 n-10	0.292	9.469×10^{-2}	0.310	0.197	0.525
M-ID AP G1 n-25	2.897×10^{-5}*	1.044×10^{-3}*	4.335×10^{-3}*	7.511×10^{-5}*	4.414×10^{-3}*

	I-ID AP G1 n-10	M-ID TP n-50	M-ID TP n-10	M-ID AP G1 n-10	M-ID AP G1 n-25
R-ID AP	1.001×10^{-4}*	4.924×10^{-4}*	3.041×10^{-3}*	6.363×10^{-5}*	2.660×10^{-6}*
R-ID TP G1	1.028×10^{-3}*	4.456×10^{-4}*	4.713×10^{-4}*	1.936×10^{-5}*	4.811×10^{-7}*

$x^5 - 2x^3 + x$ Symbolic Regression Wilcoxon rank-sun test results continued.

F-ID AP	2.637×10^{-3}*	1.082×10^{-2}*	1.720×10^{-2}*	3.000×10^{-4}*	1.980×10^{-5}*
GE	2.300×10^{-2}*	6.266×10^{-2}	4.776×10^{-2}*	2.843×10^{-3}*	1.402×10^{-6}*
F-ID TP	1.401×10^{-2}*	4.503×10^{-2}*	3.626×10^{-2}*	1.482×10^{-3}*	2.853×10^{-5}*
I-ID TP n-25	1.898×10^{-2}*	6.807×10^{-2}	5.595×10^{-2}	2.639×10^{-2}*	1.544×10^{-6}*
F-ID TP G1	9.273×10^{-2}	7.730×10^{-2}	3.534×10^{-2}*	7.074×10^{-3}*	1.124×10^{-5}*
I-ID TP n-10	4.249×10^{-2}*	4.944×10^{-2}*	0.145	2.695×10^{-3}*	4.076×10^{-5}*
I-ID TP G1 n-50	0.341	0.648	0.294	0.116	9.650×10^{-6}*
F-ID AP G1	8.985×10^{-2}	0.231	0.249	4.311×10^{-2}*	1.147×10^{-4}*
I-ID AP n-10	4.163×10^{-2}*	0.194	0.172	1.649×10^{-2}*	6.092×10^{-5}*
R-ID TP	1.402×10^{-2}*	5.596×10^{-2}	0.193	8.635×10^{-3}*	1.689×10^{-5}*
R-ID AP G1	8.747×10^{-2}	9.178×10^{-2}	0.122	1.991×10^{-2}*	3.049×10^{-4}*
I-ID TP n-50	0.210	0.385	0.255	0.103	5.840×10^{-5}*
M-ID TP G1 n-10	0.125	0.431	0.505	4.833×10^{-2}*	3.342×10^{-5}*
ADF	0.140	0.240	0.111	0.120	5.432×10^{-4}*
M-ID TP G1 n-25	0.257	0.835	0.674	0.174	7.052×10^{-4}*
I-ID TP G1 n-25	0.406	0.591	0.909	0.731	3.314×10^{-5}*
I-ID AP G1 n-25	0.650	0.858	0.671	0.198	9.557×10^{-4}*
M-ID AP n-10	6.643×10^{-2}	0.415	0.798	4.292×10^{-2}*	2.233×10^{-3}*
I-ID AP n-25	0.454	0.866	0.913	0.292	2.897×10^{-5}*
M-ID AP n-25	0.532	0.521	0.554	9.469×10^{-2}	1.044×10^{-3}*
M-ID TP G1 n-50	0.719	0.858	0.477	0.310	4.335×10^{-3}*
M-ID TP n-25	0.471	0.689	0.901	0.197	7.511×10^{-5}*
I-ID TP G1 n-10	0.796	0.637	0.426	0.525	4.414×10^{-3}*
I-ID AP G1 n-10		0.992	0.739	0.633	6.948×10^{-3}*
M-ID TP n-50	0.992		0.866	0.835	4.565×10^{-3}*
M-ID TP n-10	0.739	0.866		0.420	1.113×10^{-2}*
M-ID AP G1 n-10	0.633	0.835	0.420		2.353×10^{-2}*
M-ID AP G1 n-25	6.948×10^{-3}*	4.565×10^{-3}*	1.113×10^{-2}*	2.353×10^{-2}*	

Table A.31: This table shows the average best fitness, standard deviation, standard error, and number of runs which solved the 8×8 Lawn Mower problem after 100000 fitness evaluations of each approach to identifying modules, GE, and GE with ADFs.

	Best Fitness	Std. Dev.	Std. Err.	Number Solved
ADF	1.000×10^{-5}	3.000×10^{-5}	0	47
M-ID AP n-10	0.224	1.58389	0.22400	19
M-ID TP G1 n-50	0.230	1.62632	0.23000	21
M-ID TP G1 n-25	0.238	1.68290	0.23800	24
M-ID AP G1 n-10	0.260	1.83846	0.26000	18
R-ID AP	0.462	2.287	0.323	20
M-ID TP n-10	0.464	2.299	0.325	21
M-ID AP n-25	0.492	2.435	0.344	19
M-ID TP G1 n-10	0.492	2.435	0.344	22
M-ID TP n-50	0.506	2.510	0.355	17
M-ID TP n-25	1.188	3.608	0.510	21
M-ID AP G1 n-25	1.212	3.674	0.520	14
R-ID TP G1	1.318	3.999	0.566	21
R-ID AP G1	1.376	4.219	0.597	17
R-ID TP	1.736	4.354	0.616	24
F-ID TP	5.966	6.894	0.975	15
F-ID AP	9.634	6.510	0.921	3
F-ID TP G1	10.672	6.339	0.896	2
F-ID AP G1	14.886	4.421	0.625	1
I-ID AP n-10	28.288	1.323	0.187	0
I-ID AP G1 n-10	28.422	1.485	0.210	0
I-ID AP n-25	28.532	1.426	0.202	0
I-ID TP n-10	28.556	1.332	0.188	0
I-ID AP G1 n-25	28.700	1.232	0.174	0
I-ID TP n-50	28.718	1.229	0.174	0
I-ID TP G1 n-25	28.860	1.114	0.158	0
I-ID TP n-25	28.914	1.403	0.198	0
I-ID TP G1 n-50	28.928	1.127	0.159	0
I-ID TP G1 n-10	28.940	1.167	0.165	0
GE	29.250	1.155	0.163	0

Table A.32: This table reports the p-value of Wilcoxon rank-sum tests performed on the best fitness values of each approach to identifying modules, GE, and GE with ADFs after fitness evaluations on the 8×8 Lawn Mower problem. The p-values reported are calcula a confidence interval of 0.05. Values marked with an asterisk (*) are significant.

	ADF	M-ID AP n-10	M-ID TP G1 n-50	M-ID TP G1 n-25	M-ID AP G1 n-10
ADF		1.490×10^{-6}*	1.061×10^{-5}*	3.992×10^{-5}*	1.825×10^{-6}*
M-ID AP n-10	1.490×10^{-6}*		0.817	0.188	0.634
M-ID TP G1 n-50	1.061×10^{-5}*	0.817		0.584	0.984
M-ID TP G1 n-25	3.992×10^{-5}*	0.188	0.584		0.289
M-ID AP G1 n-10	1.825×10^{-6}*	0.634	0.984	0.289	
R-ID AP	2.806×10^{-6}*	0.596	0.289	0.198	0.248
M-ID TP n-10	5.016×10^{-6}*	0.380	0.185	0.159	0.393
M-ID AP n-25	1.519×10^{-6}*	0.732	0.633	0.369	0.918
M-ID TP G1 n-10	7.206×10^{-6}*	0.528	0.468	0.143	0.261
M-ID TP n-50	1.923×10^{-6}*	0.839	0.826	0.328	0.879
M-ID TP n-25	6.813×10^{-6}*	0.370	0.237	2.356×10^{-2}*	0.221
M-ID AP G1 n-25	2.717×10^{-7}*	0.201	0.302	3.002×10^{-2}*	0.174
R-ID TP G1	5.636×10^{-6}*	0.608	0.416	0.184	0.665
R-ID AP G1	8.834×10^{-7}*	0.165	8.360×10^{-2}	2.036×10^{-2}*	0.114
R-ID TP	1.929×10^{-5}*	0.389	7.589×10^{-2}	5.750×10^{-2}	0.346
F-ID TP	2.569×10^{-7}*	3.599×10^{-5}*	1.094×10^{-5}*	1.285×10^{-5}*	5.107×10^{-6}*
F-ID AP	4.000×10^{-9}*	1.140×10^{-8}*	3.280×10^{-9}*	4.910×10^{-9}*	1.252×10^{-8}*
F-ID TP G1	1.410×10^{-9}*	9.520×10^{-9}*	6.210×10^{-9}*	4.700×10^{-9}*	1.741×10^{-8}*
F-ID AP G1	1.140×10^{-9}*	1.300×10^{-9}*	8.500×10^{-10}*	1.220×10^{-9}*	1.300×10^{-9}*
I-ID AP n-10	7.600×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-10	7.600×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-25	7.600×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID TP n-10	7.600×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-25	7.400×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP n-50	7.400×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID TP G1 n-25	7.000×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*
I-ID TP n-25	7.500×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP G1 n-50	6.900×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*
I-ID TP G1 n-10	7.300×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
GE	6.100×10^{-10}*	7.700×10^{-10}*	7.500×10^{-10}*	7.500×10^{-10}*	7.600×10^{-10}*

	R-ID AP	M-ID TP n-10	M-ID AP n-25	M-ID TP G1 n-10	M-ID TP n-50
ADF	2.806×10^{-6}*	5.016×10^{-6}*	1.519×10^{-6}*	7.206×10^{-6}*	1.923×10^{-6}*
M-ID AP n-10	0.596	0.380	0.732	0.528	0.839
M-ID TP G1 n-50	0.289	0.185	0.633	0.468	0.826
M-ID TP G1 n-25	0.198	0.159	0.369	0.143	0.328
M-ID AP G1 n-10	0.248	0.393	0.918	0.261	0.879
R-ID AP		0.705	0.431	0.861	0.289
M-ID TP n-10	0.705		0.391	0.995	0.403
M-ID AP n-25	0.431	0.391		0.785	0.694
M-ID TP G1 n-10	0.861	0.995	0.785		0.520
M-ID TP n-50	0.289	0.403	0.694	0.520	
M-ID TP n-25	0.964	0.414	0.406	0.974	0.305
M-ID AP G1 n-25	0.581	0.647	0.514	0.942	0.305
R-ID TP G1	0.665	0.861	0.855	0.880	0.768
R-ID AP G1	0.659	0.303	6.054×10^{-2}	0.305	5.716×10^{-2}
R-ID TP	0.658	0.402	0.426	0.516	0.398
F-ID TP	3.028×10^{-4}*	6.065×10^{-5}*	2.712×10^{-4}*	4.414×10^{-5}*	1.293×10^{-5}*
F-ID AP	9.083×10^{-8}*	1.205×10^{-8}*	7.244×10^{-8}*	5.468×10^{-8}*	2.421×10^{-8}*
F-ID TP G1	1.952×10^{-8}*	5.550×10^{-9}*	3.212×10^{-8}*	8.400×10^{-9}*	4.480×10^{-9}*

8×8 Lawn Mower Wilcoxon rank-sun test results continued.

F-ID AP G1	1.220×10^{-9}*	9.300×10^{-10}*	1.220×10^{-9}*	8.500×10^{-10}*	1.140×10^{-9}*
I-ID AP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID TP n-10	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP n-50	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP G1 n-25	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*
I-ID TP n-25	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP G1 n-50	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
I-ID TP G1 n-10	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
GE	7.600×10^{-10}*	7.500×10^{-10}*	7.600×10^{-10}*	7.500×10^{-10}*	7.600×10^{-10}*

	M-ID TP n-25	M-ID AP G1 n-25	R-ID TP G1	R-ID AP G1	R-ID TP
ADF	6.813×10^{-6}*	2.717×10^{-7}*	5.636×10^{-6}*	8.834×10^{-7}*	1.929×10^{-5}*
M-ID AP n-10	0.370	0.201	0.608	0.165	0.389
M-ID TP G1 n-50	0.237	0.302	0.416	8.360×10^{-2}	7.589×10^{-2}
M-ID TP G1 n-25	2.356×10^{-2}*	3.002×10^{-2}*	0.184	2.036×10^{-2}*	5.750×10^{-2}
M-ID AP G1 n-10	0.221	0.174	0.665	0.114	0.346
R-ID AP	0.964	0.581	0.665	0.659	0.658
M-ID TP n-10	0.414	0.647	0.861	0.303	0.402
M-ID AP n-25	0.406	0.514	0.855	6.054×10^{-2}	0.426
M-ID TP G1 n-10	0.974	0.942	0.880	0.305	0.516
M-ID TP n-50	0.305	0.305	0.768	5.716×10^{-2}	0.398
M-ID TP n-25		0.874	0.890	0.590	0.757
M-ID AP G1 n-25	0.874		0.958	0.495	0.933
R-ID TP G1	0.890	0.958		9.717×10^{-2}	0.668
R-ID AP G1	0.590	0.495	9.717×10^{-2}		0.737
R-ID TP	0.757	0.933	0.668	0.737	
F-ID TP	9.708×10^{-6}*	2.167×10^{-4}*	4.797×10^{-4}*	3.328×10^{-3}*	3.237×10^{-4}*
F-ID AP	1.466×10^{-8}*	2.846×10^{-8}*	5.983×10^{-8}*	3.291×10^{-6}*	1.816×10^{-7}*
F-ID TP G1	9.370×10^{-9}*	2.164×10^{-8}*	9.350×10^{-8}*	3.336×10^{-7}*	3.620×10^{-8}*
F-ID AP G1	1.470×10^{-9}*	1.420×10^{-9}*	1.560×10^{-9}*	1.190×10^{-9}*	1.710×10^{-9}*
I-ID AP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-25	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
I-ID TP n-10	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP G1 n-25	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
I-ID TP n-50	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*
I-ID TP G1 n-25	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP n-25	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
I-ID TP G1 n-50	7.600×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
I-ID TP G1 n-10	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
GE	7.400×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.600×10^{-10}*	7.500×10^{-10}*

	F-ID TP	F-ID AP	F-ID TP G1	F-ID AP G1	I-ID AP n-10
ADF	2.569×10^{-7}*	4.000×10^{-9}*	1.410×10^{-9}*	1.140×10^{-9}*	7.600×10^{-10}*
M-ID AP n-10	3.599×10^{-5}*	1.140×10^{-8}*	9.520×10^{-9}*	1.300×10^{-9}*	7.800×10^{-10}*
M-ID TP G1 n-50	1.094×10^{-5}*	3.280×10^{-9}*	6.210×10^{-9}*	8.500×10^{-10}*	7.800×10^{-10}*
M-ID TP G1 n-25	1.285×10^{-5}*	4.910×10^{-9}*	4.700×10^{-9}*	1.220×10^{-9}*	7.700×10^{-10}*
M-ID AP G1 n-10	5.107×10^{-6}*	1.252×10^{-8}*	1.741×10^{-8}*	1.300×10^{-9}*	7.800×10^{-10}*
R-ID AP	3.028×10^{-4}*	9.083×10^{-8}*	1.952×10^{-8}*	1.220×10^{-9}*	7.800×10^{-10}*
M-ID TP n-10	6.065×10^{-5}*	1.205×10^{-8}*	5.550×10^{-9}*	9.300×10^{-10}*	7.800×10^{-10}*
M-ID AP n-25	2.712×10^{-4}*	7.244×10^{-8}*	3.212×10^{-8}*	1.220×10^{-9}*	7.800×10^{-10}*
M-ID TP G1 n-10	4.414×10^{-5}*	5.468×10^{-8}*	8.400×10^{-9}*	8.500×10^{-10}*	7.800×10^{-10}*
M-ID TP n-50	1.293×10^{-5}*	2.421×10^{-8}*	4.480×10^{-9}*	1.140×10^{-9}*	7.800×10^{-10}*

8×8 Lawn Mower Wilcoxon rank-sun test results continued.

M-ID TP n-25	$9.708 \times 10^{-6}*$	$1.466 \times 10^{-8}*$	$9.370 \times 10^{-9}*$	$1.470 \times 10^{-9}*$	$7.800 \times 10^{-10}*$
M-ID AP G1 n-25	$2.167 \times 10^{-4}*$	$2.846 \times 10^{-8}*$	$2.164 \times 10^{-8}*$	$1.420 \times 10^{-9}*$	$7.800 \times 10^{-10}*$
R-ID TP G1	$4.797 \times 10^{-4}*$	$5.983 \times 10^{-8}*$	$9.350 \times 10^{-8}*$	$1.560 \times 10^{-9}*$	$7.800 \times 10^{-10}*$
R-ID AP G1	$3.328 \times 10^{-3}*$	$3.291 \times 10^{-6}*$	$3.336 \times 10^{-7}*$	$1.190 \times 10^{-9}*$	$7.800 \times 10^{-10}*$
R-ID TP	$3.237 \times 10^{-4}*$	$1.816 \times 10^{-7}*$	$3.620 \times 10^{-8}*$	$1.710 \times 10^{-9}*$	$7.800 \times 10^{-10}*$
F-ID TP		$2.221 \times 10^{-3}*$	$2.221 \times 10^{-3}*$	$2.844 \times 10^{-7}*$	$7.800 \times 10^{-10}*$
F-ID AP	$2.221 \times 10^{-3}*$		0.235	$1.477 \times 10^{-4}*$	$7.800 \times 10^{-10}*$
F-ID TP G1	$2.221 \times 10^{-3}*$	0.235		$1.802 \times 10^{-4}*$	$7.800 \times 10^{-10}*$
F-ID AP G1	$2.844 \times 10^{-7}*$	$1.477 \times 10^{-4}*$	$1.802 \times 10^{-4}*$		$7.800 \times 10^{-10}*$
I-ID AP n-10	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	
I-ID AP G1 n-10	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	0.778
I-ID AP n-25	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	0.487
I-ID TP n-10	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	0.434
I-ID AP G1 n-25	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	0.191
I-ID TP n-50	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	7.045×10^{-2}
I-ID TP G1 n-25	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	5.165×10^{-2}
I-ID TP n-25	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$3.452 \times 10^{-2}*$
I-ID TP G1 n-50	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$2.172 \times 10^{-2}*$
I-ID TP G1 n-10	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$1.613 \times 10^{-2}*$
GE	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$1.664 \times 10^{-3}*$

	I-ID AP G1 n-10	I-ID AP n-25	I-ID TP n-10	I-ID AP G1 n-25	I-ID TP n-50
ADF	$7.600 \times 10^{-10}*$	$7.600 \times 10^{-10}*$	$7.600 \times 10^{-10}*$	$7.400 \times 10^{-10}*$	$7.400 \times 10^{-10}*$
M-ID AP n-10	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$
M-ID TP G1 n-50	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.700 \times 10^{-10}*$
M-ID TP G1 n-25	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.700 \times 10^{-10}*$
M-ID AP G1 n-10	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$
R-ID AP	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$
M-ID TP n-10	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$
M-ID AP n-25	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$
M-ID TP G1 n-10	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.700 \times 10^{-10}*$
M-ID TP n-50	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.700 \times 10^{-10}*$
M-ID TP n-25	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$
M-ID AP G1 n-25	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$
R-ID TP G1	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.700 \times 10^{-10}*$
R-ID AP G1	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$
R-ID TP	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$
F-ID TP	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$
F-ID AP	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$
F-ID TP G1	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$
F-ID AP G1	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.800 \times 10^{-10}*$
I-ID AP n-10	0.778	0.487	0.434	0.191	7.045×10^{-2}
I-ID AP G1 n-10		0.849	0.659	0.543	0.240
I-ID AP n-25	0.849		0.891	0.470	0.518
I-ID TP n-10	0.659	0.891		0.750	0.616
I-ID AP G1 n-25	0.543	0.470	0.750		0.887
I-ID TP n-50	0.240	0.518	0.616	0.887	
I-ID TP G1 n-25	0.177	0.332	0.320	0.451	1.000
I-ID TP n-25	0.172	0.312	0.224	0.263	0.433
I-ID TP G1 n-50	9.959×10^{-2}	0.312	0.173	0.664	0.567
I-ID TP G1 n-10	0.110	0.109	0.279	0.363	0.521
GE	$1.058 \times 10^{-2}*$	$4.966 \times 10^{-2}*$	$1.205 \times 10^{-2}*$	9.848×10^{-2}	7.733×10^{-2}

	I-ID TP G1 n-25	I-ID TP n-25	I-ID TP G1 n-50	I-ID TP G1 n-10	GE
ADF	$7.000 \times 10^{-10}*$	$7.500 \times 10^{-10}*$	$6.900 \times 10^{-10}*$	$7.300 \times 10^{-10}*$	$6.100 \times 10^{-10}*$
M-ID AP n-10	$7.700 \times 10^{-10}*$	$7.800 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.700 \times 10^{-10}*$	$7.700 \times 10^{-10}*$

8×8 Lawn Mower Wilcoxon rank-sun test results continued.

M-ID TP G1 n-50	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.500×10^{-10}*
M-ID TP G1 n-25	7.600×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.500×10^{-10}*
M-ID AP G1 n-10	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
R-ID AP	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
M-ID TP n-10	7.600×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.500×10^{-10}*
M-ID AP n-25	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
M-ID TP G1 n-10	7.600×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.500×10^{-10}*
M-ID TP n-50	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
M-ID TP n-25	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*	7.700×10^{-10}*	7.400×10^{-10}*
M-ID AP G1 n-25	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*
R-ID TP G1	7.600×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
R-ID AP G1	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.600×10^{-10}*
R-ID TP	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.700×10^{-10}*	7.500×10^{-10}*
F-ID TP	7.800×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*	7.800×10^{-10}*	7.700×10^{-10}*
F-ID AP	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID TP G1	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
F-ID AP G1	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*	7.800×10^{-10}*
I-ID AP n-10	5.165×10^{-2}	3.452×10^{-2}*	2.172×10^{-2}*	1.613×10^{-2}*	1.664×10^{-3}*
I-ID AP G1 n-10	0.177	0.172	9.959×10^{-2}	0.110	1.058×10^{-2}*
I-ID AP n-25	0.332	0.312	0.312	0.109	4.966×10^{-2}*
I-ID TP n-10	0.320	0.224	0.173	0.279	1.205×10^{-2}*
I-ID AP G1 n-25	0.451	0.263	0.664	0.363	9.848×10^{-2}
I-ID TP n-50	1.000	0.433	0.567	0.521	7.733×10^{-2}
I-ID TP G1 n-25		0.611	0.689	0.533	0.213
I-ID TP n-25	0.611		0.627	0.861	0.428
I-ID TP G1 n-50	0.689	0.627		0.890	0.216
I-ID TP G1 n-10	0.533	0.861	0.890		0.357
GE	0.213	0.428	0.216	0.357	

Appendix B

An Analysis of Modules (Chapter 7)

B.1 Module Content

B.1.1 Module Semantics

Table B.1: This table shows the average semantic value and standard error of all the modules ever discovered on the Santa Fe Ant Trail problem. The semantic value is calculated by evaluating each module on the fitness function as if it was a stand-alone individual.

Approach	Semantic Value	Std. Err.
F-ID TP G1	68.118	1.237
F-ID AP G1	68.868	1.160
R-ID TP G1	68.713	0.993
I-ID TP n-25	79.710	1.087
I-ID AP G1 n-10	75.769	0.608
I-ID AP G1 n-25	79.405	0.415
I-ID TP n-50	79.912	1.120
F-ID AP	70.867	0.875
ADF	83.707	0.011
R-ID AP	67.217	0.691
M-ID TP G1 n-50	68.557	1.142
M-ID TP G1 n-25	66.812	1.138
M-ID TP G1 n-10	68.017	1.099
M-ID AP G1 n-10	70.474	0.551
I-ID AP n-10	77.130	0.574
M-ID AP n-10	67.992	0.642
I-ID AP n-25	78.025	0.561
R-ID AP G1	72.116	0.500
I-ID TP G1 n-25	78.874	1.131
I-ID TP G1 n-10	77.760	1.243
M-ID TP n-25	67.527	1.048
M-ID TP n-10	67.883	1.103
I-ID TP G1 n-50	80.491	1.122
F-ID TP	68.595	0.965
I-ID TP n-10	76.761	1.199
M-ID AP n-25	69.348	0.650
M-ID TP n-50	68.297	1.127
R-ID TP	69.939	0.977
M-ID AP G1 n-25	72.744	0.572

Table B.2: This table shows the average semantic value and standard error of all the modules ever discovered on the Even 7 Parity problem. The semantic value is calculated by evaluating each module on the fitness function as if it was a stand-alone individual.

Approach	Semantic Value	Std. Err.
F-ID AP G1	62.428	0.292
I-ID TP G1 n-50		
F-ID TP G1	61.884	0.334
R-ID AP G1	56.633	0.287
M-ID TP G1 n-10	51.704	0.582
M-ID TP G1 n-25	52.794	0.573
I-ID TP G1 n-25		
M-ID TP G1 n-50	52.559	0.595
I-ID AP G1 n-10	58.143	2.819
R-ID AP	55.697	0.307
ADF	63.601	0.005
R-ID TP G1	56.625	0.364
M-ID AP G1 n-10	52.556	0.435
I-ID TP n-10	54.400	0.584
F-ID TP	61.112	0.379
M-ID AP G1 n-25	52.071	0.461
F-ID AP	61.626	0.361
R-ID TP	58.123	0.314
I-ID AP G1 n-25		
I-ID TP G1 n-10	42.783	0.569
M-ID TP n-10	52.745	0.551
M-ID TP n-25	52.165	0.568
I-ID TP n-50		
I-ID TP n-25	0	0
M-ID TP n-50	54.473	0.553
I-ID AP n-10	64.000	0
M-ID AP n-10	50.564	0.503
M-ID AP n-25	50.358	0.540
I-ID AP n-25		

Table B.3: This table shows the average semantic value and standard error of all the modules ever discovered on the $x^5 - 2x^3 + x$ Symbolic Regression problem. The semantic value is calculated by evaluating each module on the fitness function as if it was a stand-alone individual.

Approach	Semantic Value	Std. Err.
R-ID AP	5.0776×10^{10}	3.576×10^{10}
R-ID TP G1	3.297×10^{5}	2.275×10^{5}
F-ID AP	5.386×10^{3}	2.822×10^{3}
F-ID TP	1.817×10^{5}	1.267×10^{5}
I-ID TP n-25	1.115×10^{2}	16.159
F-ID TP G1	1.090×10^{6}	1.082×10^{6}
I-ID TP n-10	2.728×10^{2}	140.135
I-ID TP G1 n-50	9.351×10^{1}	16.470
F-ID AP G1	7.039×10^{3}	2.521×10^{3}
I-ID AP n-10	1.047×10^{2}	7.146
R-ID TP	4.617×10^{6}	1.894×10^{6}
R-ID AP G1	5.044×10^{9}	4.976×10^{9}
I-ID TP n-50	9.747×10^{1}	19.949
M-ID TP G1 n-10	2.907×10^{13}	1.677×10^{13}
ADF	1.085×10^{37}	9.427×10^{36}
M-ID TP G1 n-25	4.537×10^{3}	1.559×10^{3}
I-ID TP G1 n-25	1.027×10^{2}	13.709
I-ID AP G1 n-25	8.471×10^{1}	6.629
M-ID AP n-10	1.425×10^{3}	628.146
I-ID AP n-25	9.635×10^{1}	9.567
M-ID AP n-25	5.108×10^{4}	4.855×10^{4}
M-ID TP G1 n-50	3.826×10^{5}	3.823×10^{5}
M-ID TP n-25	2.114×10^{4}	1.320×10^{4}
I-ID TP G1 n-10	1.077×10^{2}	7.519
I-ID AP G1 n-10	1.509×10^{3}	1.208×10^{3}
M-ID TP n-50	4.572×10^{8}	3.231×10^{8}
M-ID TP n-10	2.789×10^{3}	1.180×10^{3}
M-ID AP G1 n-10	2.415×10^{6}	1.922×10^{6}
M-ID AP G1 n-25	2.584×10^{10}	2.400×10^{10}

Table B.4: This table shows the average semantic value and standard error of all the modules ever discovered on the 8×8 Lawn Mower problem. The semantic value is calculated by evaluating each module on the fitness function as if it was a stand-alone individual.

Approach	Semantic Value	Std. Err.
ADF	57.905	0.012
M-ID AP n-10	12.505	0.189
M-ID TP G1 n-50	13.833	0.197
M-ID TP G1 n-25	12.854	0.199
M-ID AP G1 n-10	16.012	0.247
R-ID AP	13.381	0.168
M-ID TP n-10	12.872	0.187
M-ID AP n-25	16.269	0.213
M-ID TP G1 n-10	13.042	0.187
M-ID TP n-50	13.621	0.199
M-ID TP n-25	13.132	0.195
M-ID AP G1 n-25	25.482	0.303
R-ID TP G1	10.830	0.180
R-ID AP G1	18.007	0.205
R-ID TP	11.047	0.179
F-ID TP	30.284	0.198
F-ID AP	37.113	0.188
F-ID TP G1	40.916	0.219
F-ID AP G1	43.405	0.187
I-ID AP n-10	61.230	0.146
I-ID AP G1 n-10	61.553	0.148
I-ID AP n-25	61.466	0.202
I-ID TP n-10	60.694	0.331
I-ID AP G1 n-25	61.285	0.224
I-ID TP n-50	61.740	0.236
I-ID TP G1 n-25	61.810	0.219
I-ID TP n-25	61.362	0.334
I-ID TP G1 n-50	62.093	0.266
I-ID TP G1 n-10	61.822	0.293

List of Abbreviations

Γ	An integer array representing the chromosome of an individual
Γ_i	The i^{th} codon in a chromosome, Γ
κ	The amount of modules that are kept after the module replacement operation is performed
λ	A candidate module
μ	A module that has passed its evaluations and has been accepted into the module list
ρ	An double value specifying how many evaluations a candidate module must pass in order to be accepted into the module list. This value is defined as a percentage of n and rounded to the nearest integer value
Σ	A finite set of terminal productions
σ	A sub-derivation tree
τ	The number of generations between module identification, module replacement, and grammar modification occurrences
G	A context-free grammar. $G = (V, \Sigma, R, S)$
I	An individual in the evolving population
M	A set of all modules that have passed the necessary evaluations and may be made available to the population
n	How many evaluations each candidate module undergoes
R	A finite relation from V to $(V \cup \Sigma)^*$.
S	The start symbol used to represent the entire program. S must be a non-terminal symbol ($S \in V$)
V	A finite set non-terminal production

MIG Module Identification Gap: A gap data plotted in relation to fitness evaluation where some amount of fitness evaluations are spent identifying modules instead of evolving the population