

國立臺灣大學電機資訊學院電機工程學系



碩士論文

Department of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

針對符號回歸之值域與綁定基因編程演算法

Ranging-Binding Genetic Programming
for Symbolic Regression

方文忠

Wen-Zhong Fang

指導教授: 于天立 博士

Advisor: Tian-Li Yu, Ph.D.

中華民國 112 年 7 月

July, 2023



Acknowledgements

碩士的兩年將以這份作品畫下句點，這是一段混雜尋找方向的迷茫、得到不同觀點的新奇、實驗失敗的苦惱、結果符合預期的喜悅與完成整份論文的成就感的旅程。在過程中慢慢從聽不懂學長姊與老師在討論甚麼到能用專有名詞精確的表達自己觀點與大家討論，從盲目的翻閱別人的論文到知道甚麼順序、甚麼重點去吸收別人想表達的觀點。感謝于天立老師與實驗室的大家能讓我經歷這些成長。而本篇論文之所以能完成，首先要感謝于天立老師的指導，除了研究內容細心的指導，研究的方法與態度也令我受益良多。再來是實驗室的大家，尤其是同屆一起努力的勗辰、柏維與容均，在與我討論研究內容之餘，也常陪我聊天緩解壓力，麒仙學姊、庭碩學長常常幫助我釐清研究方向與分享技巧與工具，梓豪、讀晉幫忙我修正錯誤以及練習口試，還有上一屆的靖中學長、智凱學長、奕昀學姊以及其他學長姊的經驗與幫助。最後，感謝我的家人以及一切的機運，讓我能完整的完成這份作品。





摘要

基因編程演算法是一種利用群體中程式個體重組、突變、選擇操作，進行演化以達到生成完成目標的程式的演算法。本論文提出了一種針對符號回歸問題的基因編程演算法，以利用程式語法資訊和語義資訊讓演化更有效率。提出的演算法由兩個機制組成，綁定機制與值域機制。綁定機制是針對語法資訊所設計，藉由保護族群中常見的兩層結構函式來避免在重組操作中重要的結構被破壞。值域機制是針對語義資訊所設計，藉由當前程式的輸出範圍與目標的輸出範圍之大小差異，來選擇重組中的子樹對象，用以保留父代的優異性。此兩種機制在實驗中顯示在實際應用資料集中具有優於其他當代方法的最佳化能力。此外，此演算法在綁定機制中存在一個待保護的常見函式的個數，是一個需調整的參數，因此本論文使用了適應機制來自動調整此參數。最後，前期實驗顯示此演算法在高維度中相較其他當代方法會有較不穩定的表現，為此本論文提出了基於最小冗餘最大相關特徵選擇演算法進行降維，得到更穩定的表現。

關鍵字：基因編程演算法、符號回歸





Abstract

Genetic programming is an evolutionary algorithm that utilizes recombination, mutation, and selection operations in a population to evolve programs. This thesis presents a genetic programming algorithm specifically designed for symbolic regression problems, aiming to utilize both syntax and semantic information for more efficient evolution. The algorithm consists of two mechanisms: the binding mechanism and the ranging mechanism. The binding mechanism protects frequently occurring two-layer function structures during recombination to preserve their importance. The ranging mechanism adjusts the selection of subtrees for recombination based on the difference between the output range of the current program and the target output range, aiming to retain the superiority of the parent programs. Empirical results demonstrate that ranging-binding genetic programming outperforms other contemporary methods in terms of the mean absolute error on Penn machine learning benchmarks. However, two issues are identified: the need to adjust the number of protected functions as a parameter and the algorithm's instability in

high-dimensional problems. To automatically adjust the number of protected functions in the binding mechanism, an adaptive mechanism is proposed to automatically adjust this parameter. Furthermore, to stabilize the performance in high-dimensional problems, a feature selection based on minimum redundancy maximum relevance is proposed for dimensionality reduction, resulting in more stable performance.

Keywords: Genetic programming, Symbolic regression



Contents

	Page
Acknowledgements	i
摘要	iii
Abstract	v
Contents	vii
List of Figures	xi
List of Tables	xiii
Chapter 1 Introduction	1
Chapter 2 Simple Genetic Programming and Its SOTA Variants	5
2.1 Simple Genetic Programming Framework	5
2.1.1 Automatically Defined Function	8
2.2 Gplearn	8
2.3 GP-GOMEA	9
2.3.1 The Representation in GP-GOMEA	9
2.3.2 Linkage Learning and Random Tree	10
2.3.3 Gene-pool Optimal Mixing	11
2.4 EllynGP	11
2.4.1 Epigenetic Information	12



2.4.2	Epigenetic Hill Climbing	13
2.5	AFP	14
2.6	ELPEX	15
Chapter 3	Ranging-Binding Genetic Programming	17
3.1	Framework	18
3.2	Binding Mechanism	21
3.2.1	Binding	22
3.2.2	Binding Adaption	24
3.3	Ranging Mechanism	26
3.4	Feature Selection	30
Chapter 4	Test Problems	35
4.1	Problems with Ground-truth Mathematical Expression	36
4.1.1	Shear Flow Problems	36
4.1.2	Predator-prey Problems	37
4.1.3	Lotka-Volterra Model of Competition	37
4.1.4	Glider Problems	38
4.1.5	Bar Magnets Problems	38
4.1.6	Bacterial Respiration Problems	39
4.2	Problems with Real-world Data Points	39
Chapter 5	Experiments and Discussions	43
5.1	Preliminary Experiment Results	43
5.1.1	Binding	44

5.1.2	Ranging	49
5.2	Experiment Results on Benchmarks	52
5.2.1	RBGP	52
5.2.2	RBGP with Binding Adaptation	55
5.2.3	RBGP with Binding Adaptation and Feature Selection	58
5.3	RBGP with Other Mechanisms	61
5.3.1	Constant	61
5.3.2	Operon Local Search	62
Chapter 6	Conclusion	67
	References	69
	Appendix A — Introduction to MRMR and Operon	77
A.1	Maximum Relevance and Minimum Redundancy Feature Selection	77
A.2	Operon	78







List of Figures

2.1	The flowchart of SGP framework	7
2.2	Example of the representation in GP-GOMEA	10
2.3	Example of GOM in GP-GOMEA	11
2.4	Example of epigenome in ellynGP	13
2.5	EllynGP and its variants	15
3.1	The framework of RBGP algorithm	20
3.2	An example of the binding mechanism	24
3.3	An example of the binding adaptation	26
3.4	An example of the ranging crossover	29
3.5	An example of proposed feature selection	32
4.1	Bar magnets problem	39
5.1	The impact of including ADF in the function set	47
5.2	The comparison results between SGP + Binding and SGP on $x^3 + x^2 + x$	47
5.3	The relationship between average fitness and the number of bindings	48
5.4	The evolutionary structures of RGP and gplearn	51
5.5	The ranking analysis of the four methods	55
5.6	The variation of b_t as generations increase	57
5.7	Statistical analysis after incorporating feature selection	59
5.8	Lowest MAE ratio after incorporating feature selection	60
5.9	The computation time after incorporating feature selection	60
5.10	Statistical analysis after incorporating constant	62





List of Tables

4.1	UCI test problems	40
4.2	PMLB test problems	41
5.1	The experiment settings for the binding preliminary experiments	46
5.2	The experiment settings for the ranging preliminary experiments	50
5.3	The results of ranging preliminary experiments	51
5.4	The experiment settings for RBGP experiments	53
5.5	The comparison of first version RBGP	54
5.6	RBGP with binding adaptation compared with other methods	56
5.7	The average dimension after feature selection	58
5.8	Raw fitness of RBGP compared with other SOTAs on UCI datasets	61
5.11	RBGP with one-layer operon local search compared with other methods combined with one-layer operon local search	63
5.9	Raw fitness of RBGP- β and RBGP compared with other SOTAs	64
5.10	Raw fitness of RBGP compared with RBGP+constant	65
5.12	Raw fitness of RBGP and SOTAs combined with Operon	66





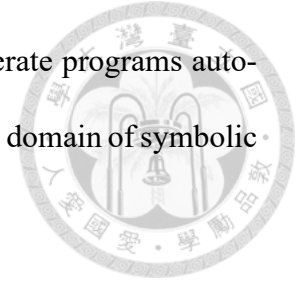
Chapter 1 Introduction

Symbolic regression is a subfield of machine learning that aims to learn a mathematical expression as a model by utilizing input data and target values. Symbolic regression is useful in domains that require mathematical expressions to describe processes, such as physics [31], biology [33], and engineering [15, 34]. In the field of symbolic regression, most researchers consider this problem to be NP-hard [20, 42] and genetic programming (GP) is seemed as a popular approach to solve symbolic regression [16, 17, 36, 44].

Symbolic regression differs from numerical regression in that it seeks to identify the best combination of basic operators, such as $+$, $-$, \times , \div , exp , sin , and terminal variables to fit the target, rather than searching for optimal parameters within a fixed equation. Symbolic regression requires few prior restrictions and expertise compared to numerical regression, where the choice of the equation is crucial and often relies on domain-specific knowledge. Furthermore, symbolic regression differs from black-box machine learning. Symbolic regression provides a mathematical expression that can be analyzed and interpreted.

GP is an evolutionary computation algorithm introduced by Koza [13]. GP shares similarities with genetic algorithms (GAs) as they both follow a population-based black-box optimization framework. However, GP focuses on evolving program encodings rather

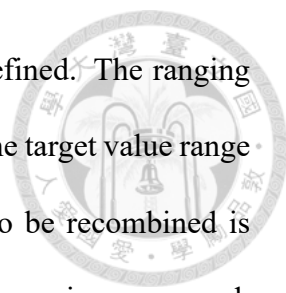
than decision variables. GP is designed to enable computers to generate programs automatically. Since its introduction, GP has found wide application in the domain of symbolic regression [7, 14, 48].



Starting from the 1990s, genetic algorithms have undergone a transformation towards the development of model-building genetic algorithms (MBGAs). Two branches in the field of MBGAs are dependency structure matrix genetic algorithm II [6, 11] and linkage tree genepool optimal mixing evolutionary algorithm (LT-GOMEA) [3, 39, 40], representing the state-of-the-art (SOTA) approaches. The primary objective of these algorithms is to unveil concealed patterns among potential chromosomes through the application of machine learning techniques. Through leveraging these acquired patterns, MBGAs can produce offspring with potentially enhanced quality during the recombination process. We believe that genetic programming can also gain advantages from adopting a comparable approach.

The objective of this thesis is to design a genetic programming approach tailored for symbolic regression, known as the ranging-binding genetic programming algorithm (RBGP). RBGP incorporates the principles of model building and symbolic regression genetic programming (SRGP) to tackle the challenge of offspring not preserving the superior traits of their parents throughout the SRGP evolution process. To preserve the parents' superiority as much as possible, RBGP makes use of syntax and semantics information of the program. In previous GP research, semantic represents the output of a program, and based on semantics, variants of crossover operations are proposed. [24, 25].

For semantics, this thesis introduces the ranging mechanism, which utilizes information from the numerical range of the output. As the output of symbolic regression reflects



the semantics of a program, the numerical range can be precisely defined. The ranging mechanism promotes offspring to adjust their output range to match the target value range during recombination. For instance, if the output range of a tree to be recombined is broader than the target range, recombination that narrows its output range is encouraged. Otherwise, recombination that maintains the wider output range is discouraged.

For syntax, this thesis introduces the binding mechanism, which is similar to the concept of automatically defined functions (ADF) [26]. The binding mechanism identifies frequently occurring structures in the population and ensures that they remain together during the recombination process. This statistical approach is similar to estimation of distribution genetic programming (EDA-GP) [10]. However, unlike ADF and EDA-GP, the binding operators are not added to the function set, thus not affecting the generation probability of new trees in GP. Empirical evidence suggests that ADF utilization is generally low because a commonly used function is effective only in specific contexts [29]. For example, when evolving $y = x^{10}$ using the operators $+$, $-$, \times , \div , it is clear that combining multiple multiplications is necessary. However, if three consecutive multiplications are defined as an ADF, it greatly increases the probability of generating $y = x^3$, $y = x^6$, or $y = x^9$, which may not be suitable for the problem.

In this thesis, binding adaptation is proposed to automatically adjust the binding rate. The binding rate is a parameter that determines a threshold in the binding mechanism. Only the function structures that appear more frequently than this threshold in the population are considered significant and protected by binding. However, using different binding rates have a significant impact on performance, and different binding rates are also needed in the early and later stages of evolution. In order to address these challenges, we introduce binding adaptation, which automatically adjusts the binding rate. Further-

more, preliminary experiments indicate that the algorithm demonstrates lower stability in high-dimensional problems compared to other contemporary methods. To address this issue, this thesis proposes a feature selection algorithm based on minimum redundancy maximum relevance (MRMR) [47] for dimensionality reduction.

The proposed algorithm demonstrates a more robust optimization capability on 37 problems out of the Penn machine learning benchmarks (PMLB) [28] on average than GP-GOMEA [44], ellynGP [16], ellynGP with epsilon-lexicase selection (EPLEX) [19], and ellynGP with age-fitness Pareto (AFP) [32], and gplearn [36].



Chapter 2 Simple Genetic Programming and Its SOTA Variants

This chapter provides readers with background information on the simple genetic programming (SGP) framework, and state-of-the-art (SOTA) GP algorithms, including GP-GOMEA, ellynGP, EPLEX, AFP, and gplearn. These SOTAs are based on the SGP framework and are used as comparisons in this thesis.

2.1 Simple Genetic Programming Framework

SGP [13] utilizes computer programs as representations for problem-solving. The programs are composed of functions and terminals, and their size and shape are not predefined. SGP explores the space of possible program compositions to find solutions. SGP's symbolic expressions are similar to parse trees in compilers, providing a convenient approach to create and manipulate these compositions.

The SGP paradigm involves evolving computer programs to solve problems using a three-step process. Firstly, an initial population of random compositions of functions

and terminals, representing computer programs, is generated. Secondly, the process iteratively proceeds until a termination criterion is met. Each program in the population is executed, and its fitness value is assigned based on its effectiveness in solving the problem.

A new population of computer programs is then created using three operations, with the selection probability based on fitness. The first operation is reproduction, where existing programs are copied into the new population. The second operation is crossover, where parts of two randomly selected programs are combined to create two new programs. The third operation is subtree mutation, which involves randomly selecting a subtree within an individual's program and replacing it with a newly generated subtree. Finally, the best computer program in the population at the termination point is designated as the result, potentially providing a solution or an approximate solution to the problem. Figure 2.1 presents the flowchart of the SGP framework. Algorithm 1 presents an implementation way of the SGP framework.

Algorithm 1: Framework of SGP

Input: P : population;
Output: best program;

```

1 Randomly initialize population  $P$ ;
2 while  $\neg$  ShouldTerminate do
3   for  $i \leftarrow 1$  to  $|P|$  do
4     if Choose to use crossover then
5        $P_{donor} \leftarrow$  RandomSelection( $P$ );
6        $P_i \leftarrow$  Crossover( $P_{donor}, P_i$ );
7     end
8     if Choose to use subtree mutation then
9        $P_i \leftarrow$  SubtreeMutation( $P_i$ );
10    end
11    if Choose to use reproduction then
12       $P_i \leftarrow$  Reproduction( $P_i$ );
13    end
14  end
15 end
16 return Best program in  $P$  ;

```

The basic genetic operations in SGP are fitness proportionate. Reproduction involves

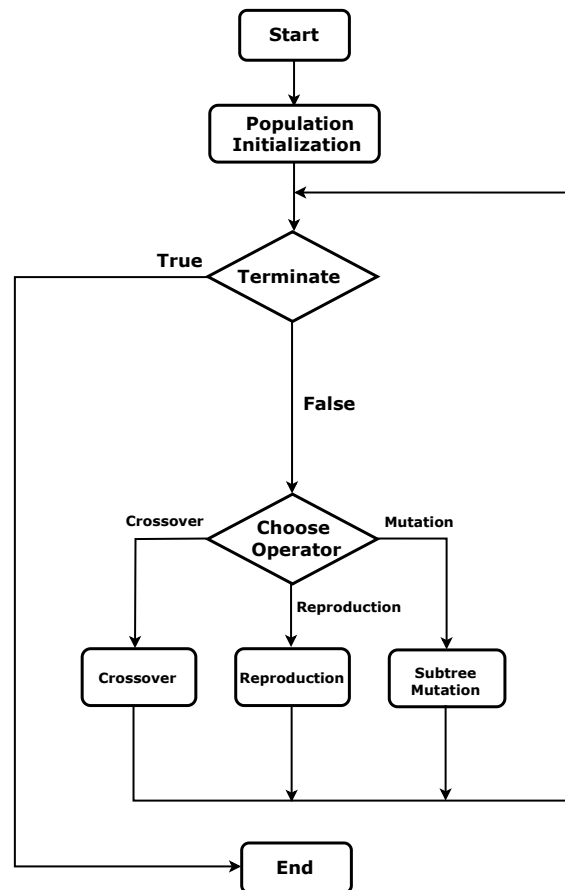
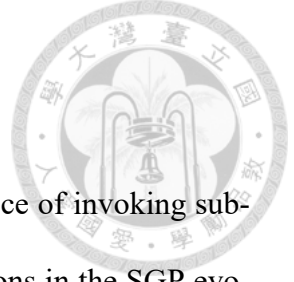


Figure 2.1: The flowchart of SGP framework.

copying individuals into the next generation based on their fitness, allowing the fittest individuals to survive and reproduce. Crossover is a sexual operation that exchanges sub-trees between two parent programs, producing offspring that are syntactically and semantically valid. Subtree mutation is a genetic operator in evolutionary algorithms that introduces variation in the population by randomly selecting a subtree within an individual's program and replacing it with a newly generated subtree. This operation helps explore new areas of the search space and can lead to the discovery of improved solutions. Overall, SGP offers a powerful framework for evolving computer programs and finding solutions to complex problems.



2.1.1 Automatically Defined Function

ADF [26], introduced by Koza in 1994, is inspired by the practice of invoking sub-routines in human programming. It aims to evolve useful sub-functions in the SGP evolutionary process. Previous research has shown that incorporating ADF often leads to improved performance.

2.2 Gplearn

Gplearn [36] is an implementation of GP designed for the Python platform. It utilizes the training framework of scikit-learn and operates within the framework of SGP. One of the challenges in GP is the phenomenon known as bloat, where programs tend to grow larger without any significant improvement in fitness. This can lead to increased computation costs [23, 43]. To address the issue of bloat, gplearn incorporates several mechanisms. One such mechanism is the use of a parsimony coefficient [21] with adaptive control. The parsimony coefficient penalizes larger programs, encouraging the evolution of smaller and more efficient solutions. Additionally, gplearn employs the hoist mutation [12] operator, which restructures the program by moving subtrees to higher levels of the tree, reducing redundancy and improving efficiency. Due to its effective handling of bloat and its compatibility with scikit-learn, gplearn has become a popular choice for comparisons and as a framework in various GP applications [19, 30, 38]. Researchers and practitioners in the field of GP often rely on gplearn to benchmark their algorithms and methodologies.

2.3 GP-GOMEA



Gene-pool optimal mixing evolutionary algorithm (GOMEA) has been applied in various domains and has achieved success in experiments, such as discrete optimization [40], multi-objective optimization [22], and real-valued optimization [4].

GP-GOMEA [44] is a variant of GOMEA, which is a model-based evolutionary algorithm that utilizes linkage model. It is specifically designed for solving symbolic regression problems, with the objective of finding precise and concise solutions. Furthermore, GP-GOMEA proposed the interleaved multistart scheme (IMS) to enhance its performance in solving symbolic regression. The IMS involves executing multiple evolutionary runs with progressively increasing evolutionary budgets, all in an interleaved manner.

2.3.1 The Representation in GP-GOMEA

GP-GOMEA uses a special type of tree structure, different from traditional GP trees. This modified tree structure has a fixed template that allows for linkage learning and variation based on linkages, similar to other fixed-length versions of GOMEA. In GP-GOMEA, solutions are represented as perfect r -ary trees with a specific height h , where each non-leaf node has exactly r children. In this framework, it is possible for terminals to appear in non-leaf nodes or for function nodes to have arity less than r . In such cases, the children of the terminal or the portion exceeding the function node's arity will not be executed and will not affect the output. These nodes are referred to as introns. An example of the representation in GP-GOMEA and the concept of introns can be seen in Figure 2.2.

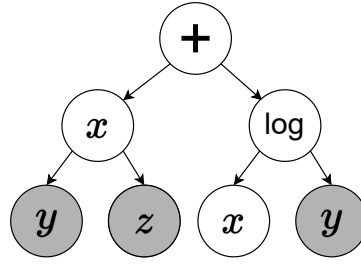


Figure 2.2: Example of the representation in GP-GOMEA. Example with arity 2 and height 3. The gray portion represents the introns.

2.3.2 Linkage Learning and Random Tree

GOMEAs utilize the family of subsets (FOS) as their linkage model. An FOS is a set of sets, denoted as $F = \{F_1, \dots, F_{|F|}\}$, where each F_i is a subset of indices representing specific locations within the genotype. In GP, a specific location in the genotype corresponds to a node in the program tree. In the case of GP-GOMEA, where the shape of the trees is constrained, the nodes represented by an FOS in the population correspond to the same positions. The linkage learning in GOMEA involves learning an FOS, which captures hidden patterns among genotypes. An FOS can take on various forms, and among them, the linkage tree (LT) is recommended in previous research [41].

LT is constructed by following steps. Firstly, the probability distribution over symbols at each location is estimated by computing the statistical occurrence of each symbol in each program. Then calculates the entropy and mutual information based on these probabilities. Secondly, the LT is constructed in a hierarchical manner by treating each genotype location as a cluster and defining the similarity between clusters based on mutual information. Clusters with higher similarity are merged together, and this merging process is repeated until no further merging is possible. The resulting structure is a final LT. In GP-GOMEA, the random tree is employed as FOS. Random tree is a similar framework to LT, it randomly combines two clusters during the merging process.

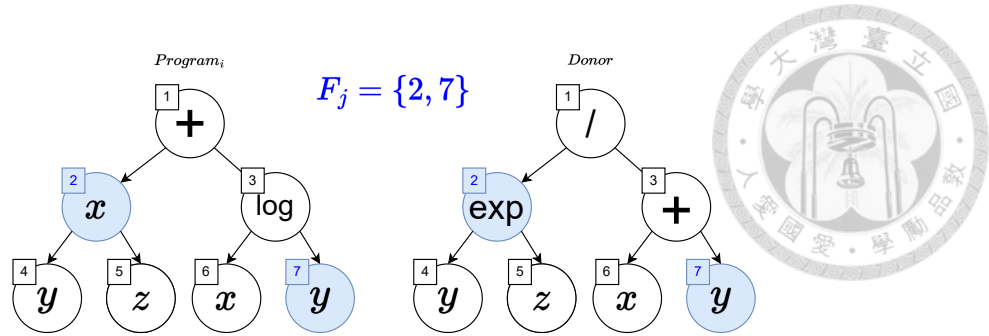


Figure 2.3: Example of GOM in GP-GOMEA. Example of GOM in GP-GOMEA. $Program_i$ and the donor exchange nodes located at positions within F_j .

2.3.3 Gene-pool Optimal Mixing

After constructing an FOS, GOMEA performs GOM on each program P_i within the population to generate the next generation. GOM iterates through each F_j in an FOS and randomly selects another program as a donor. It then exchanges the corresponding F_j between P_i and the donor. If the fitness of the exchanged result is superior to that of P_i , the result replaces P_i . This process is repeated until all subsets in an FOS have been traversed. Once the gene-pool optimal mixing (GOM) is performed on each program within the population, the population in the next generation is obtained. Figure 2.3 illustrates an example of the exchange between $program_i$ and the donor at positions within F_j during the GOM process. Algorithm 2 describes the GP-GOMEA framework.

2.4 EllynGP

The basic framework of ellynGP [16] combines epigenetic information with SGP. Epigenetic information is used to determine the expression or non-expression of each genotype in the phenotype. In GP, this means using epigenetic information to determine which function nodes should be executed. Additionally, ellynGP draws inspiration from the local search methods in genetic algorithms [45] and introduces epigenetic hill climb-



Algorithm 2: GP-GOMEA

Input: P : population;
Output: best program;

```
1 Randomly initialize population  $P$ ;  
2 while  $\neg ShouldTerminate$  do  
3   FOS  $\leftarrow$  LearnRandomTree( $P$ );  
4   for  $i \leftarrow 1$  to  $|P|$  do  
5      $B_i \leftarrow P_i$ ;  
6      $O_i \leftarrow P_i$ ;  
7     for  $F_j \in FOS$  do  
8        $P_{donor} \leftarrow RandomSelection(P)$ ;  
9        $O_i \leftarrow OverrideNodes(O_i, P_{donor}, F_j)$ ;  
10      if  $O_i$  fitness is better than  $B_i$  fitness then  
11         $B_i \leftarrow O_i$ ;  
12      else  
13         $O_i \leftarrow B_i$ ;  
14      end  
15    end  
16     $P_i \leftarrow B_i$ ;  
17  end  
18 end  
19 return best program in  $P$ ;
```

ing to update the epigenetic information. Additionally, ellynGP incorporates two different variants by combining different selection techniques. These variants are introduced in the following sections. Figure 2.5 illustrates the relationship between ellynGP and its variants.

2.4.1 Epigenetic Information

In ellynGP, a program is represented in the genotype using postfix notation. For example, $[x, y, -]$ represents the expression $x - y$. This representation may result in invalid expressions. In such cases, ellynGP considers the valid portion within the postfix notation. For example, $[x, y, -, +, /]$ and $[z, w, x, y, -]$ both represent $x + y$ by considering the valid portion. The former is due to the lack of additional terminal symbols for the operations $+$ and $/$ to perform the computation. The latter is because all

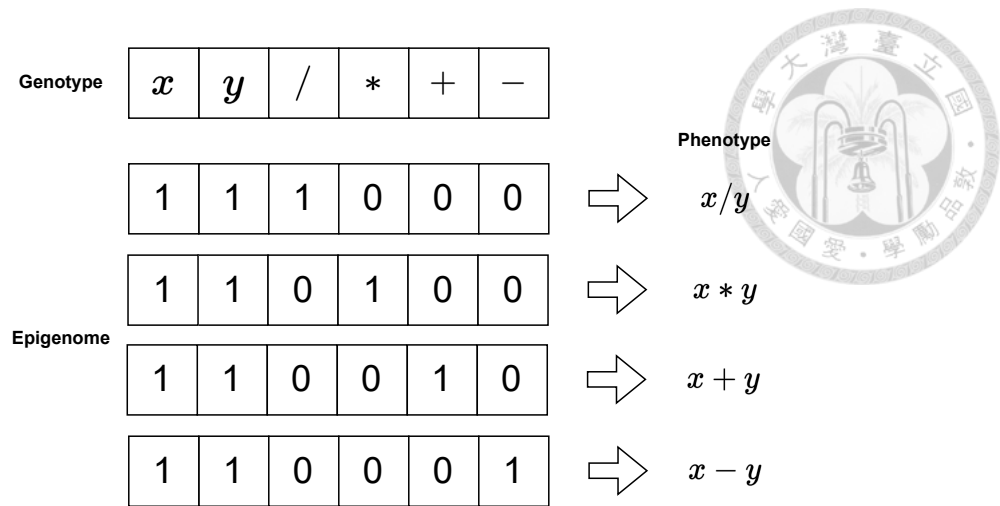


Figure 2.4: Example of epigenome in ellynGP. The epigenome influences the genotype and phenotype correspondence.

the function symbols have already been evaluated, and there is no room to include z and w .

ellynGP stores epigenetic information in the form of a bitstring called the epigenome. Each program has an epigenome, and when a position in the epigenome is 1, the symbol at that position is expressed in the phenotype expression. Conversely, when a position in the epigenome is 0, the symbol at that position is not expressed in the phenotype expression. Figure 2.4 provides an example of an epigenome.

2.4.2 Epigenetic Hill Climbing

In each generation of evolution, ellynGP incorporates epigenetic hill climbing into the basic SGP process to update the epigenome. Epigenetic hill climbing is conceptually inspired by the local search in genetic algorithms. It searches for a bitstring that is close in terms of Hamming distance to the current bitstring and determines if the new string improves the performance. In ellynGP, it attempts to search for an epigenome with a Hamming distance of 1 from the current one and updates the epigenome if it results in better performance. EllynGP introduces epigenetic hill climbing for the search process.

Since searching for all epigenomes with a Hamming distance of 1 is computationally expensive, epigenetic hill climbing iteratively flips each bit of the epigenome. If there is an improvement, the epigenome is updated. The number of iterations for this process in each generation is a parameter and can be repeated multiple times. Algorithm 3 describes the ellynGP framework.

Algorithm 3: EllynGP

Input: P : population; h : number of epigenetic hill climbing iteration;
Output: best program;

```

1 Randomly initialize population  $P$ ;
2 while  $\neg$  ShouldTerminate do
3    $P \leftarrow$  SGPEvolution( $P$ );
4   for  $i \leftarrow 1$  to  $|P|$  do
5      $B_i \leftarrow P_i$ ;
6     for  $n \in \{1, 2, \dots, h\}$  do
7       for  $\ell \leftarrow 1$  to  $\text{length}(P_i.\text{epigenome})$  do
8          $P_i.\text{epigenome}[\ell] \leftarrow$  Flip( $P_i.\text{epigenome}[\ell]$ );
9          $P_i.\text{fitness} \leftarrow$  GetFitness( $P_i, P_i.\text{epigenome}$ );
10        if  $P_i.\text{fitness}$  is better than  $B_i.\text{fitness}$  then
11           $B_i \leftarrow P_i$ ;
12        else
13           $P_i \leftarrow B_i$ ;
14        end
15      end
16    end
17     $P_i \leftarrow B_i$ ;
18  end
19 end
20 return best program in  $P$ ;
```

2.5 AFP

AFP [32] is a selection method used in symbolic regression. EllynGP incorporates AFP into its own framework, resulting in a variant. AFP transforms the original single-objective pursuit of maximizing fitness into a multi-objective approach that aims to maximize fitness while minimizing age. In AFP, age is defined as the time that a program

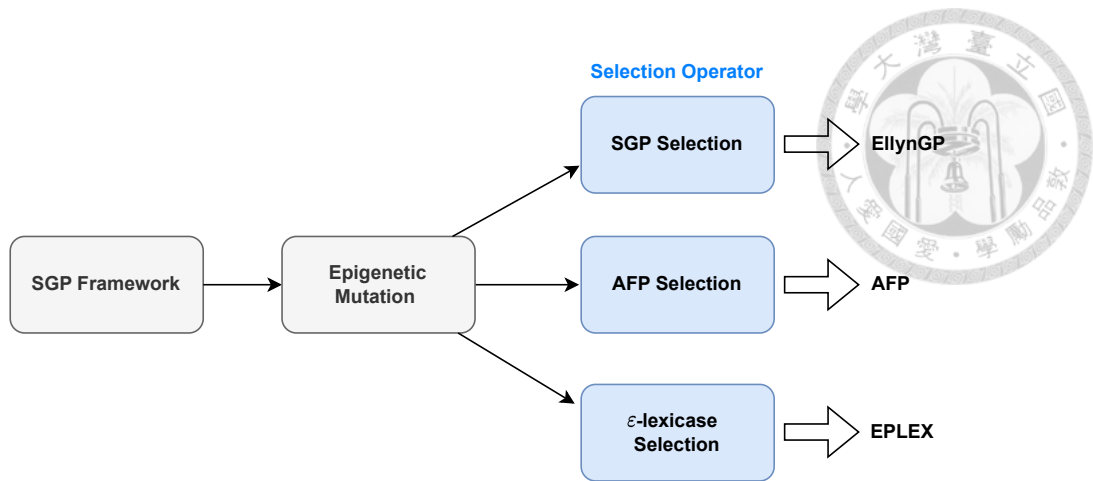


Figure 2.5: EllynGP and its variants.

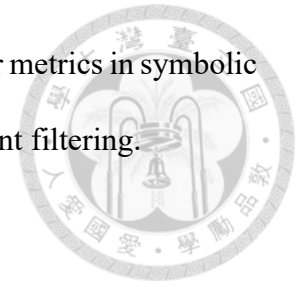
existed in the population, measured in generations. The objective of minimizing age helps prevent premature convergence.

2.6 ELPEX

ELPEX [19] is a variant of ellynGP that combines the ε -lexicase selection technique. Lexicase [9, 35] selection selects programs for the next generation based on the fitness of each individual test case, rather than using a global fitness metric. For example, in symbolic regression problems, instead of using the overall average error as the fitness, lexicase selection considers the error for each sampled data point as a separate fitness criterion. The selection process is repeated multiple times in each generation until the desired number of programs are selected.

The selection method of lexicase selection involves removing individuals from the candidate pool that perform worse than the best-performing individual in the population on each test case. However, this approach may be too stringent for continuous metrics in symbolic regression, resulting in poor performance [19]. ε -lexicase selection builds upon lexicase selection by introducing an ε tolerance for error during the selection process.

This approach helps overcome the challenge of using continuous error metrics in symbolic regression, where strict selection criteria may result in overly stringent filtering.

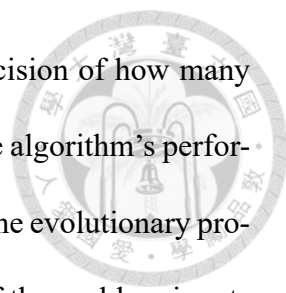




Chapter 3 Ranging-Binding Genetic Programming

This chapter provides a comprehensive introduction to our proposed RBGP for symbolic regression. The objective of RBGP is to effectively utilize the syntax and semantics of program tree structures in symbolic regression. RBGP consists of two components: ranging and binding. Ranging is designed to leverage semantic information by utilizing the difference between the output range of a program and the target range. This information is used to guide the selection of crossover targets, aiming to make the offspring's range closer to the target range. By doing so, it helps to avoid extreme range changes after crossover, ensuring the retention of the superiority of the parent programs. Binding is designed to utilize syntax information by preserving frequently occurring two-layer function node structures from being disrupted by crossover. This concept is similar to ADF [26], but with a difference. Instead of adding the two-layer structures to the function set, binding aims to protect them while avoiding an increase in the population of two-layer structures that can lead to significant output variations.

After incorporating the ranging and binding mechanisms, RBGP has shown improved optimization capabilities compared to state-of-the-art algorithms, as evidenced by its higher average performance on test datasets. However, we have identified two issues and made

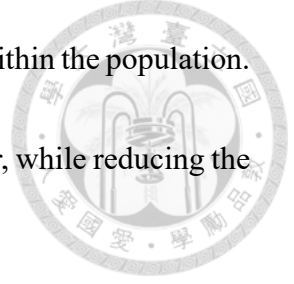


attempts to address them. Firstly, in the binding mechanism, the decision of how many frequently occurring functions to protect is a parameter that affects the algorithm's performance. Furthermore, This parameter may need to be adjusted during the evolutionary process. Secondly, RBGP exhibits less stability as the dimensionality of the problem inputs increases, in comparison to other state-of-the-art algorithms. To address the issue of the binding parameter, we propose the binding adaptation mechanism, which adjusts the number of protected function structures in each generation. Users only need to determine the initial value, and we found in experiments that the initial value does not significantly impact the performance and, on average, converges to the same value. For high-dimensional problems, we propose a feature selection technique based on MRMR [47]. Before starting the optimization process for each problem, a search process is performed to find the optimal number of dimensions that achieve dimensionality reduction. This helps stabilize the performance of the algorithm. The following sections provide a detailed explanation of the RBGP framework and its mechanisms.

3.1 Framework

The framework of RBGP is illustrated in Figure 3.1. The method consists of the following steps:

1. Determine the suitable dimensionality for the problem using MRMR with binary search.
2. Initialize the population of programs.
3. Divide the population into three equal parts for binding adaptation, each with a different number of protected two-layer functions.



4. Calculate the frequency of occurrence for two-layer functions within the population.
5. Select the cut point, which determines the position for crossover, while reducing the probability of selecting protected two-layer functions.
6. Divide the program trees into two parts based on the cut point. Calculate the ratio based on the range and use this ratio to select and exchange subtrees for crossover.
7. Calculate the average fitness for each of the three parts of the population and adjust the number of protected two-layer functions to match the value used by the part with the best performance.
8. Repeat from step 3 until the termination criteria are met.

Algorithm 4: Framework of RBGP

Input: P : population; X : input points; y : target points;
 b_t : number of protected binding function; r runs for feature selection;
Output: best program;

```
1  $X \leftarrow \text{FeatureSelection}(X, y, r)$ ;  
2 Randomly initialize population  $P$ ;  
3 while  $\neg \text{ShouldTerminate}$  do  
4   Randomly divide the population into three equal groups;  
5    $\text{CutPoints} \leftarrow \text{Binding}(P, b_t)$  ;  
6   for  $i = 1$  to  $|P|$  do  
7     if Apply ranging then  
8        $O_i \leftarrow \text{Ranging}(P_i, P, \text{CutPoints}, X, y)$  ;  
9     else  
10       $O_i \leftarrow \text{SubtreeMutation}(P_i)$ ;  
11    end  
12  end  
13   $P \leftarrow (\lambda + \mu)\text{-Selection}(P, O)$ ;  
14   $b_t \leftarrow \text{BindingAdaptation}(P, b_t)$ ;  
15 end  
16 return best program in  $P$  ;
```

Algorithm 4 presents the pseudo-code of the proposed algorithm, outlining the step-by-step procedure. The algorithm incorporates the protected and ranging operators, which are discussed in detail in the following subsection. In this thesis, the population is denoted

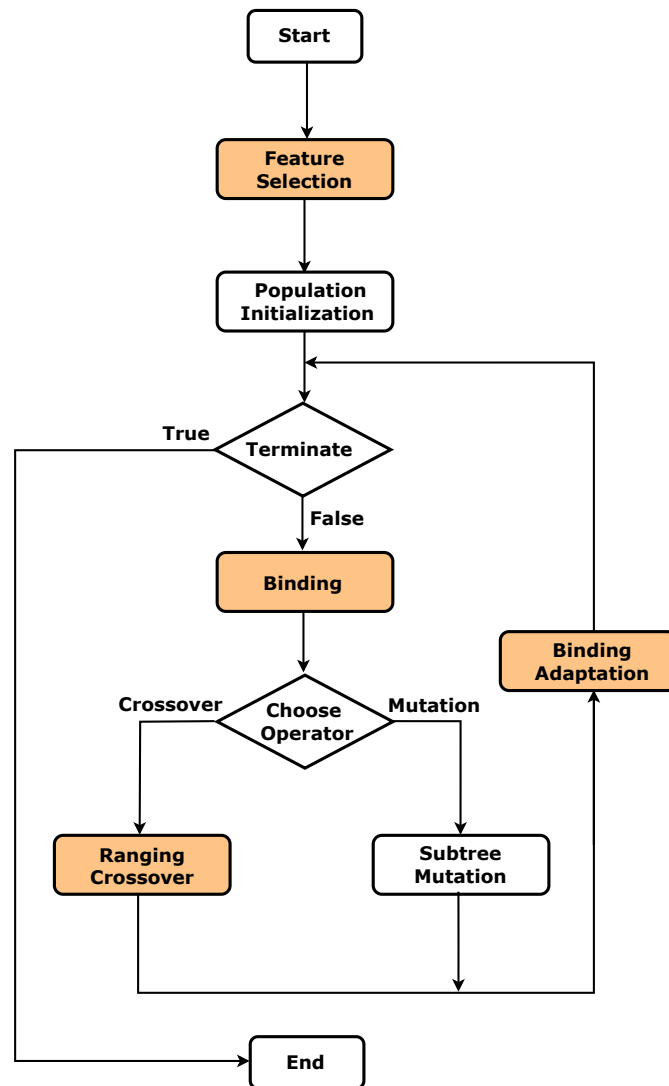


Figure 3.1: The framework of RBGP algorithm. The colored parts represent the main components proposed in this thesis.

by P , and P_i represents the i -th program in the parental population. The number of protected binding functions in each generation is denoted by b_t . The offspring of the next generation is represented by O , with O_i referring to the i -th program in the offspring population.

The "while loop" in the algorithm signifies the iteration over generations. In each iteration, either ranging crossover or subtree mutation is randomly applied to each program. After generating all offspring, the next population is selected using the $(\lambda + \mu)$ -selection. Additionally, the number of functions requiring protection in binding is updated through

statistical analysis of the fitness within the three groups of the population.

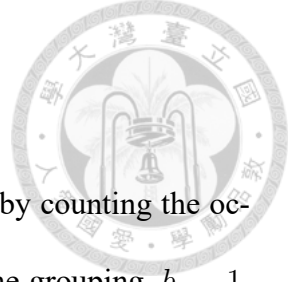


3.2 Binding Mechanism

Binding adaptation aims to automatically adjust b_t in each generation. The population is randomly divided into three equal groups at the beginning of each generation. The binding mechanism uses $b_t - 1$, b_t , and $b_t + 1$ in these three groups. In this thesis, these three groups of the population are referred to as the minus group, the base group, and the plus group, respectively. After dividing the population and determining the respective b_t values for each program, the cut point for each program in the population is determined through binding. These cut points are used for ranging crossover.

During the evolutionary process, certain subtrees are observed to occur more frequently, and in some ADF-based approaches [1, 29], these subtrees are treated as modules that need to be protected. In the RBGP algorithm, the binding operator identifies pairs of parent-child nodes as two-layer functions and treats them as modules. The target structure protected by binding is a two-layer function composed of two function nodes. This choice is made to ensure a sufficiently significant statistical quantity. Otherwise, if the quantity is not significant enough, it may lead to erroneous signals and protect inappropriate structures.

When selecting cut points for crossover, the binding mechanism aims to preserve the two-layer functions with higher frequency by reducing the probability of selecting them as cut points between parent and child programs. The detailed mechanisms of these two components are explained in the following subsections.



3.2.1 Binding

The binding mechanism is explained in Algorithm 5. It begins by counting the occurrence of two-layer functions in the population. Then, based on the grouping, $b_t - 1$, b_t , and $b_t + 1$ functions are selected for consideration. Next, the probability of selecting edges between frequently occurring two-layer functions is reduced as cut points, aiming to increase the likelihood of preserving common two-layer functions in the next generation. The protection probability for each two-layer function is calculated based on its frequency of occurrence relative to the total frequency of occurrence of all two-layer functions. Figure 3.2(a) presents an example of the binding mechanism.

Algorithm 5: Binding

Input: P : population; b_t : number of protected binding function in this generation;
Output: CutPoints: positions of one point crossover for each program;

```

1  $F \leftarrow$  Get the number of observed 2-layer functions;
2 for  $i = 1$  to  $|P|$  do
3   if  $P_i$  in minus group then
4      $F \leftarrow$  Get the top  $b_t - 1$  most frequently in  $F$ ;
5   end
6   if  $P_i$  in base group then
7      $F \leftarrow$  Get the top  $b_t$  most frequently in  $F$ ;
8   end
9   if  $P_i$  in plus group then
10     $F \leftarrow$  Get the top  $b_t + 1$  most frequently in  $F$ ;
11  end
12  Prob  $\leftarrow$  The probability of avoiding disrupting the structure in  $F$ ;
13 end
14 for  $i = 1$  to  $|P|$  do
15    $CutPoints_i \leftarrow$  DecidingCutPoint( $P$ , Prob);
16 end
17 return CutPoints;
```

Algorithm 6 demonstrates how to determine the cut points for each program based on the frequency of occurrence of two-layer functions. After statistically determining the frequency of occurrence of two-layer functions, the algorithm proceeds to select a cut



Algorithm 6: DecidingCutPoint

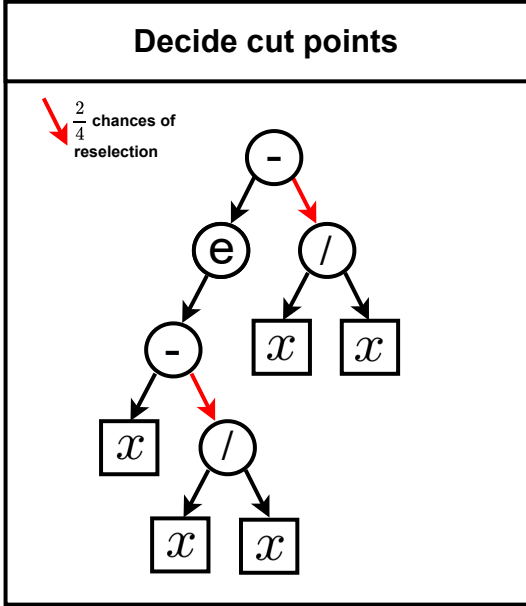
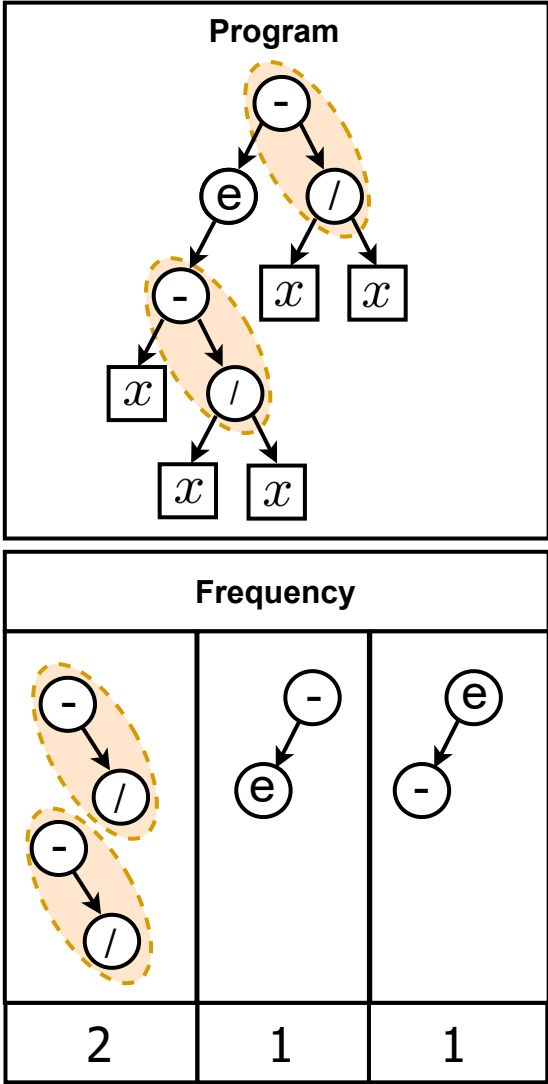
Input: P_i : i -th program; Prob: The probability of avoiding disrupting the common structure;
Output: CutPoint $_i$: position of one point crossover for P_i ;

```
1 F ← Get the number of observed 2-layer functions;
2 do
3    $edge \leftarrow$  Randomly select an edge in  $P_i$ ;
4   IsReject ← True;
5   coin ← Conduct a Bernoulli trial with a success probability of Prob $_{edge}$ ;
6   if coin is successful then
7     | IsReject ← True;
8   else
9     | IsReject ← False;
10  end
11 while Not try enough times AND IsReject is True;
12 return edge ;
```

point for each program. Firstly, a parent node-child node edge is randomly selected as a cut point, and a Bernoulli trial is conducted with the protection probability as the success probability. If the trial is successful, a new edge is randomly selected, and the process is repeated. This process continues until the Bernoulli trial fails or the process has been repeated a sufficient number of times. In our implementation, a while loop is used, and repeated 10 times is considered sufficient. Figure 3.2(b) presents an example of how to decide the cut point in a program.



Binding

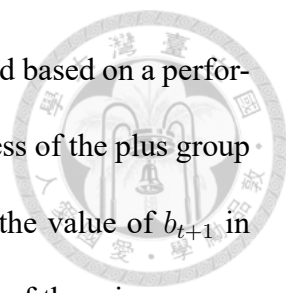


(a) **(b)**

Figure 3.2: An example of the binding mechanism.

3.2.2 Binding Adaption

Algorithm 7 presents the binding adaptation mechanism in detail. In each generation, the population is divided into three groups: the minus group, the base group, and the plus group. The number of protected functions considered in each group is adjusted as follows: $b_t - 1$ in the minus group, b_t in the base group, and $b_t + 1$ in the plus group.



At the end of the generation, the fitness of each group is evaluated based on a performance metric such as mean absolute error (MAE). If the average fitness of the plus group is better than that of the base group, indicating a lower MAE, then the value of b_{t+1} in the next generation is set to $b_t + 1$. Conversely, if the average fitness of the minus group is better, then b_{t+1} is set to $b_t - 1$. If neither group outperforms the base group, b_{t+1} remains unchanged as b_t . This rule ensures that the value of b_t adapts with each generation t , allowing for dynamic adjustments to optimize the protection mechanism for the selected functions.

An example of binding adaptation is illustrated in Figure 3.3. In this example, it is assumed that the average fitness of the Plus group performs the best. Therefore, in the next generation, the number of protected binding functions, denoted as b_{t+1} , is updated to $b_t + 1$. The adaptation formula for updating the binding functions is as follows:

$$b_{t+1} = \begin{cases} b_t + 1 & , \text{ if fitness of the plus group is better.} \\ b_t - 1 & , \text{ if fitness of the minus group is better.} \\ b_t & , \text{ if fitness of the base group is better.} \end{cases} \quad (3.1)$$

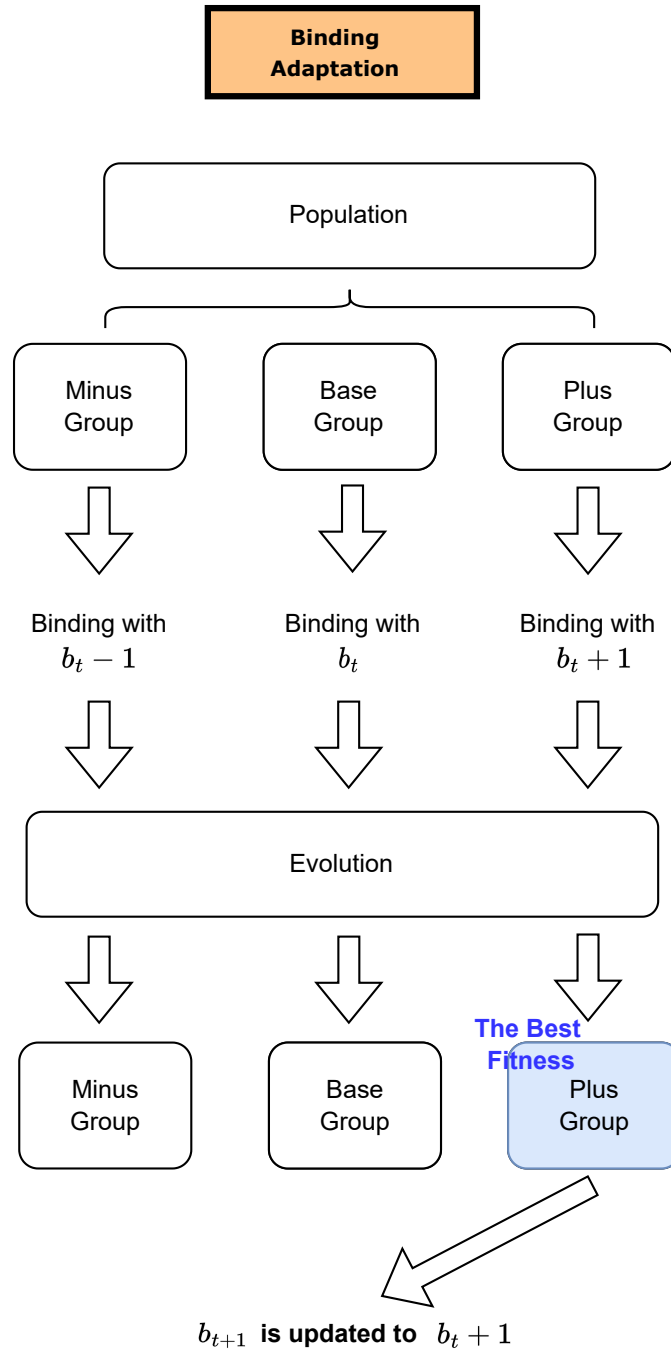
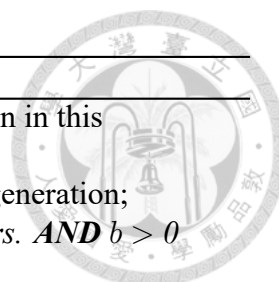


Figure 3.3: An example of the binding adaptation.

3.3 Ranging Mechanism

After the binding process, each program determines a cut point, which divides the program into two parts. One part is the subtree used for exchange, which is also an exe-



Algorithm 7: BindingAdaptation

Input: P : population. b_t : number of considered binding function in this generation;
Output: b_{t+1} : number of considered binding function for next generation;
1 **if** *The average fitness of the minus group is better than the others. AND* $b > 0$
 then
2 | $b_{t+1} \leftarrow b_t - 1$;
3 **end**
4 **if** *The average fitness of the plus group is better than the others. AND* $b < \max$
 number of combination **then**
5 | $b_{t+1} \leftarrow b_t + 1$;
6 **end**
7 **return** b_{t+1} ;

cutable program. This part is referred to as the "subtree portion" in this thesis. The other part is the portion that accepts the subtree from another program during crossover and is referred to as the "retained portion" in this thesis.

The output of a program is considered as semantics in symbolic regression genetic programming [24, 25]. The ranging mechanism aims to improve the efficiency of crossover using semantics. The definition of the range is provided as follows:

$$range(\mathcal{X}) = |\max(\mathcal{X}) - \min(\mathcal{X})|, \quad (3.2)$$

where \mathcal{X} is a vector representing the data point or output of the program. The ranging mechanism then calculates two ratios based on the range: the diff ratio and the swap ratio. The diff ratio is the ratio between the range of the target value and the range of the program's output. The swap ratio is the ratio between the range of the terminals in the subtree to be exchanged and the range of the output in the subtree portion.

After calculating the two ratios, we determine which program to perform crossover with based on whether the diff ratio is greater than 1. When the diff ratio is greater than 1, it indicates that the range of the target value is larger than the range of the current program's

output. In this case, we search for a subtree portion in another program that has a swap ratio larger than our own swap ratio, aiming to bring the diff ratio closer to 1. Conversely, when the diff ratio is less than 1, we look for a program with a swap ratio smaller than our own swap ratio to perform crossover.

In the implementation, to satisfy the need for extreme ranging and convergence in search for problem-specific structures, the ranging crossover considers the extreme cases of the diff ratio, which determines how far it deviates from 1. For example, if the diff ratio is 2, indicating that the range of the target value is twice that of the program's output value, it becomes crucial to bring the diff ratio closer to 1. On the other hand, if the diff ratio is 1.1, there is a higher probability of selecting a crossover that deviates from 1 in terms of the diff ratio. An example of a ranging crossover is illustrated in Figure 8.

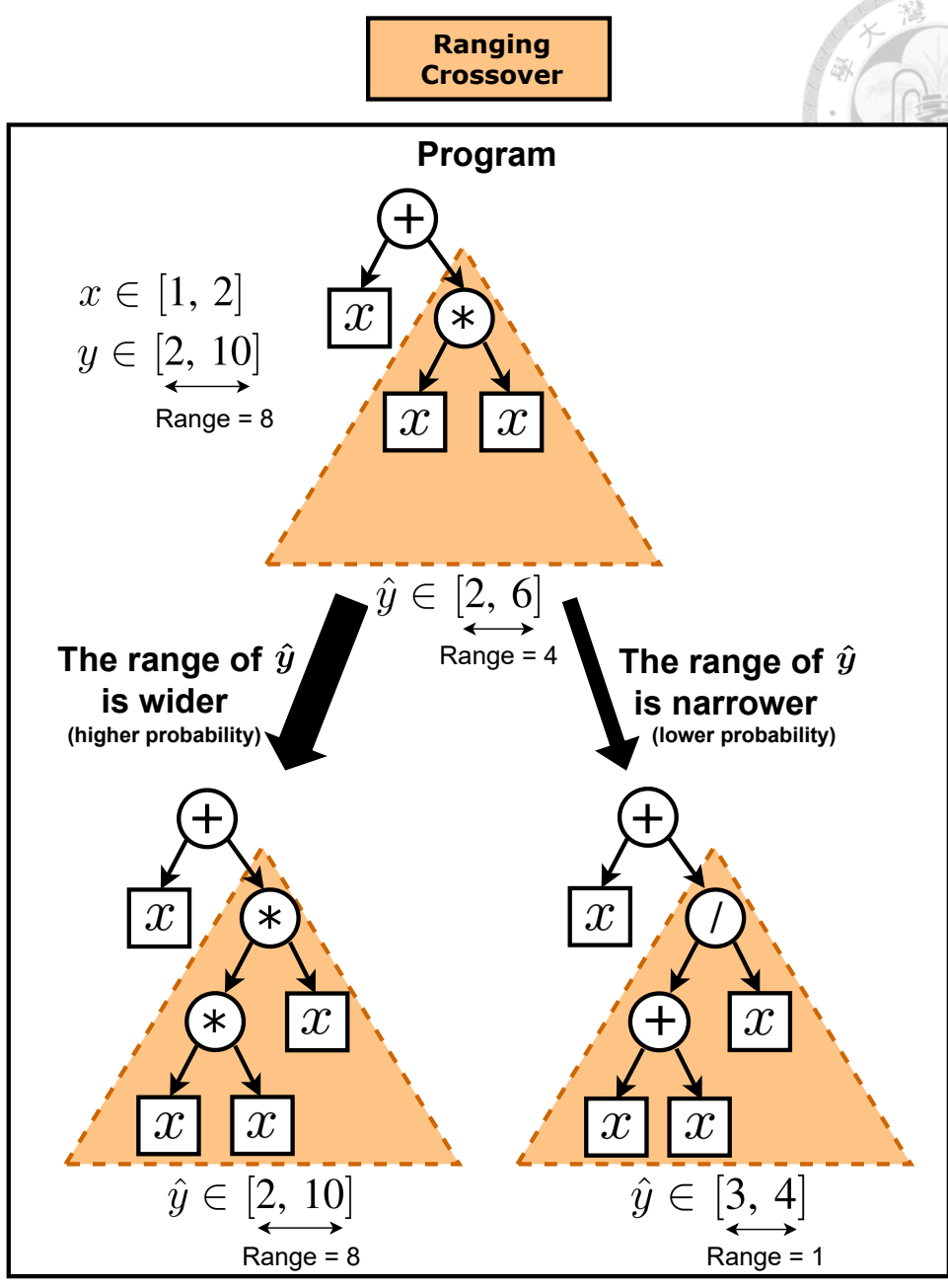


Figure 3.4: An example of the ranging crossover.

The ranging crossover is presented in Algorithm 8. $P_{i,subtree}$ and $P_{i,retained}$ are two segments of P_i that are divided by $Cutpoints_i$. X_t represents the variables used in $P_{i,subtree}$. $ratio_{i,swap}$ and $ratio_{i,diff}$ correspond to the swap ratio and diff ratio, respectively. $P_i(X)$ denotes the execution result of the i -th program with input X . The range is defined in the same way as Equation 3.2. The extremity of the ratio is quantified by applying the ratio itself to a sigmoid function and adding a bias term, then taking the absolute value



Algorithm 8: Ranging crossover

Input: P : population; P_i : Program for performing ranging;
CutPoints: positions of one point crossover for each program;
 X : input points; y : output points;
Output: O_i : offspring;

- 1 $P_{i,subtree}, P_{i,retained} \leftarrow$ Get subtree by CutPoints $_i$;
- 2 $X_t \leftarrow$ Select a random dimension used in $P_{i,subtree}$;
- 3 $ratio_{i,subtree} \leftarrow$ Range($P_{i,subtree}(X)$) / Range(X_t);
- 4 $ratio_{i,diff} \leftarrow$ Range(y) / Range($P_i(X)$);
- 5 Donor \leftarrow { program P_j used X_t };
- 6 Prob \leftarrow Calculate a probability based on the degree of ratio deviation from 1;
- 7 coin \leftarrow Conduct a Bernoulli trial with a success probability of Prob;
- 8 **if** coin is successful **then**
- 9 | $ratio_{i,diff} \leftarrow \frac{1}{ratio_{i,diff}}$;
- 10 **end**
- 11 **if** $ratio_{i,diff} > 1$ **then**
- 12 | // Choose an appropriate ratio
- 12 | Donor \leftarrow donors \cup { P_j | $ratio_{j,swap} > ratio_{i,swap}$ };
- 13 **else**
- 14 | Donor \leftarrow donors \cup { P_j | $ratio_{j,swap} < ratio_{i,swap}$ };
- 15 **end**
- 16 **if** donors $\neq \emptyset$ **then**
- 17 | $P_{donor,subtree} \leftarrow$ Randomly choose from Donor;
- 18 **else**
- 19 | $P_{donor,subtree} \leftarrow$ Randomly initialize a program with proper $ratio_{donor,swap}$;
- 20 **end**
- 21 $O_i \leftarrow$ OnePointCrossover($P_{donor,subtree}, P_{i,retained}$);
- 22 **return** O_i ;

to obtain the probability, Prob. This approach aims to make the ratio closer to 1, that is less extremity, allowing for a wider search space. Finally, based on the ratio, a suitable crossover target is selected. If there are no suitable targets in the population, a suitable swap ratio tree is randomly initialized for crossover.

3.4 Feature Selection

In the experiments, we observed that using only the ranging mechanism and binding mechanism led to less stable performance as the dimensionality increased. To address this

issue, we propose a feature selection mechanism in order to overcome this problem.

We propose a feature selection mechanism using the MRMR [47] algorithm for dimensionality reduction. MRMR is introduced in A.1. The algorithm runs GP t times, and the average fitness is calculated as an indicator to evaluate the performance in specific dimensions. Once the indicator is determined, the goal of the algorithm is to search for the best-performing dimension. In our implementation, we adopt a coarse-to-fine search strategy. When the algorithm starts, a coarse interval is determined, and the search direction (higher dimension, lower dimension, or finer search) is decided based on the performance at the starting position minus the interval, the starting position, and the starting position plus the interval. If the performance at the starting position minus the interval is the best, the search will proceed to lower dimensions. If the performance at the starting position plus the interval is the best, the search will proceed to higher dimensions. If the performance at the starting position is the best, a finer search will be conducted by narrowing down the interval and exploring the neighboring points around the starting position. This process is repeated until the interval can no longer be further refined or until a boundary is reached. An example of our proposed feature selection search method is presented in Figure 3.5.

Algorithm 9 describes how our feature selection mechanism works. X_f represents the input X with a dimension of f obtained through MRMR feature selection. UpperBound and LowerBound represent the upper and lower bounds of the search, respectively, and they change as the search becomes more refined. n_{low} , n_{mid} , and n_{high} represent the current search dimension minus the interval, the current search dimension, and the current search dimension plus the interval, respectively. f_{low} , f_{mid} , and f_{high} represent the average fitness obtained by running GP t times using inputs with dimensions n_{low} , n_{mid} , and

Feature Selection

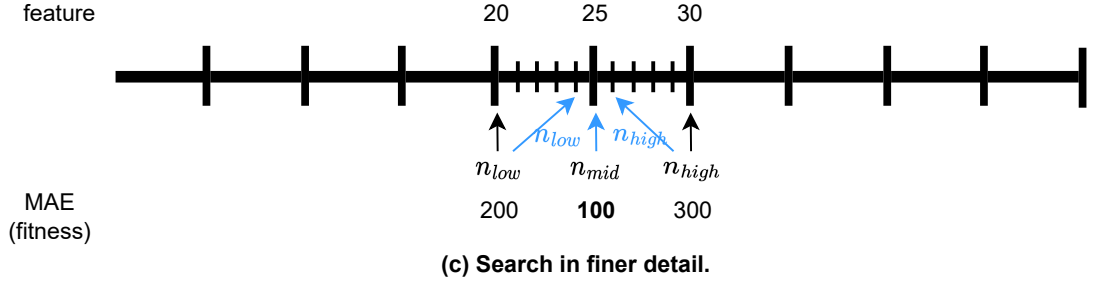
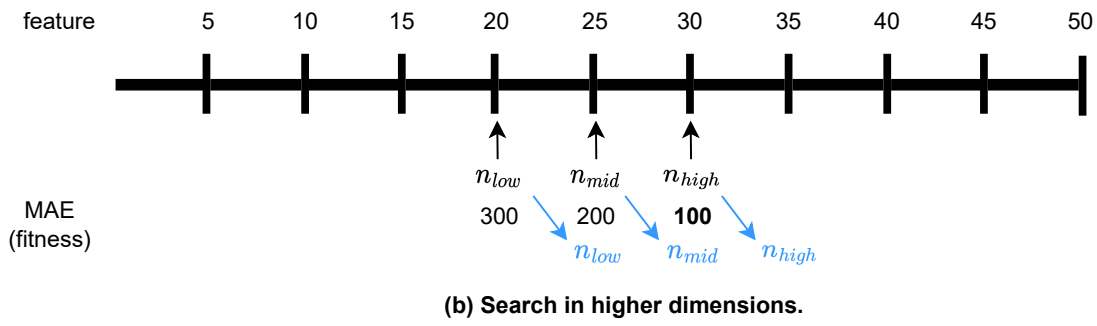
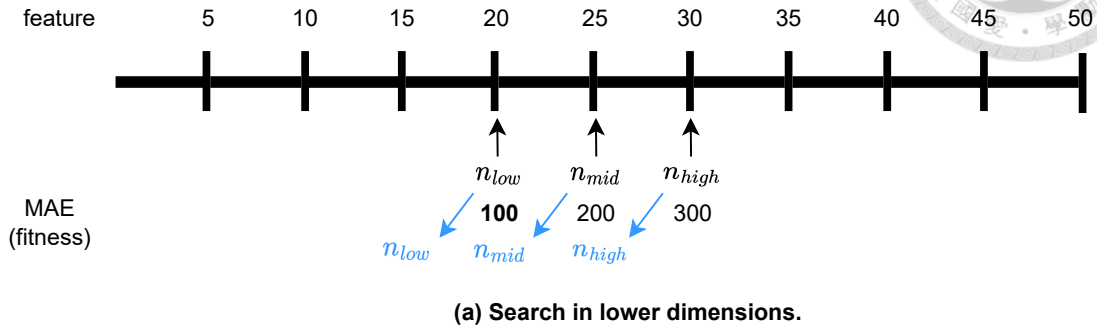
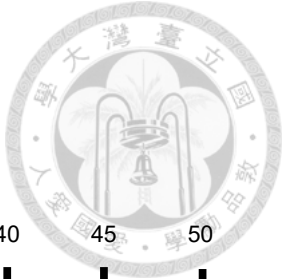


Figure 3.5: An example of proposed feature selection. The blue arrows mean the search target in the next iteration.

n_{high} , respectively.



Algorithm 9: FeatureSelection

Input: X input points; y target points; t : number of test iterations;
Output: X_n : input points after feature selection;

- 1 UpperBound \leftarrow The number of features in X ;
- 2 LowerBound \leftarrow 1;
- 3 Interval $\leftarrow \lfloor \frac{UpperBound}{10} \rfloor$;
- 4 $n_{low} \leftarrow$ Interval ;
- 5 $n_{mid} \leftarrow$ Interval \times 2 ;
- 6 $n_{high} \leftarrow$ Interval \times 3;
- 7 **while** Interval \neq 0 **AND** $n_{mid} \neq$ UpperBound **AND** $n_{mid} \neq$ LowerBound **do**
- 8 $X_{n_{low}} \leftarrow$ MRMRFeatureSelection(X, y, n_{low});
- 9 $X_{n_{mid}} \leftarrow$ MRMRFeatureSelection(X, y, n_{mid});
- 10 $X_{n_{high}} \leftarrow$ MRMRFeatureSelection(X, y, n_{high});
- 11 $f_{low} \leftarrow$ The average fitness of RunGP($X_{n_{low}}, y$) in t runs;
- 12 $f_{mid} \leftarrow$ The average fitness of RunGP($X_{n_{mid}}, y$) in t runs;
- 13 $f_{high} \leftarrow$ The average fitness of RunGP($X_{n_{high}}, y$) in t runs;
- 14 **if** f_{low} is the best in $f_{low}, f_{mid}, f_{high}$ **then**
- 15 $n_{high} \leftarrow n_{mid}$;
- 16 $n_{mid} \leftarrow n_{low}$;
- 17 $n_{low} \leftarrow \max(n_{low} - \text{Interval}, \text{LowerBound})$;
- 18 **end**
- 19 **if** f_{high} is the best in $f_{low}, f_{mid}, f_{high}$ **then**
- 20 $n_{low} \leftarrow n_{mid}$;
- 21 $n_{mid} \leftarrow n_{high}$;
- 22 $n_{high} \leftarrow \min(n_{high} + \text{Interval}, \text{UpperBound})$;
- 23 **end**
- 24 **if** f_{mid} is the best in $f_{low}, f_{mid}, f_{high}$ **then**
- 25 UpperBound $\leftarrow n_{high}$;
- 26 LowerBound $\leftarrow n_{low}$;
- 27 Interval $\leftarrow \lfloor \frac{UpperBound - LowerBound}{10} \rfloor$;
- 28 $n_{low} \leftarrow n_{mid} - \text{Interval}$;
- 29 $n_{high} \leftarrow n_{mid} + \text{Interval}$;
- 30 **end**
- 31 **end**
- 32 $X_{n_{mid}} \leftarrow$ MRMRFeatureSelection(X, y, n_{mid});
- 33 **return** $X_{n_{mid}}$;





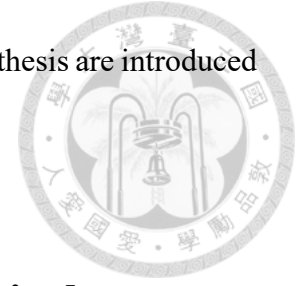
Chapter 4 Test Problems

This chapter aims to introduce the Penn machine learning benchmarks (PMLB) [17, 28] and eight real-world problems from the university of California, Irvine (UCI) dataset [2], which are widely used datasets for evaluating GP algorithms [8, 18, 46] and utilized as test problems in this thesis. PMLB is a collection of datasets specifically designed to evaluate supervised machine learning algorithms for classification and symbolic regression tasks. The dataset repository consists of two types of problems: problems with a ground-truth mathematical expression and problems only with real-world application data points.

The problems with a ground-truth mathematical expression are sourced from two repositories. One is Feynman symbolic regression dataset [27], which focuses on mathematical expressions that accurately represent static physical systems. The other one is ODE-Strogatz repository [37] includes non-linear and chaotic dynamical problems, such as models for bacterial respiration and glider patterns.

On the other hand, the problems only with real-world data points cover a wide range of domains like healthcare, business, and technology. In these problems, the goal is to predict or classify outcomes based on input features without having access to the underlying mathematical expressions or structures. These problems simulate real-world scenarios and allow for a comprehensive evaluation of machine learning algorithms in practical ap-

plications. In the following sections, the main problems tested in this thesis are introduced in detail.



4.1 Problems with Ground-truth Mathematical Expression

In this section, we present the problems used in this thesis, which have known mathematical expressions serving as ground truth. These problems include population models and bacterial respiration models in the field of biology, as well as fluid models and magnetic models in the field of physics.

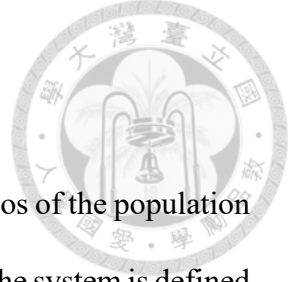
4.1.1 Shear Flow Problems

Shearflow1 and Shearflow2 are two problems that represent scenarios of rotational dynamics and phase portraits on a sphere. The rotational dynamics of an object in a shear flow are governed as follows:

$$y_{Shearflow1} = \cot(x_1) \times \cos(x_2), \quad (4.1)$$

$$y_{Shearflow2} = (\cos^2(x_1) + 0.1 \times \sin^2(x_1)) \times \sin(x_2), \quad (4.2)$$

where $y_{Shearflow1}$ and $y_{Shearflow2}$ are target value for Shearflow1 and Shearflow2 respectively. x_1 and x_2 are common inputs for both of these problems. x_1 represents "longitude," which is the angle around the z-axis. x_2 represents "latitude," which is the angle measured northward from the equator.



4.1.2 Predator-prey Problems

Predprey1 and Predprey2 are two problems that represent scenarios of the population dynamics between predators and prey in an ecological environment. The system is defined by the following equations:

$$y_{Predprey1} = x_1 \times \left(4 - x_1 - \frac{x_2}{1 + x_1}\right), \quad (4.3)$$

$$y_{Predprey2} = x_2 \times \left(\left(\frac{x_1}{1 + x_1}\right) - 0.075 \times x_2\right), \quad (4.4)$$

where $y_{Predprey1}$ and $y_{Predprey2}$ are target value for Predprey1 and Predprey2 respectively. x_1 and x_2 are common inputs for both of these problems. x_1 represents the population of the prey, while x_2 represents the population of the predator.

4.1.3 Lotka-Volterra Model of Competition

Lv1 and Lv2 are two problems that represent scenarios of the population dynamics between two species where they share the same food source and the availability of food is limited. This situation leads to a competitive relationship, for example, between rabbits and sheep. The system is defined by the following equations:

$$y_{Lv1} = 3 \times x_1 - 2 \times x_1 \times x_2 - x_1^2, \quad (4.5)$$

$$y_{Lv2} = 2 \times x_2 - x_1 \times x_2 - x_2^2, \quad (4.6)$$

where y_{Lv1} and y_{Lv2} are target value for Lv1 and Lv2 respectively. x_1 and x_2 are common inputs for both of these problems and represent the population of two species.



4.1.4 Glider Problems

Glider1 and Glider2 are two problems that represent scenarios of glider flying. The system is defined by the following equations:

$$y_{Glider1} = -0.05 \times x_1^2 - \sin(x_2), \quad (4.7)$$

$$y_{Glider2} = x_1 - \frac{\cos(x_2)}{x_1}, \quad (4.8)$$

where $y_{Glider1}$ and $y_{Glider2}$ are target value for Glider1 and Glider2 respectively. x_1 and x_2 are common inputs for both of these problems. x_1 is speed and x_2 is the angle to the horizontal.

4.1.5 Bar Magnets Problems

Barmag1 and Barmag2 are two problems that represent scenarios of a rough physical interpretation. Suppose that two bar magnets are confined to a plane, but are free to rotate about a common pin joint, as shown in Figure 4.1. The system is defined by the following equations:

$$y_{Barmag1} = -0.5 \times \sin(x_1 - x_2) - \sin(x_1), \quad (4.9)$$

$$y_{Barmag2} = 10 - \frac{(x_1 \times x_2)}{(1 + 0.5 \times x_1^2)}, \quad (4.10)$$

where $y_{Barmag1}$ and $y_{Barmag2}$ are target value for Barmag1 and Barmag2 respectively. x_1 and x_2 represent the angular orientations of the north poles of the magnets.

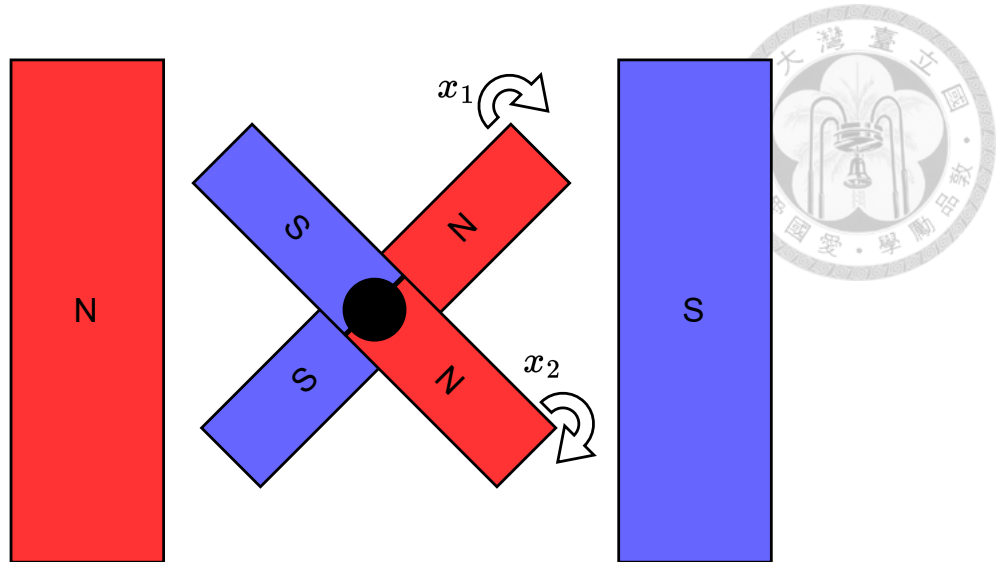


Figure 4.1: Bar magnets problem.

4.1.6 Bacterial Respiration Problems

Bacres1 and Bacres2 are two problems that represent models for respiration in a bacterial culture. The system is defined by the following equations:

$$y_{Bacres1} = 20 - x_1 - \frac{(x_1 \times y)}{(1 + 0.5 \times x_1^2)}, \quad (4.11)$$

$$y_{Bacres2} = 10 - \frac{(x_1 \times x_2)}{(1 + 0.5 \times x_1^2)} \quad (4.12)$$

where $y_{Bacres1}$ and $y_{Bacres2}$ are target value for Bacres1 and Bacres2 respectively. x_1 represent levels of nutrient and x_2 represent levels of oxygen.

4.2 Problems with Real-world Data Points

This type of test data constitutes the majority of PMLB. The dimensionality of the input data, also known as features, ranges from 2 to 100. In order to avoid the curse of dimensionality, this thesis conducts experiments using test data with dimensions below 10 and validates the effectiveness of dimensionality reduction methods using data of di-

Table 4.1: UCI test problems with their dimension.

Test Problems	Dimension
Airfoil	5
EnergyCooling	8
Concrete_compress	8
EnergyHeating	8
Tower	25
WineRed	11
WineWhite	11
Yacht	6



mension 50. By focusing on lower-dimensional test data, we can gain insights into the performance and scalability of the proposed algorithms without being overwhelmed by high-dimensional complexity.

Table 4.2 presents the PMLB benchmark problems used in this paper, along with the corresponding abbreviations and their dimensions. We categorize the dimensions as 2, 5, and 10 for further analysis. Furthermore, to test the feature selection mechanism, we also tested 9 problems with a dimensionality of 50, which are denoted as higher dimension problems.

The UCI dataset includes real-world problems from various domains, such as the chemical composition of wines and engine cooling, representing practical application scenarios. Table 4.1 presents the problems and their corresponding dimensions used in the experiments.



Table 4.2: PMLB test problems with their dimension and abbreviation.

Abbreviation	Test Problems	Dimension
Dimension = 2		
f_1	Shearflow-1	2
f_2	Shearflow-2	2
f_3	Predprey-1	2
f_4	Predprey-2	2
f_5	Lv-1	2
f_6	Lv-2	2
f_7	Glider-1	2
f_8	Glider-2	2
f_9	Barmag-1	2
f_{10}	Barmag-2	2
f_{11}	Bacres-1	2
f_{12}	Bacres-2	2
Dimension = 5		
f_{13}	Visualizing-galaxy	4
f_{14}	Sleuth-ex-1605	5
f_{15}	Fri-c0-100-5	5
f_{16}	Fri-c0-500-5	5
f_{17}	Fri-c1-100-5	5
f_{18}	Fri-c1-500-5	5
f_{19}	Fri-c3-100-5	5
f_{20}	Fri-c3-250-5	5
f_{21}	Fri-c3-500-5	5
f_{22}	Fri-c3-1000-5	5
Dimension = 10		
f_{23}	Rmftsa-ladata	10
f_{24}	Fri-c0-500-10	10
f_{25}	Fri-c1-250-10	10
f_{26}	Fri-c2-250-10	10
f_{27}	Fri-c3-500-10	10
f_{28}	Chatfield-4	12
Dimension = 50		
f_{29}	Fri-c2-500-50	50
f_{30}	Fri-c4-500-50	50
f_{31}	Fri-c3-1000-50	50
f_{32}	Fri-c2-1000-50	50
f_{33}	Fri-c1-500-50	50
f_{34}	Fri-c0-250-50	50
f_{35}	Fri-c4-1000-50	50
f_{36}	Fri-c0-1000-50	50
f_{37}	Fri-c1-1000-50	50



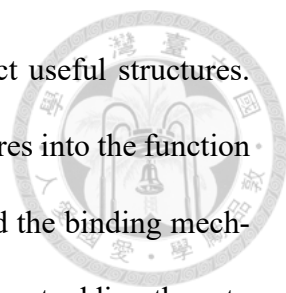


Chapter 5 Experiments and Discussions

In this chapter, the first part presents the preliminary experiments on the binding and ranging mechanisms, thereby explaining the motivation behind our approach. The second part presents the initial version of RBGP tested on real-world datasets from the PMLB, demonstrating its superior optimization capabilities compared to other SOTAs. However, it also identifies issues such as the need to adjust the parameter b_t and the instability of performance with increasing dimensions. To address these challenges, we propose the binding adaptation mechanism and the feature selection mechanism. When incorporated into RBGP, the results show improved optimization capabilities compared to the initial version. The third part presents RBGP's potential to incorporate other mechanisms, such as the constant mechanism and the operon local search mechanism.

5.1 Preliminary Experiment Results

In this section, we present the preliminary experimental results of ranging and binding mechanisms on toy problems and discuss the motivation behind the development of these two mechanisms. Section 5.1.1 presents the preliminary empirical results of the binding



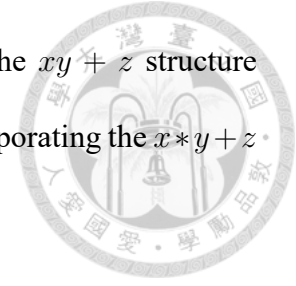
mechanism. Inspired by the ADF mechanism, our goal is to protect useful structures. However, experiments showed that incorporating some useful structures into the function set may actually lead to poorer performance. Therefore, we proposed the binding mechanism that avoids disrupting useful structures through crossover, without adding them to the function set. We validated this approach through experiments and demonstrated its improvement over SGP on the test problems. Section 5.1.2 presents the preliminary empirical results of the ranging mechanism. When considering why adding certain function structures to the function set may lead to poorer performance, we assumed that different function structures have different scaling factors from input to output. It is possible that when certain observed useful structures are used consecutively, the scaling factor becomes extremely amplified, causing the output to fall away significantly from the target value and resulting in poorer performance. This inspired us to use the scaling factor as a signal to aid evolution, which led to the development of the ranging mechanism. In the experiments, the ranging mechanism showed improved performance compared to gplearn, which is one of a SOTA, on the test problems. Furthermore, an analysis of the evolved program results confirmed that the ranging mechanism produced more suitable structures.

5.1.1 Binding

When conducting ADF-related research on $x^3 + x^2 + x$, we initially believed that the structure of multiplying x by itself, i.e., x^2 , would be useful. Therefore, we simulated the evolution of ADF and incorporated x^2 into the function set. However, as shown in Figure 5.1, the inclusion of x^2 resulted in poorer performance.

Upon further examination of the results, we discovered that the structure of $xy + z$ appeared frequently in the optimal solutions. This is because the test problem's polynomial

can be more efficiently expanded using Horner's method, where the $xy + z$ structure enhances the efficiency of evolution. As depicted in Figure 5.1, incorporating the $x * y + z$ structure yielded significant improvements for SGP.



This inspired us to develop the binding mechanism, as incorporating commonly used function structures into the function set could potentially result in poorer performance due to overuse. Instead, our approach is to protect these common function structures from being disrupted by crossovers. By employing a selection mechanism that preserves efficient structures, we achieve a more stable and reliable method. Figure 5.2 illustrates the results of SGP + Binding and SGP on the toy problem. It can be observed that SGP + binding consistently shows improvements across different mutation rates.

In addition, to confirm the beneficial effect of binding on evolution, we analyzed the relationship between fitness and binding numbers. The results are shown in Figure 5.3. The x-axis represents populations divided into five bins based on their fitness performance, and the average binding numbers are calculated for each bin. It can be observed that programs with better performance have, on average, a higher number of bindings. The correlation coefficient indicates a positive correlation between fitness and binding numbers.

Regenerate response

The experiment settings are described in Table 5.1. Note that for fitness, we adopt MAE as the evaluation metric, where a lower value indicates better performance. For the div^* function, in order to ensure a well-defined operation, we refer to the definition in Equation 5.1, which is also commonly used by other state-of-the-art algorithms.

Table 5.1: The experiment settings for the binding preliminary experiments.

Parameter	Value
Generation	10
Population	200
Fitness	MAE (lower is better)
Function set	add, sub, mul, div*
Terminal set	x
Test problems	$x^3 + x^2 + x$
Sample points	100 sample points uniformly distributed from -5 to +5
Experiment runs	100

$$div^*(x, y) = \begin{cases} \frac{x}{y}, & \text{if } |y| \geq 0.001 \\ 1, & \text{otherwise} \end{cases} \quad (5.1)$$

Figure 5.1 illustrates the results of the preliminary experiments on binding, demonstrating that incorporating seemingly appropriate structures into the function set may actually lead to worse results. The graph shows the average fitness of the best-performing program out of 100 runs.

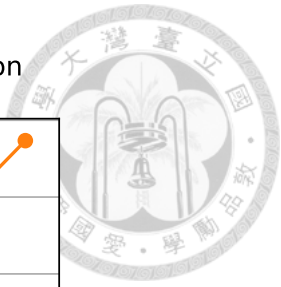
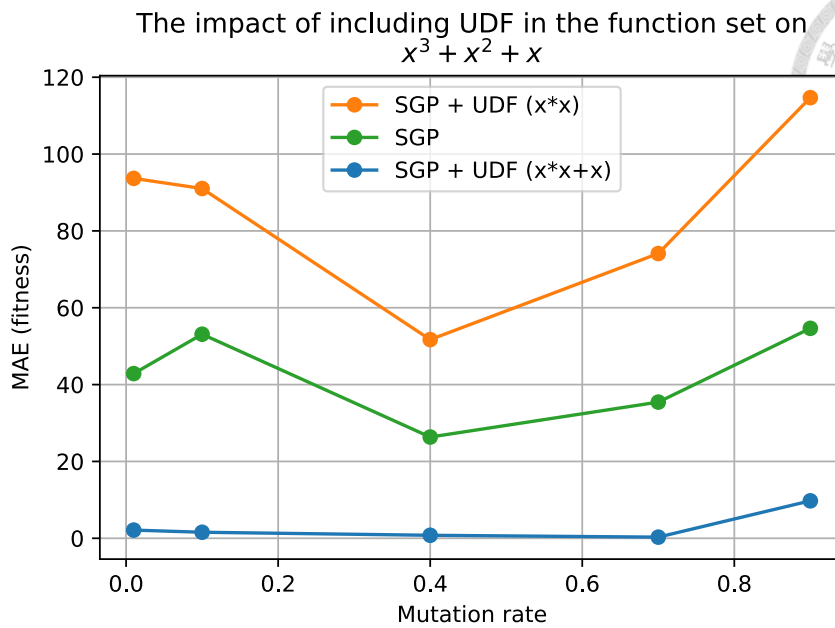


Figure 5.1: The graph shows the average fitness of the best-performing program out of 100 runs. Lower fitness means better performance. SGP + UDF ($x * x$) performs even worse in this case compared to SGP, but incorporating a suitable UDF ($x * x + x$) significantly improves the performance.

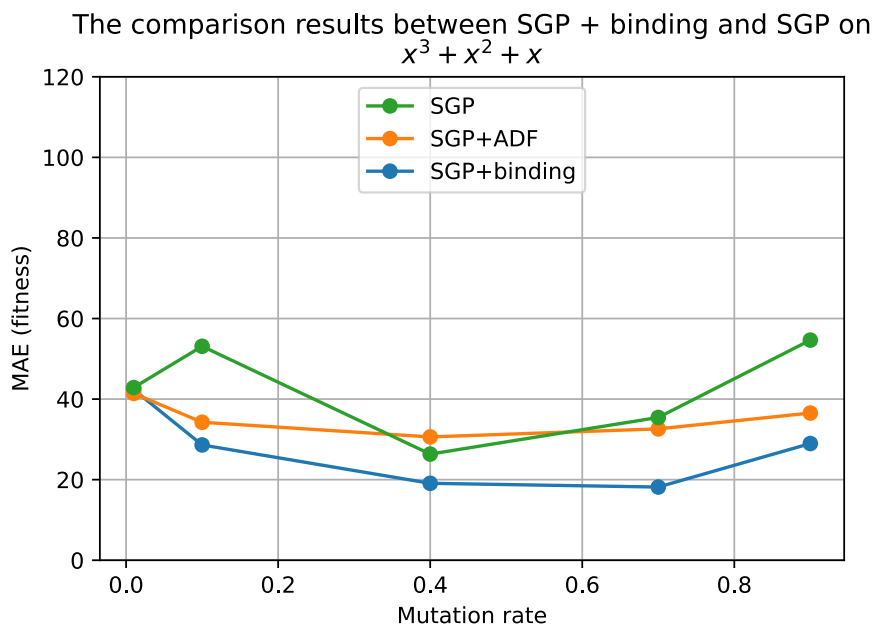


Figure 5.2: The graph shows the average fitness of the best-performing program out of 100 runs. Lower fitness means better performance. SGP + binding shows lower average MAE compare to SGP and SGP + ADF



The relationship between average fitness and the number of bindings

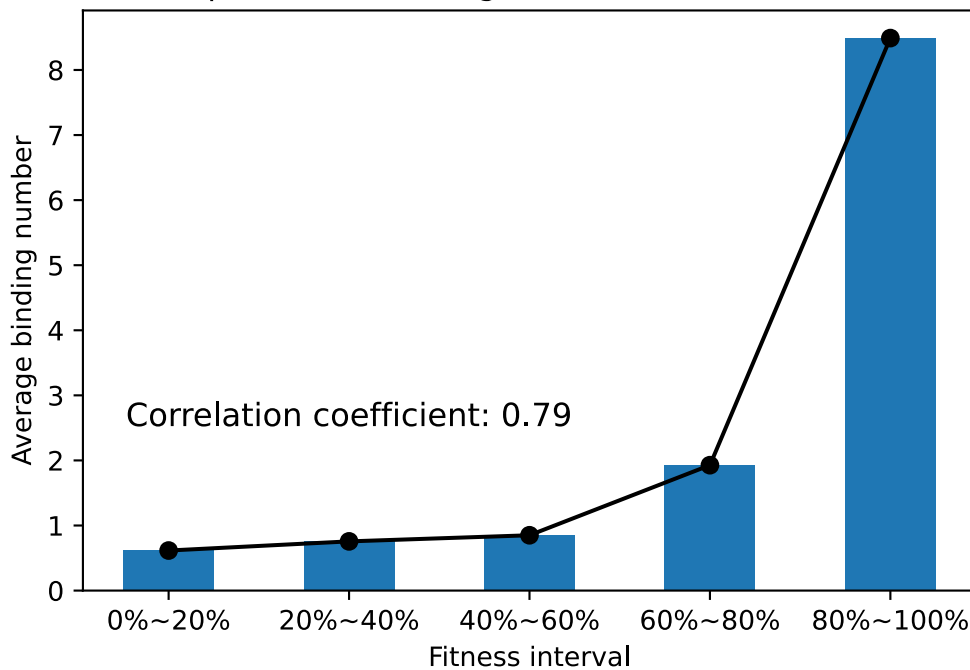


Figure 5.3: The relationship between average fitness and the number of bindings. The results show a correlation coefficient greater than 0.7, indicating a strong correlation. Each bar represents the average binding number of programs, calculated at every 20% based on their fitness performance. The x-axis represents increasing percentages of fitness performance, with a higher value indicating better performance.



5.1.2 Ranging

In the preliminary experiments of the binding mechanism, we observed that incorporating useful structures into the function set could lead to poorer results. We attributed this phenomenon to the scaling effect of the structures on input values. If a structure appears continuously, the scaling factor can become extremely large, which poses a risk when adding it to the function set. For example, the structure x^2 may combine to form x^4 or x^6 , rapidly increasing the amplification factor. This hypothesis not only explains why common structures should be protected rather than easily added to the function set but also suggests that the scaling factor of structures can serve as a signal for evolution. By avoiding structures that undergo extreme amplification or extreme reduction, we can leverage the concept as ranging mechanism to guide the evolutionary process.

To validate the effectiveness of the ranging mechanism, we compared it with `gplearn` on a test problem. The functions used in the experiment were all unary functions, as they directly reflect the signal of ranging. The results, as shown in Table 5.3, indicate that ranging mechanism + SGP (RGP) outperforms `gplearn`. Additionally, we examined the evolved structures of RGP and `gplearn`, as illustrated in Figure 5.4. Overall, RGP tends to evolve structures that are closer to the actual function.

The experiment settings are described in Table 5.2. Note that for the `log*` function and `inv*` function, in order to ensure a well-defined operation, we refer to the commonly-used definition in Equation 5.2 and Equation 5.3 respectively.

Table 5.3 presents the comparison results between RGP and `gplearn`. The values reported are the maximum, median, minimum, and mean values of the best-performing program in the population across 100 experimental runs. These values are rounded to two

Table 5.2: The experiment settings for the ranging preliminary experiments.

Parameter	Value
Generation	10
Population	200
Fitness	MAE (lower is better)
Function set	addOne, subOne, log*, exp, inv*, neg
Terminal set	x
Test problems	sigmoid function
Sample points	100 sample points uniformly distributed from -5 to +5
Experiment runs	100

decimal places, and measured in terms of their MAE.

Figure 5.4 illustrates the evolutionary structures of RGP and gplearn methods. It shows the comparison between the solutions of the best, median, and worst fitness structures obtained from 100 experiments, and the target function sigmoid. The x-values represent the sample points defined in the experimental setup. It can be observed that, overall, RGP evolves structures that are closer to the target function.

$$\log^*(x) = \begin{cases} \log(x), & \text{if } x > 0.001 \\ 0, & \text{otherwise} \end{cases} \quad (5.2)$$

$$\text{inv}^*(x) = \begin{cases} \frac{1}{x}, & \text{if } |x| \geq 0.001 \\ 0, & \text{otherwise} \end{cases} \quad (5.3)$$

Table 5.3: The results of ranging preliminary experiments compared with gplearn. These values are rounded to two decimal places, and measured in terms of their MAE.

	RGP	gplearn
Min	0	0
Median	0	0.10
Max	0.18	0.34
Mean	0.03 ± 0.04	0.11 ± 0.08

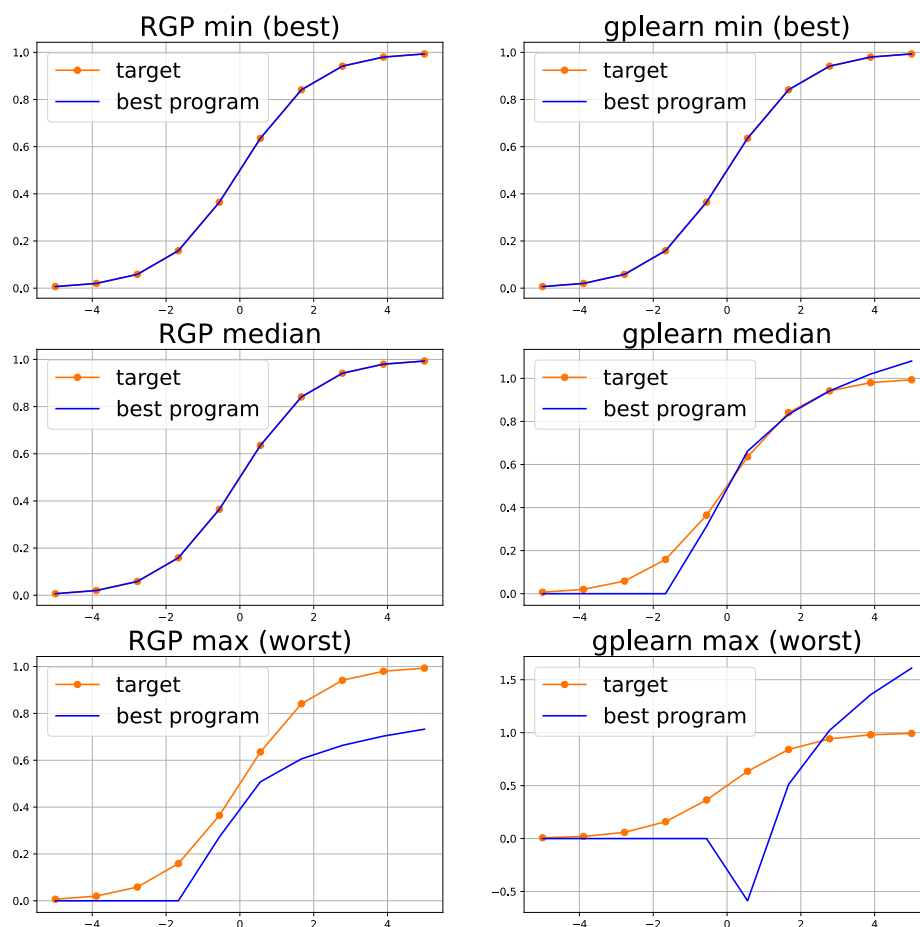


Figure 5.4: The evolutionary structures of RGP and gplearn. In 100 experimental runs, both methods yielded a set of 100 best programs. This figure shows the best-performing program, the median program, and the worst-performing program in both methods. RGP and gplearn show the same results in the best case. However, RGP achieves closer fit results in the median and maximum cases.



5.2 Experiment Results on Benchmarks

In this section, we present the results of the first version of RBGP in comparison to `gplearn`, `ellynGP`, and `GP-GOMEA`, which are SOTAs, on the test problems in 5.2.1. RBGP in this version incorporates only the ranging mechanism and binding mechanism. From the experiments, we identified two areas for improvement. Firstly, the number of protected function structures, denoted as b_t before, is a parameter that needs to be determined. To address this, we propose the binding adaptation mechanism and compare it with `gplearn`, `ellynGP`, `EPLEX`, `AFP`, and `GP-GOMEA` in 5.2.2. Additionally, experimental results show that the initial value settings for binding adaptation have minimal impact on the overall performance. Second, as the dimensionality increases, RBGP tends to exhibit instability. To address this, we introduce the feature selection mechanism and demonstrate its improvement when combined with binding adaptation RBGP in 5.2.3.

5.2.1 RBGP

This subsection presents the comparison results between RBGP and `gplearn`, `ellynGP`, `GP-GOMEA`, as well as the relationship between the stability of RBGP's performance and the dimensionality. In the following experimental results, the initial version of the algorithm, RBGP, is denoted as `RBGP- α` . The experiment settings are described in Table 5.4. Note that for the `sqrt*` function, in order to ensure a well-defined operation, we refer to the definition in Equation 5.4, which is also commonly used by other SOTA algorithms.

Table 5.5 presents the comparison results between RBGP and `gplearn`, `ellynGP`, and

Table 5.4: The experiment settings for RBGP experiments.

Parameter	Value
Generation	20
Population	200
Fitness	MAE (lower is better)
Function set	add, sub, mul, div*, log*, exp, sqrt*, square
Terminal set	x
Test problems	$f_1 \sim f_{37}$
Experiment runs	100

GP-GOMEA. The values in the table represent the fitness, which is measured by MAE. A lower MAE indicates better performance of the algorithm. The bold values in the table indicate the best performance among the four algorithms for each respective problem.

Figure 5.5 displays the ranking analysis of the four methods, categorized by the dimensionality of the test problems (low, medium, high, refer to Table 4.2). It can be observed that RBGP exhibits more unstable performance as the dimensionality increases compared to the other methods.

$$\mathit{sqrt}^*(x) = \mathit{sqrt}(|x|). \quad (5.4)$$

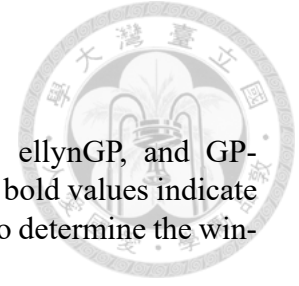


Table 5.5: The comparison of first version RBGP and gplearn, ellynGP, and GP-GOMEA. The values in the table represent the average MAE, and the bold values indicate the lowest MAE. The comparison is made to the third decimal place to determine the winner is denoted with *.

Test Problem	GP-GOMEA	ellynGP	gplearn	RBGP- α
f1	3.57E+03	6.38E-01	1.44E-01	1.18E-01
f2	8.42E+02	1.36E+00	1.88E-01	1.75E-01
f3	4.08E+03	7.12E-01	5.79E-01	5.61E-01
f4	6.74E+04	1.58E+01	6.84E-01	6.11E-01
f5	5.87E+09	4.72E-01	1.75E-01	1.56E-01
f6	3.96E+05	6.34E-01	2.93E-01	2.54E-01
f7	6.75E+02	3.58E-01	3.56E-01	3.12E-01
f8	3.04E+06	7.01E-01	6.73E-01	6.56E-01
f9	1.66E+03	3.32E+281	1.30E-01	1.08E-01
f10	4.93E+04	2.57E-01	1.14E-01	9.36E-02
f11	4.57E+00	1.35E+00	6.97E-01	5.87E-01
f12	8.73E+00	7.08E+01	8.69E-01	7.00E-01
f13	3.10E+05	2.25E+03	4.37E+02	4.05E+02
f14	3.22E+10	8.76E+00	8.08E+00	7.99E+00
f15	3.24E+01	7.62E-01	6.64E-01	6.40E-01
f16	3.44E+02	7.16E-01	6.44E-01	6.78E-01
f17	1.33E+04	7.97E-01	7.10E-01	6.91E-01
f18	6.60E+01	7.77E-01	6.92E-01	6.91E-01
f19	1.31E+01	7.39E-01	6.18E-01	6.76E-01
f20	3.74E+01	7.86E-01	6.89E-01	6.76E-01
f21	3.63E+03	7.94E-01	6.93E-01	6.69E-01
f22	1.58E+03	7.69E-01	6.38E-01	6.48E-01
f23	1.94E+01	3.52E+88	1.73E+00	1.72E+00
f24	7.11E+01	8.01E-01	7.34E-01	6.57E-01
f25	4.65E+20	1.27E+01	1.17E+01	1.20E+01
f26	2.92E+03	7.49E-01	7.13E-01	7.23E-01
f27	1.45E+01	8.30E-01	7.52E-01	7.12E-01
f28	5.81E+02	7.80E-01	7.23E-01	6.99E-01
f29	9.69E+01	1.19E+00	7.89E-01	7.68E-01
f30	3.65E+04	8.24E-01	7.70E-01	7.34E-01
f31	2.11E+02	1.04E+00	7.64E-01	7.29E-01
f32	1.35E+02	6.63E+47	7.99E-01	7.61E-01
f33	6.07E+02	7.81E+00	7.66E-01	7.68E-01
f34	4.26E+02	7.45E-01	7.83E-01	7.66E-01
f35	1.25E+04	1.06E+00	7.71E-01	7.50E-01
f36	6.06E+02	7.36E-01	7.82E-01	7.92E-01
f37	3.36E+02	1.43E+00	7.47E-01	7.70E-01

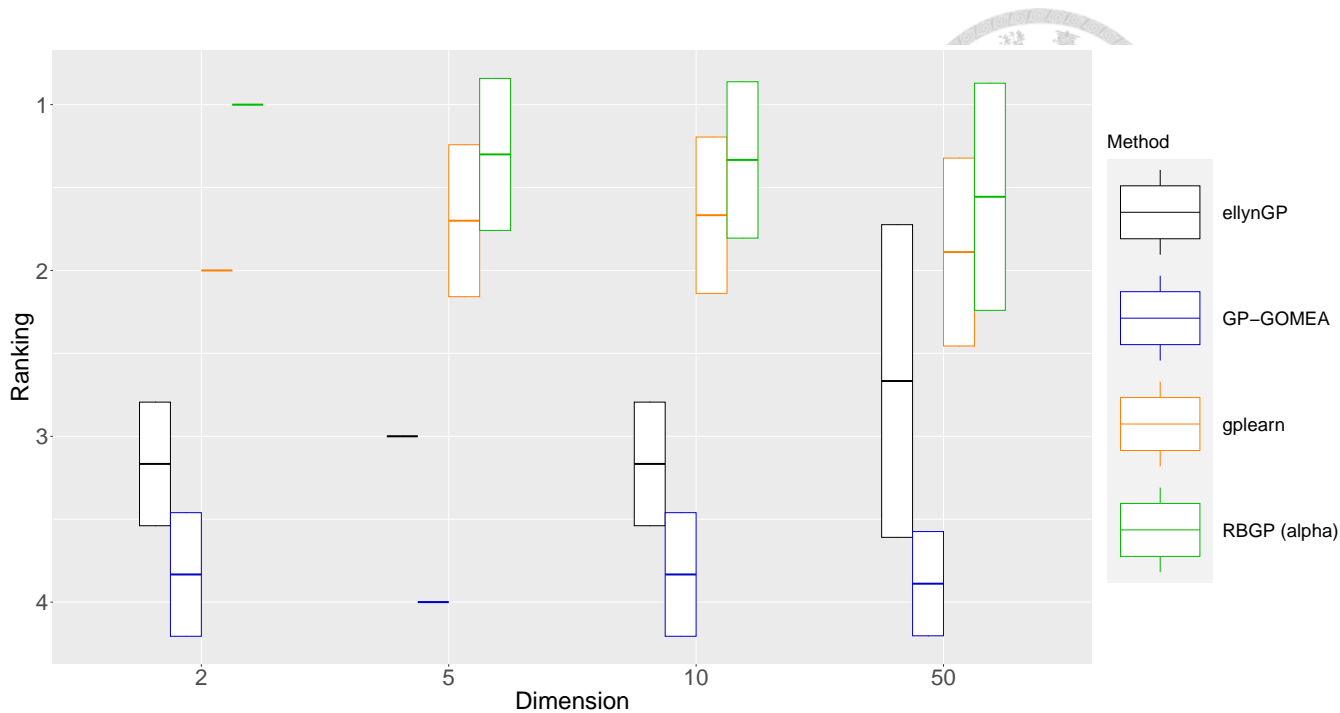


Figure 5.5: The ranking analysis of the four methods. The value is average ranking \pm standard deviation of each method. On average, RBGP outperforms other methods in terms of ranking across different dimensional problems. However, there is a trend of increased instability as the dimensionality increases.

5.2.2 RBGP with Binding Adaptation

This subsection presents the results of RBGP with binding adaptation mechanism compared to gplearn, ellynGP, EPLEX, AFP, and GP-GOMEA. It also shows the convergence of the average b_t value in binding adaptation. The experiment settings are the same as Table 5.4.

Table 5.6 presents the comparison results between RBGP with binding adaptation and the other 5 SOTAs. In Table 5.6, "Lowest MAE" represents the number of results where a method achieved the lowest MAE in the corresponding dimension and parameter settings. "Statistical test" shows the results of the statistical tests performed. We used the Kruskal-Wallis test to compare RBGP and GP-GOMEA, as GP-GOMEA failed the Homogeneity of Variance test. The other methods were compared using the Student t-test. The symbols

Table 5.6: RBGP with binding adaptation (RBGP- β) compared with other methods. In this table, "Lowest MAE" represents the number of results where a method achieved the lowest MAE. "Statistical test" shows the results of the statistical tests performed. The symbols "+" represent the number of times RBGP- β significantly outperformed other methods, " \approx " represents the number of times RBGP- β showed no significant difference compared to other methods, and "-" represents the number of times other methods significantly outperformed RBGP- β .

type	RBGP- β	GP-GOMEA	gplearn	ellynGP	ELPEX	AFP
Dimension = 2						
Lowest MAE	12	0	0	0	0	0
Statistical test (+/ \approx /-)	-	10/2/0	8/4/0	9/3/0	7/5/0	5/7/0
Dimension = 5						
Lowest MAE	8	0	0	0	2	0
Statistical test (+/ \approx /-)	-	7/3/0	7/3/0	9/1/0	8/1/1	7/2/1
Dimension = 10						
Lowest MAE	5	0	0	0	1	0
Statistical test (+/ \approx /-)	-	4/2/0	4/1/1	5/1/0	3/2/1	3/2/1
Dimension = 50						
Lowest MAE	5	0	2	0	1	1
Statistical test (+/ \approx /-)	-	9/0/0	7/0/2	9/0/0	8/0/1	8/1/0

"+" represent the number of times RBGP- β significantly outperformed other methods, " \approx " represents the number of times RBGP- β showed no significant difference compared to other methods, and "-" represents the number of times other methods significantly outperformed RBGP- β . The significance level is 0.05 for all tests. On average, RBGP with binding adaptation demonstrates better optimization capability and maintains similar performance as generations and population size increase. The raw fitness results are listed in Table 5.9.

Figure 5.6 illustrates that under the binding adaptation mechanism, the initial setting of b_t tends to converge to similar values on average. Therefore, users have more flexibility in setting the initial value of b_t .

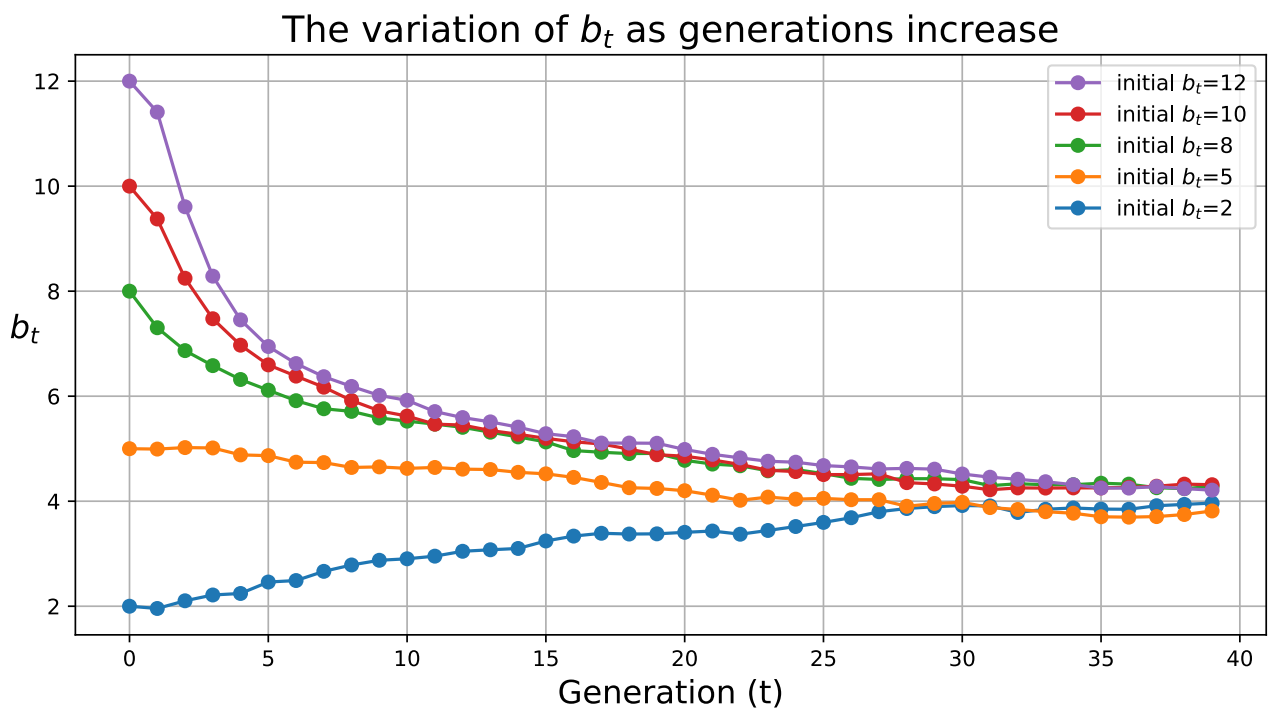


Figure 5.6: The plot of b_t variation across all test problems as generations increase. Each line represents a different initial value setting for b_t , and each point represents the average b_t across all test problems at that generation. On average, b_t converges to a constant value regardless of its initial value.

Table 5.7: The average dimension after feature selection.

Problem	f5	f10	f50
Average dimension	4.22	7.33	36.57



5.2.3 RBGP with Binding Adaptation and Feature Selection

This subsection presents the improvements brought by the feature selection mechanism added to RBGP with binding adaptation. The experiment settings are the same as Table 5.4 except for test problems are $f_{13} \sim f_{37}$. Figure 5.7 illustrates the statistical analysis of incorporating the feature selection mechanism into RBGP with binding adaptation. It can be observed that as the dimensionality increases, the improvement brought by the feature selection mechanism becomes more significant.

Figure 5.8 presents the comparison between RBGP + binding adaptation (RBGP- β) and RBGP + binding adaptation + feature selection (RBGP) against ellynGP, gplearn, GP-GOMEA, EPLEX, and AFP. Each point represents the ratio of the method achieving the lowest MAE in a particular dimensional problem. The raw fitness results are listed in Table 5.9. Additionally, Table 5.8 displays the raw fitness comparison between RBGP and other methods on the UCI dataset.

Due to the additional number of evaluations required by the feature selection mechanism compared to the other three proposed mechanisms, we compared the increased cost of incorporating feature selection at different dimensions, measured in terms of the number of evaluations. The results are presented in Figure 5.9. Furthermore, Table 5.7 presents the average dimension of the problems after feature selection.

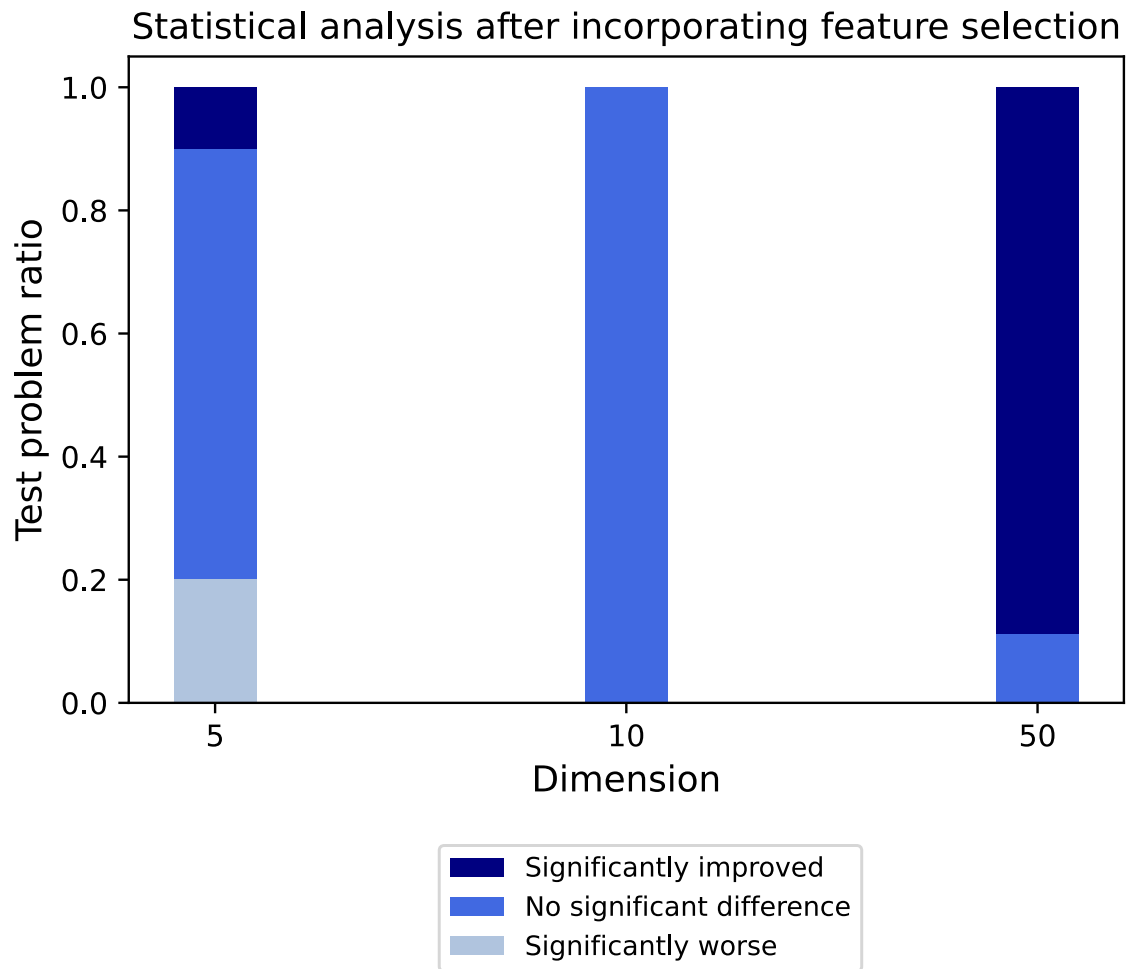


Figure 5.7: Statistical analysis after incorporating feature selection. Using t-test with a significance level of 0.05. Comparison between RBGP- β and RBGP. Since feature selection yielded unchanged results in the case of dimension 2, we only discuss datasets with dimensions of 5 or higher. It can be observed that as the dimension increases, the benefits of feature selection become more pronounced.

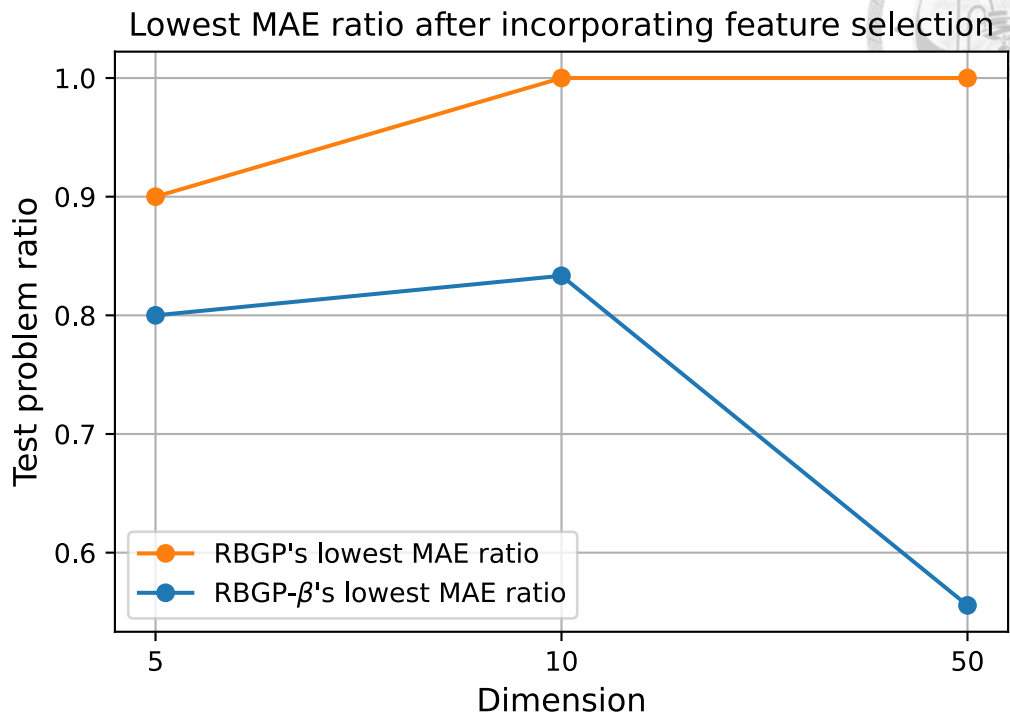


Figure 5.8: Lowest MAE ratio after incorporating feature selection. In terms of MAE, RBGP shows improvement across all dimensions.

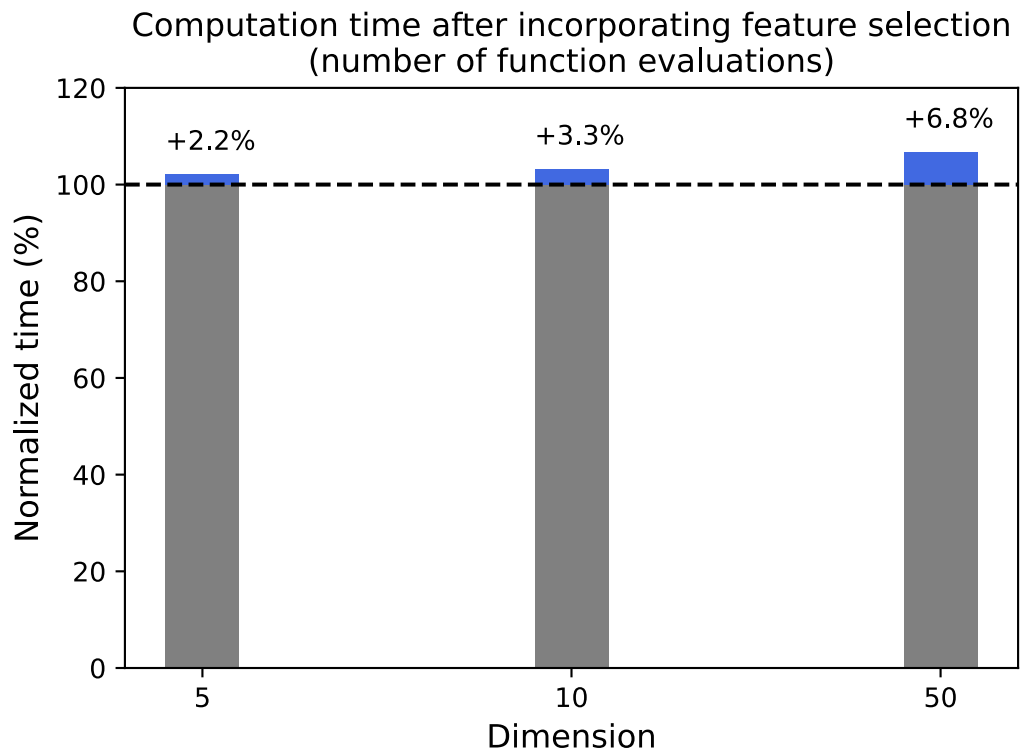


Figure 5.9: The computation time after incorporating feature selection. The additional time cost in terms of the number of evaluations is not significant.



5.3 RBGP with Other Mechanisms

In this section, we present the results of combining RBGP with other mechanisms, demonstrating RBGP’s potential as a component for other algorithms and its compatibility with machine learning. In Section 5.3.1, we compare the performance before and after incorporating the constant mechanism. The experiment shows that the addition of constants has a noticeable effect in high-dimensional problems, while its impact is limited in low-dimensional problems. In Section 5.3.2, we compare the performance before and after incorporating the local search method in operon. The experiment reveals that RBGP shows greater improvement when combined with local search compared to other methods incorporating the same mechanism.

5.3.1 Constant

The experiment settings are the same as Table 5.4 except for terminal set includes real-valued constant sampled from (0,1). Figure 5.10 illustrates the statistical analysis of incorporating the constant mechanism into RBGP. It can be observed that adding the constant mechanism may lead to a decline in performance for lower-dimensional problems,

Table 5.8: Raw fitness of RBGP compared with other SOTAs on UCI datasets. The bold font indicates the lowest MAE.

	RBGP	GP-Gomea	gplearn	AFP	ellynGP	ELPEX
Airfoil	1.36E+01	6.34E+00	1.76E+01	9.76E+01	1.03E+02	1.14E+14
EnergyCooling	1.21E+01	1.89E+01	1.14E+01	1.86E+01	1.55E+01	2.20E+01
Concrete_compress	3.93E+00	4.23E+00	3.97E+00	6.71E+00	3.38E+03	2.85E+12
EnergyHeating	4.27E+00	4.29E+00	3.77E+00	1.22E+02	4.54E+00	6.67E+00
Tower	5.52E+01	7.20E+01	5.54E+01	4.38E+04	8.83E+01	4.96E+54
WineRed	5.02E-01	6.47E-01	5.79E-01	5.79E-01	7.15E-01	6.70E-01
WineWhite	6.41E-01	9.69E-01	6.50E-01	1.15E+00	9.06E-01	6.70E-01
Yacht	2.59E+00	4.92E+00	2.68E+00	6.57E+00	5.54E+00	6.11E+00

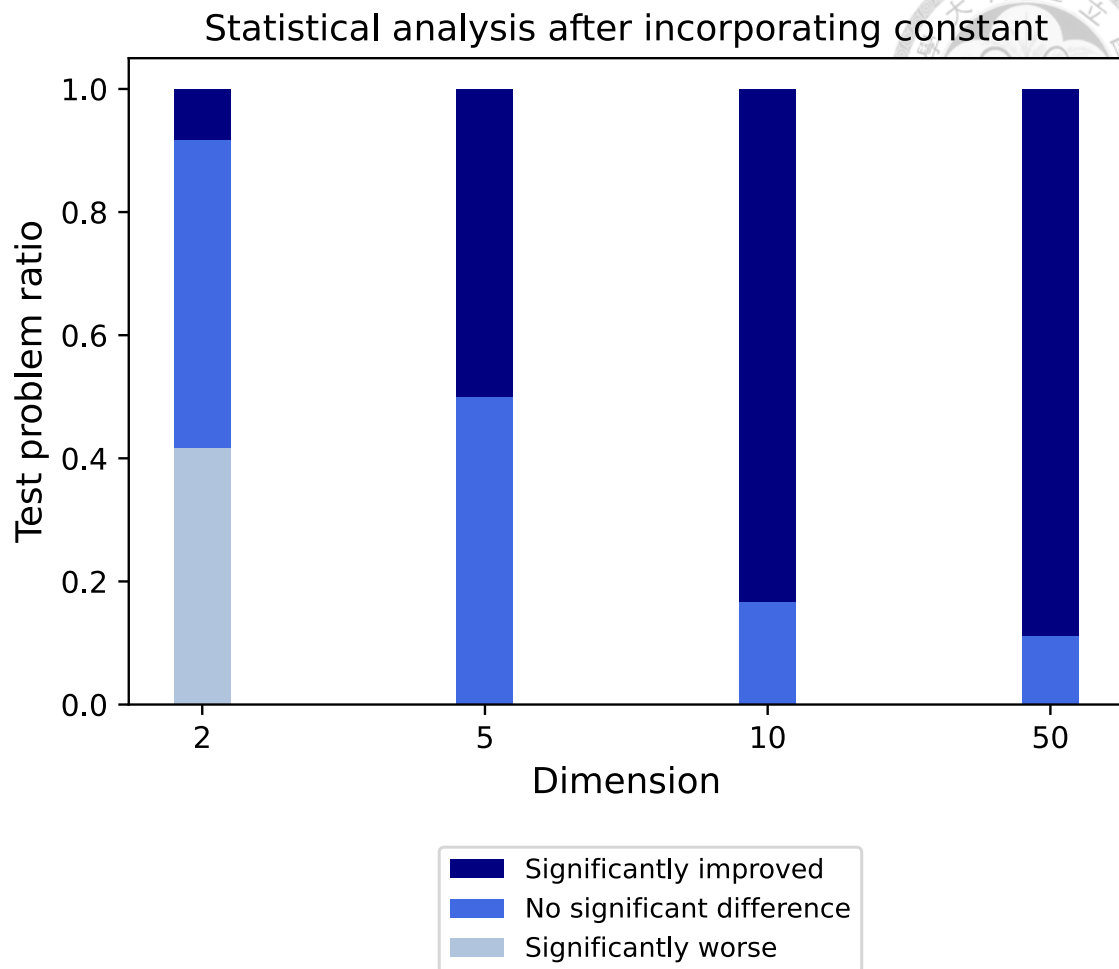
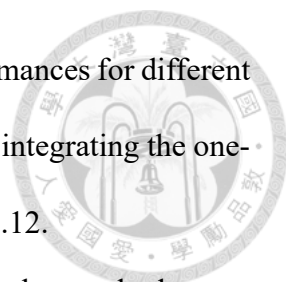


Figure 5.10: Statistical analysis after incorporating constant. Using t-test with a significance level of 0.05. Incorporating constant tends to show more improvement as the dimensionality increases.

while for higher-dimensional problems, the addition of the constant mechanism shows significant improvement. The raw fitness results are listed in Table 5.10.

5.3.2 Operon Local Search

Operon [5] and one-layer operon local search are introduced in appendix A.2. The experiment settings are the same as Table 5.4. Table 5.10 presents the statistical analysis of incorporating the one-layer operon local search mechanism into RBGP, comparing it with other methods combined with one-layer operon local search. It can be observed that



incorporating the one-layer operon local search yields different performances for different GP methods. RBGP, compared to other methods, is more suitable for integrating the one-layer operon local search. The raw fitness results are listed in Table 5.12.

Table 5.11: RBGP with one-layer operon local search compared with other methods combined with one-layer operon local search. The symbols ”+” represent the number of times RBGP significantly outperformed other methods, ”≈” represents the number of times RBGP showed no significant difference compared to other methods, and ”-” represents the number of times other methods significantly outperformed RBGP. The significance level is 0.05 for all tests.

type	RBGP	GP-GOMEA	gplearn	ellynGP	ELPEX	AFP	Operon
Lowest MAE	20	0	0	0	0	0	17
Statistical test (+/≈/-)	-	37/0/0	34/2/1	34/3/0	34/2/1	33/4/0	12/13/12

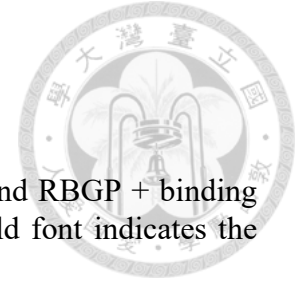


Table 5.9: Raw fitness of RBGP + binding adaptation (RBGP- β) and RBGP + binding adaptation + feature selection (RBGP) with other SOTAs. The bold font indicates the lowest MAE.

	RBGP- β	GP-GOMEA	gplearn	ellynGP	ELPEX	AFP	RBGP
f1	1.14E-01	3.57E+03	1.44E-01	6.38E-01	1.16E+00	9.61E+24	-
f2	1.50E-01	8.42E+02	1.88E-01	1.36E+00	2.56E-01	2.34E-01	-
f3	5.46E-01	4.08E+03	5.79E-01	7.12E-01	5.57E-01	7.91E-01	-
f4	5.93E-01	6.74E+04	6.84E-01	1.58E+01	9.14E-01	4.21E+00	-
f5	1.36E-01	5.87E+09	1.75E-01	4.72E-01	4.90E-01	3.73E+03	-
f6	2.69E-01	3.96E+05	2.93E-01	6.34E-01	5.98E-01	3.33E+01	-
f7	2.71E-01	6.75E+02	3.56E-01	3.58E-01	3.40E-01	3.74E-01	-
f8	6.23E-01	3.04E+06	6.73E-01	7.01E-01	8.35E-01	7.69E-01	-
f9	9.07E-02	1.66E+03	1.30E-01	3.32E+281	1.51E-01	3.24E+00	-
f10	6.47E-02	4.93E+04	1.14E-01	2.57E-01	1.94E-01	1.28E-01	-
f11	6.83E-01	4.57E+00	6.97E-01	1.35E+00	1.55E+00	1.18E+17	-
f12	7.09E-01	8.73E+00	8.69E-01	7.08E+01	7.50E+25	6.12E+00	-
f13	2.65E+02	3.10E+05	4.37E+02	2.25E+03	2.54E+03	3.08E+04	2.80E+02
f14	7.95E+00	3.22E+10	8.08E+00	8.76E+00	8.40E+00	1.14E+01	7.78E+00
f15	5.71E-01	3.24E+01	6.64E-01	7.62E-01	6.78E-01	6.93E-01	5.61E-01
f16	6.60E-01	3.44E+02	6.44E-01	7.16E-01	6.33E-01	6.43E-01	6.16E-01
f17	6.38E-01	1.33E+04	7.10E-01	7.97E-01	7.36E-01	7.44E-01	6.74E-01
f18	6.34E-01	6.60E+01	6.92E-01	7.77E-01	7.27E-01	7.15E-01	6.10E-01
f19	6.22E-01	1.31E+01	6.18E-01	7.39E-01	6.04E-01	7.03E-01	6.39E-01
f20	6.23E-01	3.74E+01	6.89E-01	7.86E-01	7.29E-01	7.24E-01	6.16E-01
f21	6.22E-01	3.63E+03	6.93E-01	7.94E-01	7.06E-01	7.06E-01	6.22E-01
f22	5.68E-01	1.58E+03	6.38E-01	7.69E-01	6.83E-01	6.80E-01	5.76E-01
f23	1.68E+00	1.94E+01	1.73E+00	3.52E+88	2.13E+00	2.04E+00	1.57E+00
f24	6.26E-01	7.11E+01	7.34E-01	8.01E-01	7.44E-01	7.54E-01	5.06E-01
f25	1.27E+01	4.65E+20	1.17E+01	1.27E+01	1.16E+01	1.23E+01	1.15E+01
f26	7.05E-01	2.92E+03	7.13E-01	7.49E-01	7.13E-01	7.08E-01	6.84E-01
f27	6.60E-01	1.45E+01	7.52E-01	8.30E-01	7.82E-01	7.54E-01	6.35E-01
f28	6.67E-01	5.81E+02	7.23E-01	7.80E-01	7.38E-01	7.41E-01	6.50E-01
f29	7.40E-01	9.69E+01	7.89E-01	1.19E+00	9.32E-01	8.87E+00	5.49E-01
f30	7.32E-01	3.65E+04	7.70E-01	8.24E-01	1.11E+00	5.85E+00	7.17E-01
f31	7.26E-01	2.11E+02	7.64E-01	1.04E+00	2.60E+01	8.51E-01	6.76E-01
f32	7.72E-01	1.35E+02	7.99E-01	6.63E+47	8.49E-01	8.26E-01	5.41E-01
f33	7.93E-01	6.07E+02	7.66E-01	7.81E+00	1.49E+00	1.09E+00	6.45E-01
f34	7.42E-01	4.26E+02	7.83E-01	7.45E-01	7.40E-01	7.50E-01	6.54E-01
f35	7.37E-01	1.25E+04	7.71E-01	1.06E+00	8.14E-01	8.62E-01	6.51E-01
f36	7.69E-01	6.06E+02	7.82E-01	7.36E-01	7.64E-01	7.51E-01	6.76E-01
f37	7.66E-01	3.36E+02	7.47E-01	1.43E+00	5.71E+02	8.33E-01	6.86E-01

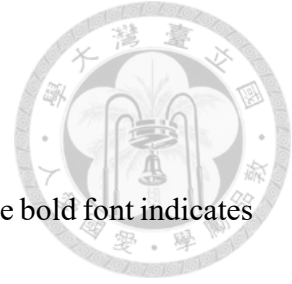


Table 5.10: Raw fitness of RBGP compared with RBGP+constant. The bold font indicates the lowest MAE.

	RBGP + constant	RBGP
f1	1.01E-01	1.19E-01
f2	1.60E-01	1.68E-01
f3	5.76E-01	5.76E-01
f4	6.02E-01	6.09E-01
f5	1.37E-01	1.32E-01
f6	2.44E-01	2.32E-01
f7	2.70E-01	2.86E-01
f8	6.49E-01	6.44E-01
f9	1.06E-01	9.50E-02
f10	6.07E-02	8.04E-02
f11	7.86E-01	7.98E-01
f12	8.33E-01	7.49E-01
f13	1.45E+02	4.98E+02
f14	7.77E+00	8.20E+00
f15	5.53E-01	6.36E-01
f16	6.13E-01	6.70E-01
f17	6.32E-01	6.86E-01
f18	6.26E-01	6.71E-01
f19	6.17E-01	6.82E-01
f20	6.39E-01	6.76E-01
f21	6.26E-01	6.62E-01
f22	5.74E-01	6.05E-01
f23	1.59E+00	1.70E+00
f24	5.68E-01	6.44E-01
f25	1.26E+01	1.41E+01
f26	6.73E-01	7.16E-01
f27	5.87E-01	6.96E-01
f28	6.24E-01	6.88E-01
f29	5.87E-01	7.34E-01
f30	6.46E-01	7.46E-01
f31	6.33E-01	7.54E-01
f32	6.15E-01	7.71E-01
f33	6.69E-01	7.48E-01
f34	7.15E-01	7.29E-01
f35	6.42E-01	7.64E-01
f36	7.06E-01	7.60E-01
f37	6.59E-01	7.54E-01



Table 5.12: Raw fitness of RBGP and SOTAs combined with Operon. The bold font indicates the lowest MAE.

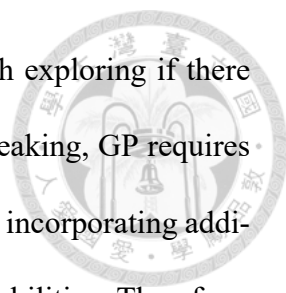
	RBGP	GP-GOMEA	gplearn	ellynGP	ELPEX	AFP	Operon
f1	1.11E-01	1.66E-01	1.42E-01	1.23E-01	1.30E-01	1.43E-01	1.20E-01
f2	1.44E-01	2.28E-01	2.14E-01	2.12E-01	1.97E-01	2.09E-01	1.62E-01
f3	4.75E-01	1.15E+00	5.35E-01	6.00E-01	5.74E-01	6.28E-01	2.20E-01
f4	6.64E-01	1.08E+00	8.64E-01	9.18E-01	9.43E-01	9.66E-01	4.13E-01
f5	1.36E-01	2.74E-01	2.29E-01	2.53E-01	2.86E-01	2.79E-01	1.23E-01
f6	1.83E-01	7.77E-01	4.93E-01	6.30E-01	7.27E-01	7.58E-01	1.66E-01
f7	2.66E-01	6.52E-01	3.46E-01	3.33E-01	3.34E-01	3.66E-01	2.84E-01
f8	6.42E-01	6.88E-01	6.80E-01	6.66E-01	6.64E-01	6.66E-01	6.60E-01
f9	9.70E-02	1.31E-01	1.46E-01	1.28E-01	1.23E-01	1.21E-01	1.01E-01
f10	3.64E-02	1.54E-01	7.49E-02	8.54E-02	9.03E-02	8.88E-02	4.13E-02
f11	1.34E-01	8.96E-01	7.71E-01	8.92E-01	7.20E-01	8.42E-01	1.93E-02
f12	2.71E-01	9.22E-01	8.27E-01	1.10E+00	1.01E+00	9.94E-01	1.55E-01
f13	3.50E+01	7.72E+01	7.68E+01	7.56E+01	7.64E+01	7.63E+01	2.43E+01
f14	7.72E+00	1.07E+01	8.06E+00	8.43E+00	8.54E+00	8.25E+00	7.51E+00
f15	5.59E-01	7.89E-01	6.47E-01	6.84E-01	6.83E-01	6.69E-01	5.27E-01
f16	6.00E-01	8.07E-01	6.16E-01	6.60E-01	6.58E-01	6.52E-01	3.77E-01
f17	6.21E-01	8.15E-01	7.08E-01	7.57E-01	7.29E-01	7.47E-01	6.80E-01
f18	6.47E-01	8.00E-01	7.03E-01	7.15E-01	7.23E-01	7.35E-01	6.78E-01
f19	5.93E-01	8.32E-01	6.11E-01	6.76E-01	6.82E-01	6.94E-01	4.13E-01
f20	6.19E-01	7.87E-01	6.93E-01	7.05E-01	6.99E-01	7.05E-01	6.89E-01
f21	6.27E-01	7.92E-01	7.11E-01	6.94E-01	7.00E-01	6.98E-01	6.61E-01
f22	5.70E-01	7.81E-01	6.33E-01	6.79E-01	6.64E-01	6.83E-01	6.13E-01
f23	1.47E+00	1.87E+00	1.75E+00	1.70E+00	1.72E+00	1.70E+00	1.26E+00
f24	5.29E-01	8.29E-01	7.63E-01	7.22E-01	7.26E-01	7.36E-01	6.35E-01
f25	1.19E+01	3.48E+01	1.19E+01	1.33E+01	1.25E+01	1.17E+01	1.12E+01
f26	6.77E-01	8.21E-01	6.84E-01	6.93E-01	7.04E-01	6.95E-01	4.33E-01
f27	6.11E-01	8.29E-01	7.43E-01	7.54E-01	7.48E-01	7.72E-01	7.14E-01
f28	6.50E-01	7.86E-01	7.09E-01	7.29E-01	7.26E-01	7.23E-01	6.67E-01
f29	6.90E-01	8.32E-01	8.06E-01	8.10E-01	8.10E-01	8.15E-01	7.22E-01
f30	7.06E-01	7.93E-01	7.70E-01	7.76E-01	7.71E-01	7.79E-01	7.20E-01
f31	7.19E-01	7.89E-01	7.72E-01	7.65E-01	7.74E-01	7.80E-01	7.09E-01
f32	7.14E-01	8.25E-01	8.20E-01	8.05E-01	8.05E-01	8.06E-01	7.25E-01
f33	7.17E-01	8.09E-01	7.90E-01	7.98E-01	7.86E-01	7.90E-01	7.46E-01
f34	7.15E-01	8.04E-01	7.26E-01	7.01E-01	6.91E-01	6.94E-01	5.56E-01
f35	7.13E-01	7.89E-01	7.71E-01	7.75E-01	7.80E-01	7.82E-01	7.24E-01
f36	7.19E-01	8.15E-01	7.28E-01	7.08E-01	7.09E-01	7.01E-01	5.62E-01
f37	7.08E-01	8.15E-01	7.85E-01	7.87E-01	7.95E-01	7.99E-01	7.35E-01



Chapter 6 Conclusion

This paper proposes a GP algorithm for symbolic regression that utilizes program syntax and semantic information to enhance evolutionary efficiency. For syntax information, we introduced the binding mechanism to protect common structures from being disrupted by crossover. For semantic information, we proposed the ranging mechanism to utilize the scaling factor between the inputs and outputs of substructures composed of functions, enabling the identification of suitable crossover targets. Furthermore, to automatically adjust the number of protected structures through binding, we proposed the binding adaptation mechanism, which dynamically adjusts parameters throughout generations. Additionally, we introduced a feature selection method that combines the MRMR approach to improve the stability of RBGP in high-dimensional problems.

Through preliminary experiments, it was observed that protecting common structures rather than directly adding them to the function set is advantageous. Additionally, the combination of ranging and SGP demonstrated good optimization capabilities in toy problems. On the PMLB dataset, RBGP, on average, outperformed other SOTAs in terms of MAE performance. Furthermore, the inclusion of binding adaptation maintained its advantage over other SOTAs, and it was observed that the initial setting of the number of protected structures had minimal impact on the results. The inclusion of feature selection yielded improvements on average, particularly in high-dimensional problems.




In terms of future work, regarding semantics, it would be worth exploring if there are better signals than ranging for symbolic regression. Generally speaking, GP requires signals other than fitness to aid in evolution. This paper validated that incorporating additional useful signals can significantly improve GP's optimization capabilities. Therefore, one direction for future research is to investigate how to identify and integrate such signals. On the syntax side, using other languages to describe programs, such as first-order languages, and protecting meaningful structures formed by these languages is also a potential research direction.


In summary, this paper introduces the ranging mechanism and the binding mechanism for symbolic regression GP. The ranging mechanism utilizes the range of program outputs to provide evolutionary signals beyond fitness, and the binding mechanism protects common structures. The proposed algorithm demonstrates superior optimization capabilities compared to other SOTAs on real-world test problems.




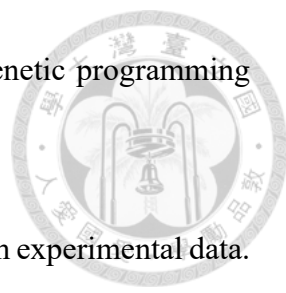
References

- [1] P. J. Angeline and J. Pollack. Evolutionary module acquisition. In *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 154–163, 1993.
- [2] A. Asuncion and D. Newman. UCI machine learning repository, 2007.
- [3] P. A. Bosman and D. Thierens. Linkage neighbors, optimal mixing and forced improvements in genetic algorithms. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO '12*, page 585–592, New York, NY, USA, 2012.
- [4] A. Bouter, T. Alderliesten, C. Witteveen, and P. A. Bosman. Exploiting linkage information in real-valued optimization with the real-valued gene-pool optimal mixing evolutionary algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 705–712, 2017.
- [5] B. Burlacu, G. Kronberger, and M. Kommenda. Operon c++ an efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 1562–1570, 2020.
- [6] P.-L. Chen, C.-J. Peng, C.-Y. Lu, and T.-L. Yu. Two-edge graphical linkage model for DSMGA-II. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17*, page 745–752, New York, NY, USA, 2017.

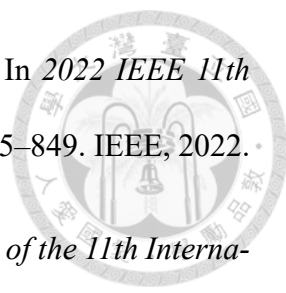
- 
- [7] V. V. De Melo. Kaizen programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 895–902, New York, NY, USA, 2014.
- [8] B. He, Q. Lu, Q. Yang, J. Luo, and Z. Wang. Taylor genetic programming for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 946–954, 2022.
- [9] T. Helmuth, L. Spector, and J. Matheson. Solving uncompromising problems with lexibase selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643, 2014.
- [10] E. Hemberg, K. Veeramachaneni, J. McDermott, C. Berzan, and U.-M. O'Reilly. An investigation of local patterns for estimation of distribution genetic programming. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pages 767–774, 2012.
- [11] S.-H. Hsu and T.-L. Yu. Optimization by pairwise linkage detection, incremental linkage set, and restricted / back mixing: DSMGA-II. GECCO '15, page 519–526, New York, NY, USA, 2015.
- [12] K. E. K. Jr. Evolving a sort: lessons in genetic programming. In *IEEE International Conference on Neural Networks*, pages 881–888 vol.2, 1993.
- [13] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
- [14] K. Krawiec. *Program synthesis*, pages 1–19. Springer International Publishing, Cham, 2016.

- 
- [15] W. La Cava, K. Danai, L. Spector, P. Fleming, A. Wright, and M. Lackner. Automatic identification of wind turbine models using evolutionary multiobjective optimization. *Renewable Energy*, 87:892–902, 2016.
- [16] W. La Cava, T. Helmuth, L. Spector, and K. Danai. Genetic programming with epigenetic local search. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, page 1055 – 1062, New York, NY, USA, 2015.
- [17] W. La Cava, P. Orzechowski, B. Burlacu, F. de Franca, M. Virgolin, Y. Jin, M. Kommenda, and J. Moore. Contemporary symbolic regression methods and their relative performance. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.
- [18] W. La Cava, P. Orzechowski, B. Burlacu, F. O. de França, M. Virgolin, Y. Jin, M. Kommenda, and J. H. Moore. Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351*, 2021.
- [19] W. La Cava, L. Spector, and K. Danai. Epsilon-lexicase selection for regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 741–748, 2016.
- [20] Q. Lu, J. Ren, and Z. Wang. Using genetic programming with prior formula knowledge to solve symbolic regression problem. *Computational Intelligence and Neuroscience*, 2016:1–1, 2016.
- [21] S. Luke and L. Panait. Lexicographic parsimony pressure. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, GECCO'02, page 829–836, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

- 
- [22] N. H. Luong, H. La Poutré, and P. A. Bosman. Multi-objective gene-pool optimal mixing evolutionary algorithms. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 357–364, 2014.
- [23] P. Monsieurs and E. Flerackers. Reducing bloat in genetic programming. In B. Reusch, editor, *Computational Intelligence. Theory and Applications*, pages 471–478, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- [24] 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*, pages 292–302, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [25] Q. U. Nguyen, M. O’Neill, N. Hoai, R. McKay, and E. López. Semantic similarity based crossover in gp: The case for real-valued function regression. pages 170–181, 01 2009.
- [26] U.-M. O’Reilly. Genetic programming II: Automatic discovery of reusable programs. *Artificial Life*, 1(4):439–441, 1994.
- [27] R. B. L. Richard P. Feynman and M. Sands. *The Feynman Lectures on Physics, Vol. I: The New Millennium Edition: Mainly Mechanics, Radiation, and Heat*. 2015.
- [28] J. D. Romano, T. T. Le, W. La Cava, J. T. Gregg, D. J. Goldberg, P. Chakraborty, N. L. Ray, D. Himmelstein, W. Fu, and J. H. Moore. PMLB v1.0: an open source dataset collection for benchmarking machine learning methods. *arXiv preprint arXiv:2012.00058v2*, 2021.
- [29] J. Rosca and D. Ballard. Genetic programming with adaptive representations. 03 1995.

- 
- [30] V. Sathia, V. Ganesh, and S. R. T. Nanditale. Accelerating genetic programming using gpus. *arXiv preprint arXiv:2110.11226*, 2021.
- [31] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.
- [32] M. D. Schmidt and H. Lipson. Age-fitness pareto optimization. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 543–544, 2010.
- [33] M. D. Schmidt and H. Lipson. Automated modeling of stochastic reactions with large measurement time-gaps. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 307–314, 2011.
- [34] G. F. Smits and M. Kotanchek. Pareto-front exploitation in symbolic regression. *Genetic Programming Theory and Practice II*, pages 283–299, 2005.
- [35] L. Spector. Assessment of problem modality by differential performance of lexibase selection in genetic programming: a preliminary report. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 401–408, 2012.
- [36] T. Stephens. Genetic programming in python, with a scikit-learn inspired API: gplearn, 2016.
- [37] S. H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*. Westview Press, 2000.
- [38] K. Tanemura, Y. Tachibana, Y. Tokuni, H. Manabe, and R. Miyadera. Application

of generic programming to unsolved mathematical problems. In *2022 IEEE 11th Global Conference on Consumer Electronics (GCCE)*, pages 845–849. IEEE, 2022.

- 
- [39] D. Thierens. The linkage tree genetic algorithm. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I, PPSN'10*, page 264–273, Berlin, Heidelberg, 2010. Springer-Verlag.
- [40] D. Thierens and P. A. Bosman. Optimal mixing evolutionary algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11*, page 617–624, New York, NY, USA, 2011.
- [41] D. Thierens and P. A. Bosman. Hierarchical problem solving with the linkage tree genetic algorithm. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, pages 877–884, 2013.
- [42] S.-M. Udrescu and M. Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16), 2020.
- [43] L. Vanneschi, M. Castelli, and S. Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO '10*, page 877–884, New York, NY, USA, 2010.
- [44] M. Virgolin, T. Alderliesten, C. Witteveen, and P. Bosman. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary Computation*, 29:1–27, 06 2020.
- [45] D. Whitley, V. S. Gordon, and K. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In *Parallel Problem Solving from Nature—PPSN*

III: International Conference on Evolutionary Computation The Third Conference on Parallel Problem Solving from Nature Jerusalem, Israel, October 9–14, 1994. Proceedings 3, pages 5–15. Springer, 1994.



- [46] H. Zhang, A. Zhou, H. Qian, and H. Zhang. Ps-tree: A piecewise symbolic regression tree. *Swarm and Evolutionary Computation*, 71:101061, 2022.
- [47] Z. Zhao, R. Anand, and M. Wang. Maximum relevance and minimum redundancy feature selection methods for a marketing machine learning platform. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 442–452. IEEE, 2019.
- [48] J. Zhong, L. Feng, W. Cai, and Y. Ong. Multifactorial genetic programming for symbolic regression problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP:1–14, 07 2018.





Appendix A — Introduction to MRMR and Operon

A.1 Maximum Relevance and Minimum Redundancy Feature Selection

MRMR [47] is a feature selection technique aimed to remove redundant information, thereby enhancing the relevance of the selected features for the trained model and reducing training costs. MRMR achieves this by maximizing the relevance between the input and output while minimizing the redundancy among the input features during feature selection. The relevance used in MRMR is defined as follows:

$$R(X_i) = I(X_i, Y), \quad (\text{A.1})$$

where X is input, Y is output, and X_i is a given feature. $I(\cdot, \cdot)$ is the mutual information function. The redundancy is defined as follows:

$$D(X_i) = \frac{1}{|S|} \sum_{X_s \in S} I(X_i, X_s), \quad (\text{A.2})$$

where S is the selected feature set. As MRMR aims to maximize relevance and minimize redundancy, the importance of a feature is determined by the following equation:

$$Importance(X_i) = R(X_i) - D(X_i). \quad (A.3)$$

By utilizing the above definition, MRMR can effectively rank the importance of features and serve as a basis for feature selection.

A.2 Operon

Operon [5] is a GP framework that combines machine learning with SGP. In each generation, it utilizes local search to find the parameters of a linear model with the GP tree as input and the target values as output. Specifically, it aims to find the values of the two parameters, Scale and Offset, in the expression $Scale * Output_{GP} + Offset$ using non-linear least squares fitting. In operon, the trees in each generation gradually become deeper due to the inclusion of two additional function layers, "*" and "+". This prevents the integration of fixed-length methods such as ellynGP and GP-GOMEA. To verify if incorporating these methods into the operon local search can lead to improvements, we propose a one-layer local search approach. In this approach, the "Scale" and "Offset" obtained from the fitting process in each generation are not involved in the evolutionary process. Algorithm 10 describes the one-layer local search framework.



Algorithm 10: One-layer operon local search

Input: GP: Other GP algorithm. X : Input; y : Output;

Output: Best program

```
1 Initialize GP.
2 while  $\neg$  ShouldTerminate do
3   GP evolve one generation.
4   for  $P_i \in$  Population of GP do
5     Scale, Offset  $\leftarrow$  non-linearLeastSquaresFitting( $P_i, X, y$ );
6      $P_i$ .updateFitness(Scale *  $P_i(X)$  + Offset,  $y$ );
7   end
8 end
9 return Best program in  $P$  with its Scale and Offset ;
```
