Adaptive Transfer of Genetic Knowledge in Evolutionary Optimization and Program Synthesis

March 2023

Yifan He

Adaptive Transfer of Genetic Knowledge in Evolutionary Optimization and Program Synthesis

Graduate School of Science and Technology

Degree Programs in Systems and Information Engineering

University of Tsukuba

March 2023

Yifan He

Abstract

A characteristic of human learning is the ability to obtain knowledge from tasks solved in the past and apply the obtained knowledge when solving tasks in the future. The learning behavior consecutively happens when a human solves endless and distinct tasks during his/her lifetime. This learning ability allows humans to solve more complex tasks.

Evolutionary Algorithms (EAs) are a group of methods that solve difficult optimization problems. Traditionally, the performance of an EA is not influenced by the tasks it has solved. In this dissertation, we discuss how to enhance an EA with learning ability across many distinct tasks.

We first provide a definition of Genetic Knowledge Transfer (GKT) which refers to the process where an EA is affected by the dynamics of another EA. Then, we propose an Adaptive Genetic Knowledge Transfer (AGKT) system that allows the adaptive transfer of the genetic knowledge obtained from many distinct past tasks. The proposed method extracts sub-solutions from the solution to the solved tasks and stores them in an archive. When solving a new task, the system selects proper sub-solutions to reuse based on the similarity approach and the trial-and-error approach.

We show two case studies that are related to GKT. The first case study aims to solve a Multi-Objective Optimization Problem in Geosciences called Seismic History Matching (SHM). We propose a GKT method using Lexicase Selection (LS) to solve the SHM problems. The proposed algorithm is tested on two SHM problems and compared with NSGA-II and RVEA. The results show that this method achieves a better optimization performance and a concentrated final solution set in the center of the Pareto Front, with fewer extreme solutions which would possibly be non-physical.

Our second case study introduces a type of problem where a Genetic Programming (GP) algorithm is required to solve a sequence of Program Synthesis (PS) tasks. When solving a new task, GP uses the knowledge learned from previously solved tasks. This problem is called Knowledge-Driven Program Synthesis (KDPS) problem. To solve the KDPS problems, we propose a method based on PushGP to solve PS tasks consecutively, extract subprograms from the solutions, and utilize subprograms in the next task. The proposed method achieves a better success rate when solving a sequence of PS problems, compared to the conventional PushGP algorithm.

In conclusion, this dissertation defines GKT as a base for the learning ability of EAs. We further develop the AGKT framework to enable effective GKT with EAs during sequential task-solving. Finally, we present the initial implementation of AGKT with GP to show the system's viability.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Main Contributions

Evolutionary Computation (EC) is a group of population-based search algorithms that are frequently used for difficult optimization problems. The recent EC community has shown an increasing interest in optimization problems with multiple objectives or tasks, such as Multi-Objective Optimization (MOO) [1] and Multi-Task Optimization (MTO) [2]. These problems with multiple tasks represent a majority of cases of the practical applications of optimization and are hard to solve.

Similarly, humans solve a lot of tasks during their whole lives. Interestingly, humans transfer the knowledge learned from one task to another. This Knowledge Transfer (KT) helps solve the current task if this task is similar to those tasks that have been solved in the past. KT techniques have been frequently applied in our real life since the tasks seldom appear in isolation in the real world. For instance, we learn knowledge about biological evolution and apply it to create EC for solving engineering optimization problems.

However, KT is not a new word in the EC literature. KT has been used in many studies in the field of MTO [3, 4] and Genetic Programming (GP) [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. Most of these studies have proposed to migrate good individuals or building blocks from an Evolutionary Algorithm (EA) to another to improve the performance of the latter EA. However, few of them provide a clear definition of their KT. In addition, most of the prior works have focused their attention on the transfer among a few similar tasks.

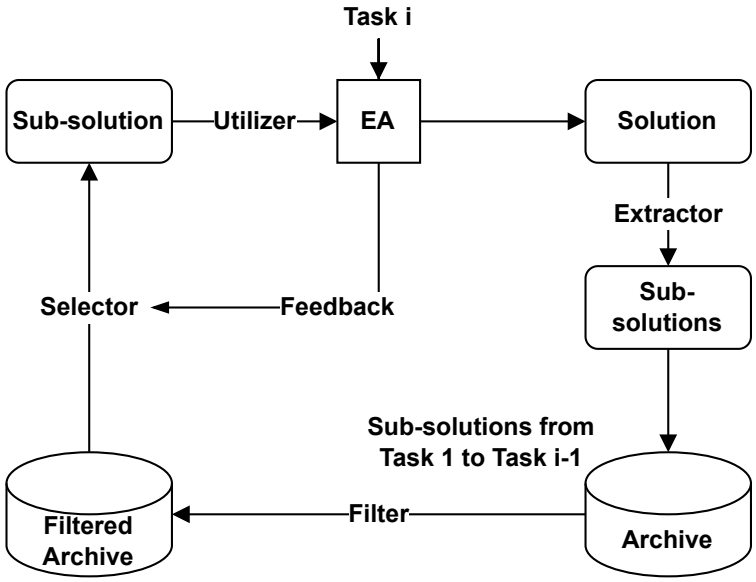

Figure 1.1: Quick view of the main contribution: the Adaptive Genetic Transfer system. The details of this system are introduced in **Chapter 4**.

Adaptive Genetic Knowledge Transfer This dissertation first provides a clear definition of Genetic Knowledge Transfer (GKT) shown in **Figure 1.1** which refers to the process where an EA is affected by the dynamics of another EA. After that, the dissertation describes a scenario where an EA is run on a cloud service. Users consecutively pose optimization tasks to this solver. After a long time, the EA on the cloud is then required to solve an endless sequence of distinct tasks, just like a human. It is then very natural to come up with the following question: *can an EA solve a sequence of many distinct tasks and improve itself through the sequential problem-solving by using the idea of GKT?*

One vital difference between this study and prior works on GKT is the larger number of tasks to solve and the dissimilarity between these tasks. Therefore, simple methods such as migrating the best individuals or reusing final populations might not work well, since there are a lot of tasks that are not related to the current one.

Therefore, this study proposes an Adaptive Genetic Knowledge Transfer (AGKT) system that allows the adaptive transfer of the genetic knowledge obtained from many distinct past tasks. The proposed method extracts sub-solutions from the solution to the solved tasks and stores them in an archive.

These sub-solutions are processed genetic knowledge. When solving a new task, the system selects proper sub-solutions based on two approaches: 1) similarity-based approach and 2) trial and error approach. The similarity approach refers to a filter that creates a subset of the archive to include sub-solutions from the tasks with high similarity to the current task. However, this approach requires a measure of similarity between tasks. The trial and error approach selects sub-solutions from the archive and uses them with the EA. The probability of selecting a sub-solution is then updated based on the feedback from the EA.

With this AGKT system, the EA can solve harder tasks with the sub-solutions extracted from simple tasks. Moreover, this system might be able to discover some unexpected helper tasks to improve performance.

Multi-Criteria Seismic History Matching This dissertation provides two case studies that are related to the topic of GKT. The first case study aims to solve an optimization task called Seismic History Matching (SHM) [17].

SHM is a key problem in Geosciences that aims to find a mathematical model of the subsurface to match the simulation data with the real records. In SHM, multiple types of data are used and thus SHM is a problem with multiple objective functions where each objective is to minimize the difference between a type of simulation and real data.

This problem is usually modeled as MOO, which focuses on the trade-off between the objectives and is solved by Multi-Objective Evolutionary Algorithms (MOEAs). However, there is only one true model of a specific subsurface in interest. This true model should minimize all the objectives at the same time. In other words, a model that minimizes some objectives but performs worse on others is possibly unphysical. Therefore, a "trade-off" relationship might not be a proper assumption in the SHM problems. As a result, the improper use of MOEAs on this problem may waste the computational budget to search some unpromising areas such as extreme points.

On the contrary, GKT might be able to apply to this problem since the objectives share the same optimal solution and are from the measure of the same physics. The genetic knowledge from solving one of the objectives in the SHM problem should help to solve the optimization of another objective.

This study proposes a GKT method [18] using Lexicase Selection (LS) [19]

on the SHM problems. LS filters the solutions based on a shuffled arrangement of all objectives so that it can drive the solutions to a better quality in all objectives at the same time while providing enough diversity during the search process.

The proposed algorithm [18] is tested on two SHM problems and compared with two well-known MOEAs, NSGA-II [20] and RVEA [21]. The results are discussed from various perspectives, including the distance to the ground truth, the difference in set coverage, the distribution of the non-dominated solutions, as well as the prediction performance. Our experiments show that this method achieves a better optimization performance and a concentrated final solution set in the center of the Pareto Front, with fewer extreme solutions which would possibly be non-physical.

Knowledge-Driven Program Synthesis Human programmers can learn from the programming problems that they have solved before and apply what they have learned to solve upcoming problems. This ability of learning is based on the relationship between problems and is vital for generating complex programs. Therefore, this second case study introduces a type of task where an agent is required to solve a sequence of Program Synthesis (PS) problems. When solving a new problem, the agent should use the knowledge learned from previously solved problems. This task is called the Knowledge-Driven Program Synthesis (KDPS) problem [22] and is a practice of the research question that we proposed in the previous paragraph.

To solve the KDPS problems, this study proposes a method [22] based on PushGP [23] to consecutively solve programming tasks, extract genetic knowledge from the solutions, and utilize genetic knowledge in the next problem. This proposed method is an application of our AGKT system in the PS field.

Our proposed method [22] uses subprograms, the sub-sequences of Push instructions, as processed genetic knowledge. The subprograms hold partial information about the original program. Moreover, any sequence of Push instructions is valid to run. Therefore, the proposed method can easily take a subprogram and use it to mutate a different program. Even Partitioning (EP) is proposed to extract subprograms from a solution. EP simply divides a solution into several partitions with equal lengths and stores them in an archive for later use. Adaptive Replacement Mutation (ARM) takes a piece of subprogram in the archive by probability and replaces a part of

the instructions of the parent solution with this subprogram. The probability to select subprograms is determined by the feedback of the replacement mutation in a self-adaptive manner. Our proposed method achieves a better success rate when solving a sequence of PS problems, compared to the conventional PushGP algorithm.

1.2 Structure of the Dissertation

This dissertation is organized as follows.

Chapter 2 introduces EC with multiple tasks, namely MOO, MTO, and GP. For every category of EC, we provide general background on the problem formulation and the frequently used algorithms to solve this problem.

Chapter 3 provides a clear definition of GKT. Several examples of GKT in the EC literature are used for easy understanding, including GKT in MOO, GKT in MTO, and GKT in GP.

Chapter 4 first illustrates a future scenario where an EA is hosted on a cloud service to solve a sequence of many distinct tasks from the users. This chapter then comes up with the research question of how an EA improves itself during solving this sequence of tasks. As a solution, the overall design of the AGKT system is proposed and the possible design of the components is explained.

Chapter 5 is a case study of GKT for solving SHM problems. The chapter provides sufficient background on the SHM problem and the related works on solving SHM with MOEAs. The limitation of the prior methods is then discussed. Following the limitation of the prior studies, we propose a method using LS to solve the SHM problems. We provide an explanation of how LS performs GKT implicitly. After that, the simulation results are presented in the later sections to show the superiority of our proposed method.

Chapter 6 is a practice of our conceptual design of the AGKT system for solving PS problems. We propose a variant of the traditional PS problem where a GP algorithm is asked to solve a sequence of PS tasks using the idea of GKT. We demonstrate a method that is an implementation of the AGKT system with GP. This method uses subprograms as genetic knowledge and transfers them properly based on an adaptive selection method. The proposed method is tested to solve a sequence of six PS problems.

Chapter 7 summarizes the whole dissertation and points out several future routines.

1.3 List of Publications

Journals

Yifan He, Claus Aranha, Antony Hallam, and Romain Chassagne. Optimization of subsurface models with multiple criteria using lexicase selection. *Operations Research Perspectives*, 9:100237, 2022.

Antony Hallam, Romain Chassagne, Claus Aranha, and Yifan He. Comparison of map metrics as fitness input for assisted seismic history matching. *Journal of Geophysics and Engineering*, 19(3):457–474, 2022.

Yifan He and Claus Aranha. Solving portfolio optimization problems using moea/d and lévy flight. *Advances in Data Science and Adaptive Analysis*, 12(03n04):2050005, 2020.

Peer-Reviewed International Conferences

Yifan He, Claus Aranha, and Tetsuya Sakurai. Knowledge-driven program synthesis via adaptive replacement mutation and auto-constructed subprogram archives. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 14–21. IEEE, 2022.

Yifan He, Claus Aranha, and Tetsuya Sakurai. Incorporating sub-programs as knowledge in program synthesis by pushgp and adaptive replacement mutation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 554–557, 2022.

Yifan He, Claus Aranha, and Tetsuya Sakurai. Parameter evolution self-adaptive strategy and its application for cuckoo search. In *International Conference on Bioinspired Methods and Their Applications*, pages 56-68. Springer, 2020.

Non Peer-Reviewed Conferences

Yifan He and Claus Aranha. Solving multi-objective optimization problems with differential evolution and lexicase selection. In *19th Symposium of the Japanese Society of Evolutionary Computation*, Online, 2021.

Yifan He and Claus Aranha. Evolving stability parameters of lévy flight in cuckoo search. In *17th Symposium of the Japanese Society of Evolutionary Computation*, Tokyo, 2020.

1.4 List of Abbreviations

AAMTEA ....Adaptive Archive-based Many-Task Evolutionary Algorithm
ADF ..... Automatically Defined Function
AGKT ..... Adaptive Genetic Knowledge Transfer
ARM ..... Adaptive Replacement Mutation
CSL ..... Compare String Lengths
EA ..... Evolutionary Algorithm
EC ..... Evolutionary Computation
EP ..... Even Partitioning
GKT .....Genetic Knowledge Transfer
GP .....Genetic Programming
GPSB .....General Program Synthesis Benchmark
KDPS .....Knowledge-Driven Program Synthesis
KT .....Knowledge Transfer
LS .....Lexicase Selection
MD .....Median
MFEA .....Multi-Factorial Evolutionary Algorithm
MMTEF .....Multi-population Muti-Task Evolutionary Framework
MOEA .....Multi-Objective Evolutionary Algorithm
MOEA/D ..... MOEA based on Decomposition
MOO .....Multi-Objective Optimization
MSL ..... Median String Length
MTEA .....Multi-Task Evolutionary Algorithm
MTO .....Multi-Task Optimization
NGKT .....Naive Genetic Knowledge Transfer
NSGA-II ..... Non-dominated Sorting Genetic Algorithm II
OP .....Optimization Problem
PF ..... Pareto Front
PS .....Program Synthesis
RM .....Replacement Mutation
SL .....Small or Large
SLM .....Small or Large Median
SLS .....Small or Large String

Chapter 2

Evolutionary Computation with Multiple Tasks

In this chapter, we review important concepts necessary for the development of the thesis. Optimization with multiple tasks represents a majority of the application in the real world and thus is essential for many fields such as engineering, management science, and scientific research.

We first introduce Evolutionary Computation (**Section 2.1**). After that, we go through two categories of optimization problems that contain more than one objective function, namely Multi-Objective Optimization (**Section 2.2**) and Multi-Task Optimization (**Section 2.3**). Then, we review Genetic Programming, an algorithm proposed to generate computer programs (**Section 2.4**). We provide some cases of how this algorithm solves multiple tasks at the same time.

Despite the problems and the algorithms in the above paragraph, there are other problem formulations considering multiple tasks, such as Multi-Level Optimization. Readers interested in this topic can refer to the review work by Gupta and Ong [24] for further details.

2.1 Evolutionary Computation

An Optimization Problem (OP) is the problem to find the best solution in a set of feasible solutions. We model real-world problems as OP in many fields such as engineering, management science, finance, and scientific research. A minimization problem is usually formulated as in (2.1). f is the objective function and $\mathbf{x}$ is a solution candidate containing n decision variables. g_i and

Algorithm 2.1: The typical procedure of Evolutionary Computation

```
X ← initialize();
repeat
    P ← select(X);
    X ← update(P);
until termination criteria are satisfied;
```

h_j are the constraints of the problem.

$$\begin{aligned} minimize \quad & f(\mathbf{x}) \\ s.t. \quad & \mathbf{x} = (x_1, \ldots, x_n) \\ & g_i(\mathbf{x}) \leq 0, i = 1, \ldots, p \\ & h_j(\mathbf{x}) = 0, j = 1, \ldots, q \end{aligned} \tag{2.1}$$

Evolutionary Computation (EC) or Evolutionary Algorithms (EAs) are a group of algorithms to solve OPs by mimicking the process of natural evolution and the population behavior of creatures. It uses the intuition of how creatures evolve to adapt to the environment and how creatures solve a task based on self-organization. In general, EC starts with a random population of solution candidates. Then, EC consecutively selects candidates with high quality and updates them until the optimal solution to the problem is found. This typical procedure of EC is shown as in **Algorithm 2.1**. In this procedure and other EA descriptions in this dissertation, we omit the step that evaluates the fitness values of the solutions.

Most of the EAs are gradient-free and thus could be applied to solve discrete or black-box OPs. Moreover, the use of global search operators allows EC to escape from the local optima during optimization. With these advantages, many EAs (e.g., Genetic Algorithm [25], Particle Swarm Optimization [26], and Differential Evolution [27]) have been proposed to solve a variety of difficult OPs. These problems vary from continuous function optimization to real-world applications, such as optimizing the structure design of cars [28] and estimating the mathematical model of the subsurface [17].

Additionally, EC allows us to study OPs with more advanced formulations. In this dissertation, we focus on the OPs with multiple tasks or objective functions that are solved simultaneously. Compared to OPs with only one task, these problems represent a major number of applications; however, they are harder to solve. The rest of this chapter introduces three

scenarios where EC deals with more than one task, namely Multi-Objective Optimization (**Section 2.2**), Multi-Task Optimization (**Section 2.3**), and Genetic Programming (**Section 2.4**).

2.2 Multi-Objective Optimization

2.2.1 Problem Description

Multi-Objective Optimization (MOO) [1] might be the most well-known problem formulation that optimizes multiple tasks. A MOO problem contains several objectives that are trade-offs with each other. In other words, the optimal solution to these objectives cannot be achieved at the same time. A minimization MOO problem in (2.2) minimizes all m objectives at the same time (Ω is the feasible region determined by constraints).

$$\begin{aligned} minimize \quad & \mathbf{f}(\mathbf{x}) \\ s.t. \quad & \mathbf{f} = (f_1, \ldots, f_m) \\ & \mathbf{x} = (x_1, \ldots, x_n) \in \Omega \end{aligned} \tag{2.2}$$

Domination defines "betterness" under multiple criteria and is one of the key concepts in MOO. A solution $\mathbf{x}$ dominates another solution $\mathbf{y}$ if and only if $\mathbf{x}$ is non-worse than $\mathbf{y}$ on all the objectives, and $\mathbf{x}$ is strictly better than $\mathbf{y}$ on at least one of the objectives. (2.3) provides a mathematical definition of domination in the minimization MOO problem, where the symbol "$\prec$" means "dominate".

$$\begin{aligned} \mathbf{f}(\mathbf{x}) \prec \mathbf{f}(\mathbf{y}) \iff & \forall i \in \{1, \ldots, m\}, f_i(\mathbf{x}) \leq f_i(\mathbf{y}) \\ & \exists i \in \{1, \ldots, m\}, f_i(\mathbf{x}) < f_i(\mathbf{y}) \end{aligned} \tag{2.3}$$

After we define domination as the "betterness" in MOO, it is not hard to realize that the optimal solutions in MOO are the solutions that are not dominated by any other solutions or the non-dominated solutions of the feasible region. We call these trade-off solutions Pareto Front (PF).

These important concepts are illustrated in **Figure 2.1**. f_1 and f_2 are minimization objectives. The circle area shows the feasible region of this problem. The three points A, B, and C are the objective vectors of three solutions. A dominates B but does not dominate C (since C holds a better

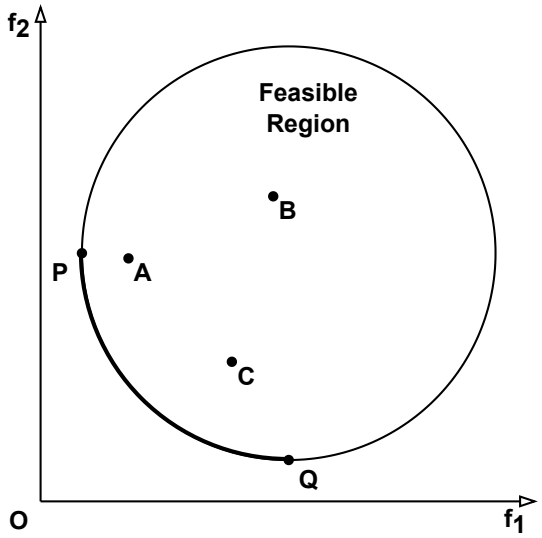

Figure 2.1: Important concepts in Multi-Objective Optimization. f_1 and f_2 are minimization objectives. A dominates B but does not dominate C. The curve PQ is the Pareto Front of this problem. P and Q are extreme points and are the optimal solutions of f_1 and f_2, respectively.

value on f_2 compared to A). The curve PQ is the PF of this MOO problem. P and Q are extreme points and are the optimal solutions of f_1 and f_2, respectively.

The MOO problem formulation represents a large group of OPs in the real world. We take the Portfolio Optimization (PO) [29] in the financial engineering field as an example. The goal of PO is to find a combination of available assets (e.g., stocks) to get the maximum return with the minimum risk. However, higher return often comes together with higher risk. This investment characteristic leads to the trade-off between the two objectives in PO, and therefore PO is an application of MOO.

The traditional approach to deal with multiple objectives is to do a weighted sum scalarization to the objectives. However, a portfolio manager cannot know the preference between return and risk until a specific user comes. In fact, even the user himself/herself cannot assign accurate weights to the two objectives. MOO addresses these issues by retrieving a set of trade-off solutions to represent the entire PF. The user will then look at the simulation results of all solutions to make a further decision. Therefore, MOO is more objective and accurate compared to the traditional weighted sum approach.

In spite of PO, the MOO formulation has been applied in many fields, including but not limited to engineering [28], finance [30], management science [31], and transportation [32].

2.2.2 Multi-Objective Evolutionary Algorithms

Multi-Objective Evolutionary Algorithms (MOEAs) are the EAs that solve MOO problems. The selection operators in MOEAs are designed to interact with multiple objectives. These special designs could be classified into three categories: 1) indicator-based approach, 2) domination-based approach, and 3) decomposition-based approach.

The indicator-based MOEAs define their selection mechanism based on quality indicators that assess the performance of the solution set, such as Hypervolume [33] and Invert Generation Distance [34]. These MOEAs optimize the indicator value during the evolutionary processes. Readers who are interested in this category of algorithms could refer to a recent survey [35].

The domination-based MOEAs use the definition of domination to select parent candidates. The most representative Non-dominated Sorting Genetic Algorithm II (NSGA-II) [20] uses fast non-dominated sorting to rank the individuals. This sorting method assigns Rank 1 to the non-dominated solutions of the population. After that, it consecutively excludes the ranked solutions and assigns an increased rank to the non-dominated solutions of the rest individuals. Despite the domination-based ranking, this type of MOEA usually contains an operator to maintain the diversity of the population to cover the entire PF (e.g., crowding distance assignment in NSGA-II [20]).

The decomposition-based MOEAs transform the original MOO problem into several single-objective sub-problems and optimize them simultaneously. Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) [36] is the first proposed algorithm in this group. In **Section 3.2**, we will discuss the Genetic Knowledge Transfer in decomposition-based MOEAs, with the example of MOEA/D. Therefore, the next section provides a detailed introduction and the pseudocode of MOEA/D [36] in **Algorithm 2.2**.

2.2.3 Multi-Objective Evolutionary Algorithm based on Decomposition

MOEA/D [36] uses a set of weight vectors to transform a MOO problem into a set of single-objective problems. This step is called decomposition. To illustrate the idea of decomposition, we first look at the most simple decomposition method, the weighted sum approach.

For example, we have a bi-objective MOO problem where $\mathbf{f} = (f_1, f_2)$ and we are given with a set of weight vectors $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_6\} = \{(\frac{i}{5}, 1 - \frac{i}{5})\}_{i=0}^{5}$. Based on the weighted sum approach, we can decompose $\mathbf{f}$ into six single-objective optimization problems as in (2.4). The optimal solutions to these sub-problems form an approximated PF of the original MOO problem.

$$\begin{aligned}
g_1(\mathbf{x}) &= 0.0 \cdot f_1(\mathbf{x}) + 1.0 \cdot f_2(\mathbf{x}) \\
g_2(\mathbf{x}) &= 0.2 \cdot f_1(\mathbf{x}) + 0.8 \cdot f_2(\mathbf{x}) \\
g_3(\mathbf{x}) &= 0.4 \cdot f_1(\mathbf{x}) + 0.6 \cdot f_2(\mathbf{x}) \\
g_4(\mathbf{x}) &= 0.6 \cdot f_1(\mathbf{x}) + 0.4 \cdot f_2(\mathbf{x}) \\
g_5(\mathbf{x}) &= 0.8 \cdot f_1(\mathbf{x}) + 0.2 \cdot f_2(\mathbf{x}) \\
g_6(\mathbf{x}) &= 1.0 \cdot f_1(\mathbf{x}) + 0.0 \cdot f_2(\mathbf{x})
\end{aligned} \tag{2.4}$$

In MOEA/D [36], these sub-problems are optimized simultaneously in a single algorithm run. Every individual in the population of MOEA/D corresponds to one of the sub-problems. That is, the population size of MOEA/D is the same as the number of the weight vectors and the sub-problems. In the previous example, a population including six individuals $\mathbf{x}_1, \ldots, \mathbf{x}_6$ is used, and $\mathbf{x}_i$ is optimized as the solution of g_i in (2.4).

Algorithm 2.2 presents the standard procedure of MOEA/D [36]. The weight vector set V is provided by users or generated by algorithms such as the Das-Dennis method [37]. The dimension of the weight vectors is equal to the number of objective functions. $\mathcal{B}$ represents a map from the index of a solution to the indexes of its neighbor solutions.

To perform the decomposition step, this standard procedure employs the Tchebycheff decomposition approach in (2.5). This approach requires a weight vector $\mathbf{v}$ and a reference point $\mathbf{z}^*$ to create a sub-problem. For an m-objective minimization MOO problem, the component z_k^* is the minimum objective value of f_k in the population. With a set of weight vectors, the Tchebycheff decomposition approach transforms the original problem into several sub-problems, to which the solutions are the cross points between the PF and the lines determined by the weight vectors and the reference point. **Figure 2.2** illustrates an example of Tchebycheff decomposition with six weight vectors.

$$\begin{aligned} minimize \quad & g^{tch}(\mathbf{x} \mid \mathbf{v}, \mathbf{z}^*) = \max_{k=1,\ldots,m} \{v_k | f_k(\mathbf{x}) - z_k^*|\} \\ s.t. \quad & \mathbf{x} = (x_1, \ldots, x_n) \\ & \mathbf{v} = (v_1, \ldots, v_m) \\ & \mathbf{z}^* = (z_1^*, \ldots, z_m^*) \\ & z_k^* = \min_{\mathbf{x}_l \in X} f_k(\mathbf{x}_l) \end{aligned} \tag{2.5}$$

To generate the child of $\mathbf{x}_i$, MOEA/D [36] randomly select two parents from the neighbor solutions of $\mathbf{x}_i$. The standard reproduction step uses simulated binary crossover [38, 39] and polynomial mutation [39]. After that, MOEA/D [36] updates the reference point $\mathbf{z}^*$ if the child $\mathbf{y}$ holds a smaller objective value in any of the objectives. This child $\mathbf{y}$ is then compared with all the neighbor solutions of $\mathbf{x}_i$ based on the Tchebycheff decomposition cost g^{tch}. The neighbor solution is updated to $\mathbf{y}$ if it is worse than $\mathbf{y}$ in terms of the decomposition cost (Line 8 and Line 9).

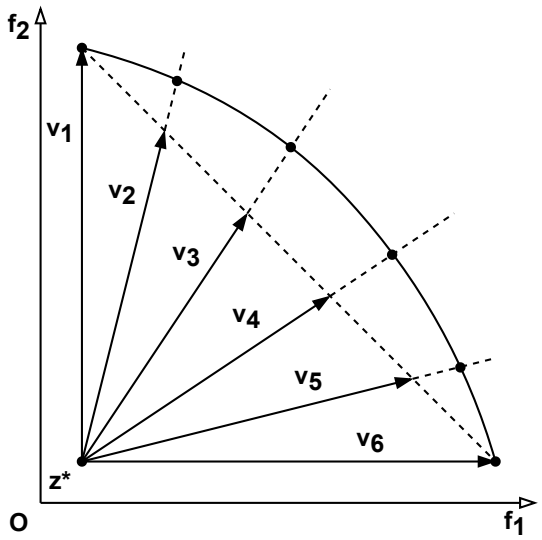

Figure 2.2: Tchebycheff decomposition with multiple weight vectors. The curve is the PF of a minimization MOO problem. $\mathbf{v}_1$ to $\mathbf{v}_6$ are six weight vectors and $\mathbf{z}^*$ is the reference point. The Tchebycheff decomposition transforms the original problem into six sub-problems, to which the solutions are the cross points between the PF and the lines determined by $\{\mathbf{v}_1, \ldots, \mathbf{v}_6\}$ and $\mathbf{z}^*$.

Algorithm 2.2: The pseudocode of Multi-Objective Evolutionary Algorithm based on Decomposition

```
   input  : weight vectors V = {v_1, ..., v_N}, neighborhood relation B
   output: final population X
 1 X ← initialize();
 2 repeat
 3     foreach x_i in X do
 4         a, b ← random-select(B(i));
 5         y ← reproduce(x_a, x_b);
 6         z* ← update-reference-point(z*, f(y));
 7         foreach j in B(i) do
 8             if g^tch(y | v_j, z*) ≤ g^tch(x_j | v_j, z*) then
 9                 x_j ← y;
10 until termination criteria are satisfied;
11 return X;
```

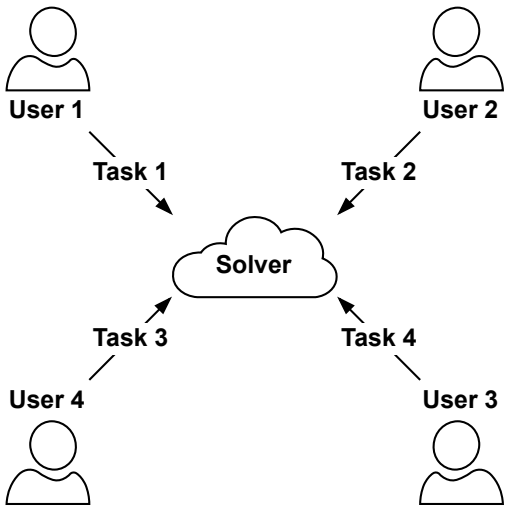

Figure 2.3: A cloud service solves multiple tasks in parallel. These tasks are from different users but at the same time. This scenario is considered a typical application of Multi-Task Optimization.

2.3 Multi-Task Optimization

2.3.1 Problem Description

The practical motivation for Multi-Task Optimization (MTO) [2] has derived from the scenario in **Figure 2.3** that a cloud service provides users with access to a solver. Naturally, this solver deals with multiple optimization tasks received from multiple users at the same time. These tasks can belong to either similar or entirely different domains. In MTO [2], multiple tasks are solved concurrently, exploiting the common knowledge between them.

An illustrative example of the tasks in MTO [2] could be shopping discount maximization under a limited budget and portfolio return maximization under limited risk. The first problem tries to maximize the discount value (i.e., the amount of saved money) considering a set of available items and a maximum budget. This problem is an Integer Knapsack Problem. Meanwhile, the second problem aims to find a combination of investment ratios and is a practice of the Fractional Knapsack Problem. Certainly, the

two tasks are in different domains; however, we can solve them in parallel since both of them belong to the larger category of the Knapsack Problem.

The MTO [2] with m tasks could be formulated as follows. In MTO, there are m tasks $(T_1, \ldots, T_m)$ in total. The objective functions of these tasks are $f_1, \ldots, f_m$. $\mathbf{x}^k$ represents the solution of the k-th task. It contains n_k decision variables and the corresponding feasible region is denoted as Ω_k. The optimal solution to the k-th task is $\mathbf{x}_*^k$ and the optimal solution set of MTO consists of the optimal solutions for every task. Unlike MOO (**Section 2.2**), MTO does not focus on the trade-off between tasks.

$$\begin{aligned} minimize \quad & f_1(\mathbf{x}^1), \ldots, f_m(\mathbf{x}^m) \\ s.t. \quad & \mathbf{x}^k = (x_1^k, \ldots, x_{n_k}^k) \in \Omega_k \\ & k = 1, \ldots, m \end{aligned} \tag{2.6}$$

Many studies have worked on MTO applications in the real world. Wei *et al.* categorize these works into the following four types [4].

- The first type is using easy tasks to help with complex tasks. For instance, a study [40] has solved an Even Parity Problem by incorporating its smaller variants.
- The second type converts bi-level OPs into MTO [41]. A bi-level OP is a problem where a lower-level task is embedded in another upper-level task and the upper-level task takes the optimal solution of the lower-level task as a part of the problem formulation. Therefore, an inaccurate solution to the lower-level task may cause a misleading estimation of the upper-level objective function. The MTO allows the parallel solving of the two levels of the tasks, considering the connection between them.
- In the third type of MTO, the multiple tasks share similar decision variables or structures, such as optimizing the parameters of a neural network for distinct deep learning tasks [42] and solving several image feature learning tasks with Genetic Programming [43].
- The last type of applications groups the tasks with similar properties. The review paper by Wei *et al.* [4] has listed a group of studies in this category, such as solving the car structure design problem [28] as MTO [44] where cars of distinct types are believed to share some commonalities.

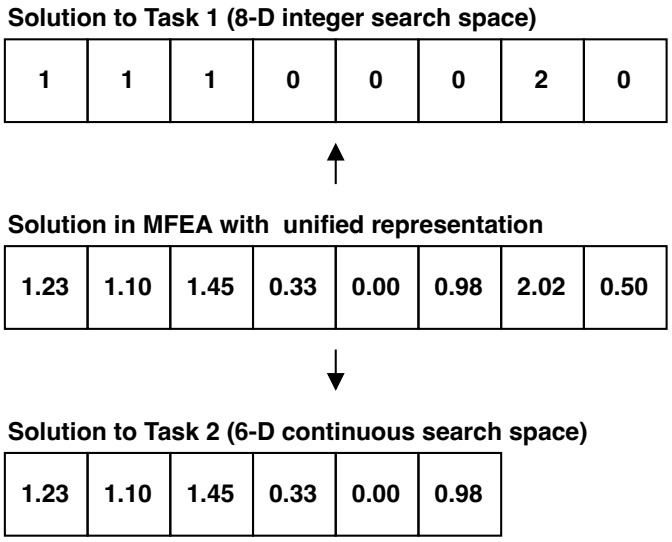

Figure 2.4: An example of unified representation in Multi-Factorial Evolutionary Algorithm (MFEA). The solution in MFEA is encoded with a unified representation. This solution could be mapped into solutions of two different tasks: 1) 8-D integer search space by truncating fractions; 2) 6-D continuous search space by truncating the last two dimensions.

2.3.2 Multi-Factorial Evolutionary Algorithm

Multi-Task Evolutionary Algorithms (MTEAs) are EAs that solve MTO. MTEAs could be divided into two categories: single-population methods and multi-population methods.

Multi-Factorial Evolutionary Algorithm (MFEA) [2] is a single-population method to solve MTO. The decision variables of different tasks are mapped into a unified representation. **Figure 2.4** shows a unified representation of a MTO problem with two tasks. The solution in MFEA [2] could be mapped into a solution in 8-D integer search space by truncating the fractions and a solution in 6-D continuous search space by truncating the last two dimensions. Therefore, by optimizing the solution in the unified representation, we could optimize two tasks across the domains.

By sharing the same unified representation, MFEA implicitly explores the common building blocks of each task. MFEA returns an optimal solution in

the unified representation. This solution holds the optimal objective values on all tasks.

MFEA transforms a vector of multiple objective values to a scalar fitness based on steps in (2.7) and (2.8).

$$\Psi_k(\mathbf{x}) = f_k(\mathbf{x}) + \lambda \cdot \delta_k(\mathbf{x}) \tag{2.7}$$

$$\phi(\mathbf{x}) = 1/\min_{k=1,\ldots,m} r_k(\mathbf{x}) \tag{2.8}$$

First, the objective f_k and the constraint violent δ_k of k-th task are aggregated into a factorial cost Ψ_k using a penalizing multiplier λ. Then, the factorial rank $r_k(\mathbf{x})$ is calculated as the rank of this factorial cost of an individual $\mathbf{x}$ in the entire population in ascending order. For a specific individual $\mathbf{x}$, there are m factorial ranks in total. The scalar fitness $\phi(\mathbf{x})$ of this individual $\mathbf{x}$ is the inverse of the minimal factorial rank among the m ranks.

The skill factor in (2.9) is another important concept in MFEA [2]. It indicates the task where an individual $\mathbf{x}$ holds the minimal factorial rank among all m tasks. In other words, the skill factor shows the most "skillful" task of an individual.

$$\tau(\mathbf{x}) = \arg\min_{k=1,\ldots,m} r_k(\mathbf{x}) \tag{2.9}$$

In MFEA [2], two parents are randomly selected to reproduce their children. MFEA performs a crossover process if the two parents hold the same skill factor (i.e., they are good solutions for the same task) or within a probability. Otherwise, the two parents will go through mutation steps separately. This design shares a similar idea of speciation. After reproducing the offspring population, MFEA merges the offspring and parent population. A survival selection is then performed based on the scalar fitness values. The details of this algorithm could be found in its original publication [2].

2.3.3 Multi-Population Multi-Task Evolutionary Algorithms

Though the pioneering MFEA [2] uses a single population, many of the MTO studies propose and apply multi-population methods [45, 3, 46, 47, 48, 49]. Multi-population Multi-Task Evolutionary Framework (MMTEF) [45, 46] is a representative framework among this group. As shown in **Algorithm 2.3**, MMTEF maintains several sub-populations and uses each sub-population

Algorithm 2.3: The procedure of Multi-population Multi-Task Evolutionary Framework

```
1  foreach task T_i in {T_1, ..., T_m} do
2      X_i ← initialize();
3  repeat
4      foreach task T_i in {T_1, ..., T_m} do
5          P_i ← ∅;
6          while |P_i| < |X_i| do
7              if rand() < α then
8                  x_{i,1}, x_{i,2} ← select(X_i);
9              else
10                 x_{i,1}, x_{i,2} ← select(⋃_{i=0}^{K} X_i);
11             x'_{i,1}, x'_{i,2} ← crossover(x_{i,1}, x_{i,2});
12             P_i ← P_i ∪ {x'_{i,1}, x'_{i,2}};
13     foreach task T_i in {T_1, ..., T_m} do
14         P_i ← local-improve(P_i);
15         X_i ← P_i;
16 until termination criteria are satisfied;
```

X_i to solve a task T_i. Building blocks of the solutions (i.e., a fraction of the genome) are transferred from one sub-population to another via a crossover operator that takes parents from different sub-populations with a probability α (line 7 to line 11). A local improvement is then performed on the solutions in all sub-populations following the crossover. This step could apply either problem-specific heuristics or mutation methods (line 15).

On the one hand, the multi-population approaches are easier to understand and control the evolutionary process compared with single-population MTEAs. Researchers have developed control techniques, including resource allocation [47] and cross-task interaction [3, 48], for multi-population MTEAs. On the other hand, the multi-population approaches require setting several ad-hoc parameters such as the size of the sub-populations; while these parameters are automatically adjusted through the evolutionary process in single-population methods such as MFEA [2].

Knowledge Transfer (KT) is a frequently used term in the research of multi-population MTEAs. In the literature of MTO, it refers to the movement of building blocks of the solution from one sub-population to another.

Table 2.1: An example of Program Synthesis problem. In this problem, the user gives three IO cases. Each IO case includes two inputs and one output. The available instructions are `ADD`, `SUB`, `MULT`, `DIV`, `ARG0`, and `ARG1`.

Input	**Ouput**	**Instructions**
1, 2	9	`ADD SUB MULT DIV ARG0 ARG1`
5, 6	121	
10, 0	100	

Similarly, this dissertation focuses on the adaptive transfer of genetic knowledge in the EAs. The difference and connection between our works and the KT in the MTO literature are discussed in **Section 3.3** and **Section 4.2**.

2.4 Genetic Programming

2.4.1 Program Synthesis

Genetic Programming (GP) is a group of EAs proposed to create computer programs [50, 51, 52]. Such an application is called Program Synthesis (PS) and is defined as follows.

PS aims to find a sequence of instructions from an available set to pass a set of Input-Output (IO) cases. This problem is usually modeled as an OP in (2.10) to minimize the difference between the actual output $\mathbf{p}(\text{in}_i)$ and the expected output out_i of the program $\mathbf{p}$. $\mathcal{P}_{\text{INSTR}}$ includes all possible programs determined by the instruction set INSTR (also called primitive set). In PS, the solution program requires satisfying a set of m IO cases.

$$\begin{aligned} minimize \quad & \{||\mathbf{p}(\text{in}_k) - \text{out}_k\,||\}_{k=1}^{m} \\ s.t. \quad & \mathbf{p} \in \mathcal{P}_{\text{INSTR}} \end{aligned} \tag{2.10}$$

Obviously, there are totally m minimization objective functions in (2.10). Thus, we consider the PS problems that GP solves as OPs with multiple tasks. Unlike MOO in **Section 2.2**, these objectives are not trade-offs, since the optimal solution minimizes all of them.

Table 2.1 gives an example of the PS problem. In this problem, the user gives three IO cases. Each IO case includes two inputs and one output. The available instructions are `ADD`, `SUB`, `MULT`, `DIV`, `ARG0`, and `ARG1`. One of the

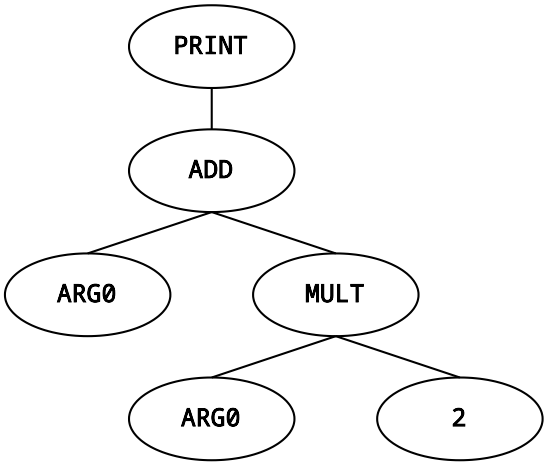

Figure 2.5: An individual in tree-based Genetic Programming. This individual represents the program `PRINT(ADD(ARG0,MULT(ARG0,2)))`.

possible programs is `MULT(ADD(ARG0,ARG1),ADD(ARG0,ARG1))`.

With computer programs, we are able to automate many processes. In fact, many applications of GP are variants of the PS problem, including symbolic regression [53, 54], circuit design [55, 56], robotics design [57, 58], artificial neural network design [59, 60, 61], trading rule extraction [62, 63, 64], and image classification [65, 66].

2.4.2 Koza's Tree-based Genetic Programming

As mentioned in the last subsection, GP is a method to solve PS problems in the EC literature. One of the major contributions of this dissertation is related to a GP variant called PushGP [67]. However, before moving on to this GP variant, we first review Koza's original work on the tree-based GP algorithm [50]. The background on PushGP [67] is presented in **Chapter 6**.

In Koza's initial work [50], a program is encoded as a tree, where a parent node takes its child nodes as arguments. The program output is the computation result of the root node. The tree-based GP [50] maintains a population of programs and evolves them through an evolutionary process.

Algorithm 2.4: The pseudocode of tree-based Genetic Programming

input : population size N, crossover rate p_c, mutation rate p_m, and maximum tree depth $d_{\max}$
output: final population X

$X \leftarrow$ initialize(N);
repeat
  $X' \leftarrow \emptyset$;
  while $|X'| < N$ **do**
    `/* tournament selection */`
    $\mathbf{x}_a, \mathbf{x}_b \leftarrow$ tournament-select(X);
    `/* one-point crossover */`
    if rand() $< p_c$ **then**
      $\mathbf{x}'_a, \mathbf{x}'_b \leftarrow$ one-point-crossover($\mathbf{x}_a, \mathbf{x}_b$);
    `/* uniform mutation */`
    if rand() $< p_m$ **then**
      $\mathbf{x}'_a \leftarrow$ uniform-mutate($\mathbf{x}'_a$);
    if rand() $< p_m$ **then**
      $\mathbf{x}'_b \leftarrow$ uniform-mutate($\mathbf{x}'_b$);
    `/* bloat control */`
    if depth($\mathbf{x}'_a$) $> d_{\max}$ **then**
      $\mathbf{x}'_a \leftarrow \mathbf{x}_a$;
    if depth($\mathbf{x}'_b$) $> d_{\max}$ **then**
      $\mathbf{x}'_b \leftarrow \mathbf{x}_b$;
    $X' \leftarrow X' \cup \{\mathbf{x}'_a, \mathbf{x}'_b\}$;
  $X \leftarrow X'$;
until termination criteria are satisfied;
return X;

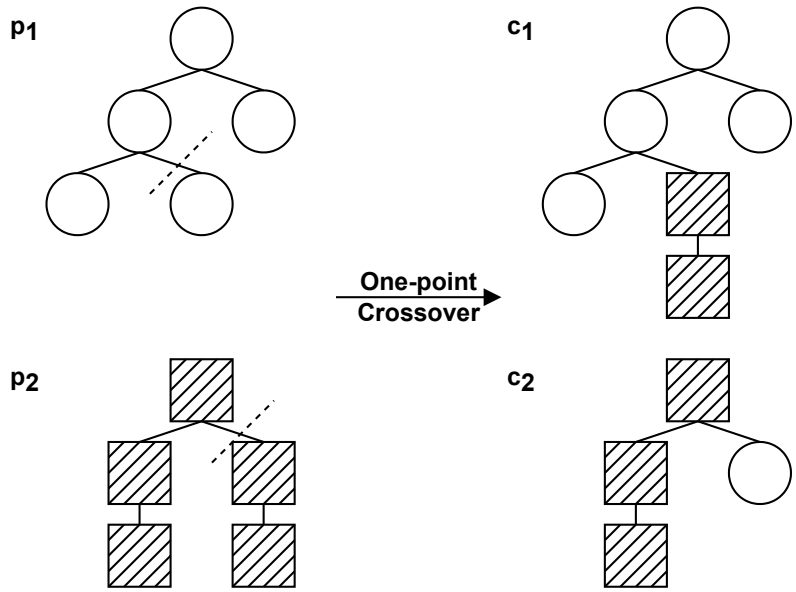

Figure 2.6: One-point crossover for tree-based Genetic Programming. $\mathbf{p}_1$ and $\mathbf{p}_2$ are two parent candidates. The subtrees under the dashed lines are exchanged during the crossover to generate child candidates $\mathbf{c}_1$ and $\mathbf{c}_2$.

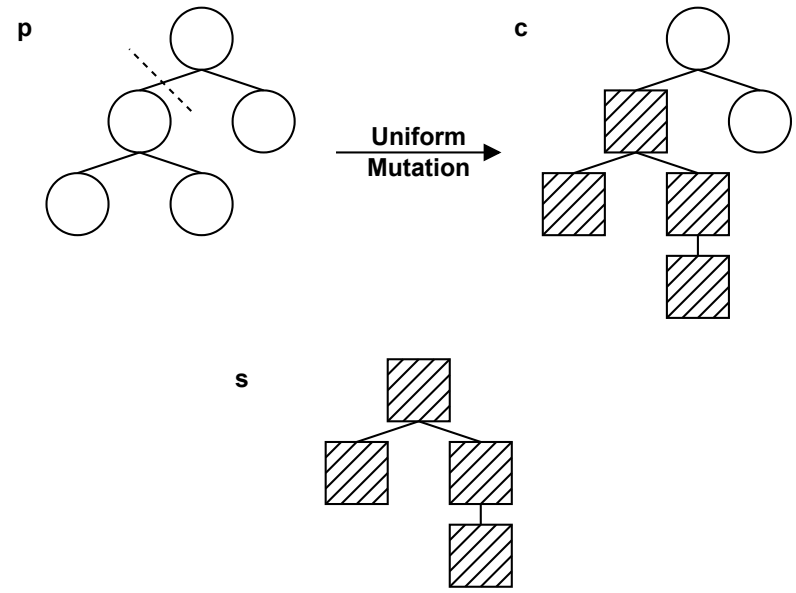

Figure 2.7: Uniform mutation for tree-based Genetic Programming. $\mathbf{p}$ is a parent candidate. To generate the child candidate $\mathbf{c}$, the subtree under the dashed line is replaced by a randomly created tree $\mathbf{s}$.

A typical tree-based GP (**Algorithm 2.4**) uses tournament selection, one-point crossover, and uniform mutation. Additionally, the tree-based GP contains a bloat control technique to prevent the unlimited increase of the tree size during the evolution.

- The tree-based GP generates a population of random programs using the available primitives (or instructions) during the initialization phase.
- In the tournament selection, the differences between actual and expected outputs in m IO cases are summed together as the fitness value of the program. The tournament selection randomly selects a subset of the population and takes the individual with the best fitness in this subset as the selected candidate.
- To perform one-point crossover (**Figure 2.6**), two parents are selected. The algorithm selects a random subtree of each parent and exchanges these subtrees to reproduce two child candidates.
- The uniform mutation (**Figure 2.7**) only requires one parent candidate. This operator selects a random subtree of the parent candidate and replaces the subtree with a randomly generated tree.
- The individuals in GP have variable sizes. For example, the uniform mutation can either increase or decrease the size of the tree. Bloat is an issue when the tree size increases without a limitation. The bloat causes two problems: 1) a long time to evaluate a program and 2) a large search space. A frequent bloat control technique is to revert the child to its parent if a maximum tree depth is exceeded.

2.4.3 Modularity and Automatically Defined Function

An essential characteristic of the programs written by human programmers is modularity [68]. In software engineering, modular programming is a technique that divides a program into independent blocks based on their functionality. These independent blocks are called modules. A module contains everything necessary to execute one of the functionalities.

For human programmers, writing programs with modules brings several advantages. First, modules allow easy reuse of the same code fragments and therefore shrink the solutions. Second, modules could reflect the structure of

```python
def function(x):
    return 5*(5*(5*(5*(5*(5*x+1)+1)+1)+1)+1)+1
```

(a) Python code without modules

```python
def module1(x):
    return 5*x+1

def module2(x, t):
    for i in range(t):
        x = module1(x)
    return x

def function(x):
    return module2(x, 6)
```

(b) Python code with modules

Figure 2.8: A comparison between code with/without using modules. Modules can shrink the code (e.g., it is a lot of work to write `module2(x, 100)` without modules). The code with modules is easier to understand and edit (e.g., we can change `module1` to `x**2-1` easily).

the problem. If a problem could be decomposed into several sub-problems, the modules could be the solution to these sub-problems. Third, using modules makes the code easier to understand and maintain.

Figure 2.8 provides examples of Python code implementing the same functionality. However, the lower one (**Figure 2.8b**) uses modules while the upper one (**Figure 2.8a**) does not use modules. With modules, we decompose the problem into several components, and thus it is easier to understand that the program repeats to perform the operations in `module1` on the input six times. Additionally, we can change the code in `module1` while we have to update several places to perform the equivalent change without using modules. Finally, it seems that **Figure 2.8b** contains more lines of code compared with **Figure 2.8a**. However, one should not be misled by this simple case. Writing an equivalent program to `module2(x, 100)` without using modules would become an endless work (and that is why we do not show this complex example here) while with modules we can just adjust a parameter.

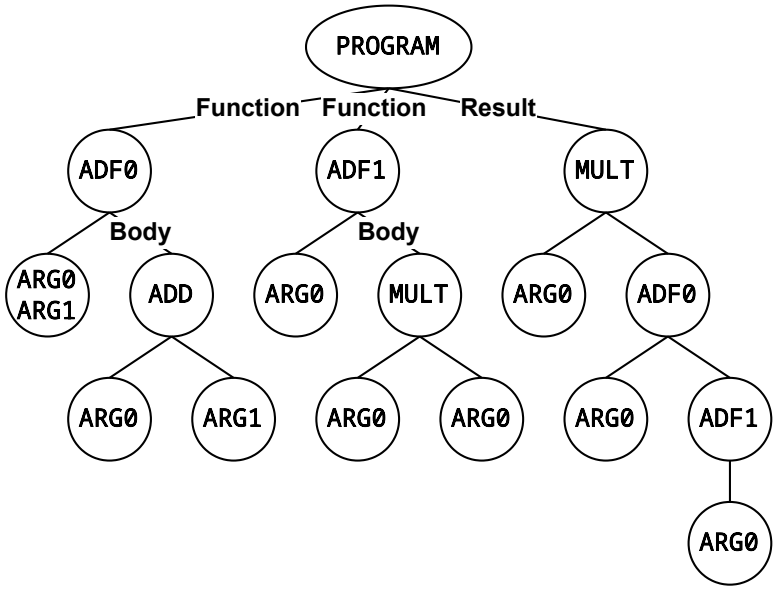

Figure 2.9: A solution with Automatically Defined Functions. The root node contains three branches. The two of them on the left are function-defining branches, while the right one is the result-producing branch. The arguments in the function-defining branches are local arguments. This solution represents the program `MULT(ARG0,ADD(ARG0,MULT(ARG0,ARG0)))`.

GP contains this great feature of modularity as well. In GP, a module is a function consisting of primitives or previously defined modules [68]. The Automatically Defined Function (ADF) [50] is a technique proposed for the tree-based GP to implement the idea of modularization. **Figure 2.9** presents a solution to the tree-based GP with two ADFs. This solution contains two function-defining branches (two left subtrees of the root node) and a result-producing branch (right subtree of the root node). The function-defining branch defines the modules with local arguments and the main body of the function. The result-producing branch may call the functions defined by the function-defining branches. The function-defining branch may also call the previously defined functions or even itself, though this recursive call may cause the infinite evaluation issue.

The word "automatically" in ADF implies that these functions are not defined by users manually. Instead, GP evolves to discover these structures during the search process, although the user is required to set a number of functions in the solution. These ADFs are randomly initialized and mutated during the evolution. However, the crossover operator is not allowed to break these building blocks. In other words, the crossover operator can only swap the function-defining branches between individuals rather than change the internal structures.

According to Woodward's review [68], ADF or modularity implements a key idea of problem-solving, divide-and-conquer. That is, GP divides a problem into smaller and easier sub-problems, solving them independently, and then combining these sub-solutions into a solution to the original problem. In GP, each module is a solution to each of the sub-problems. With modularity, we can simply refer to the solved sub-problems when it occurs again.

The advantage of using modules in GP is similar to modular programming for humans. It allows the easy reuse of the same code fragments, reflects the structure of the problem, and makes the code easier to understand.

In spite of ADF [50], many studies have developed different techniques to realize the modularity in various variants of GP, such as Module Acquisition [69], Subtree Encapsulation [70], Automatically Defined Macros [71], Hierarchy Locally Defined Modules [72], and Tag-based Modularity [73, 74]. It is necessary to point out that these techniques can only allow code reuse inside a single run of the GP on one PS problem. The code reuse across different PS problems is mentioned in the next subsection.

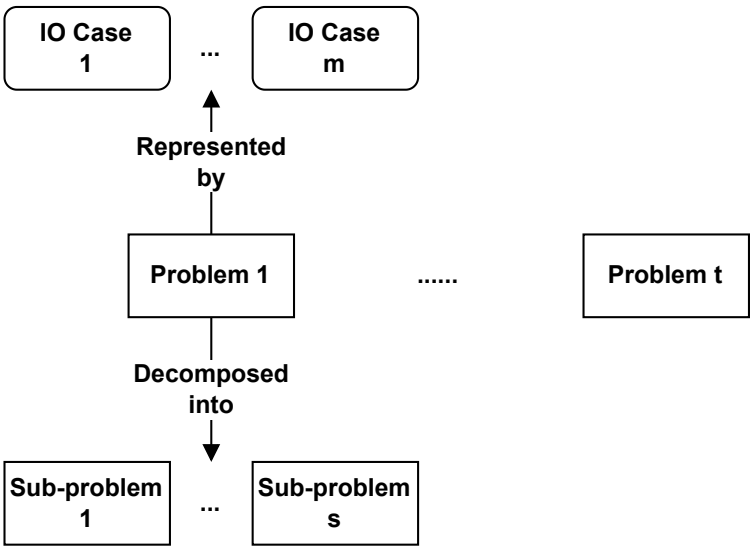

Figure 2.10: Multiple tasks in Genetic Programming at different scales. They are problem level, IO-case level, and sub-problem level.

2.4.4 Multiple Tasks in Genetic Programming

The phrase "multiple tasks" in GP could have different meanings. **Figure 2.10** shows the "multiple tasks" at three different scales in GP.

Problem level Multiple tasks could refer to the multiple PS problems solved by GP, either in parallel or in a sequence. The first work on this idea might be the External Concept Reuse by Seront [10]. Another similar but more recent work is the Run Transferable Libraries (RTLs) [11]. RTLs provide GP with a library of functions as primitives and evolve these functions based on the feedback from GP algorithms, from run to run across a set of related problems. Other studies have attempted to use GPs to solve multiple related tasks sequentially with different algorithm designs in initialization [12, 13], mutation [12, 13], and primitives [14]. Multiple tasks at the problem level in GP is similar to MTO [2] that has been reviewed in **Section 2.3**. The multiple tasks to be solved are related to each other. However, one difference between the two problem paradigms is that MTO usually solves different tasks in parallel rather than in a sequence. There are a few studies on Multi-Task GP [5, 6, 7, 8] where multiple tasks in MTO are

solved by GP algorithms in parallel.

IO-case level A PS problem is represented by a set of IO cases. Multiple tasks could also refer to minimizing the difference based on multiple IO cases. In GP algorithms, selection methods might be the corresponding component to deal with multiple tasks at this IO-case level. In the conventional GP [50], tournament selection is applied with the sum of errors on all IO cases as the fitness evaluation. Lexicase Selection (LS) [19] is an alternative selection method for GP which keeps the objectives or cases separate. LS [19] filters the worst individuals in the population based on a shuffled list of objectives and takes the rest individual as the selected parent. In **Chapter 5**, we solve Seismic History Matching problems with LS. In a more general sense, multiple tasks at the IO-case level can also refer to other objectives that guide the evolution of one PS problem. For instance, several Multi-Objective Genetic Programming (MOGPs) [75, 76, 77] take the individual complexity as another objective to optimize other than minimizing the errors on the IO cases. These MOGPs are designed to control the bloat during evolution. Readers can find more information on this topic in **Section 9** of the book by Poli *et al.* [78].

Sub-problem level A problem could be divided into more than one sub-problems. Solving these sub-problems together with modularity could be considered as another version of multiple tasks in GP. It is notable that in this sub-problem level, there are no specific fitness functions for each sub-problems to guide the search. The evolution of these sub-problems is done by the embedded representation of modules, such as ADF [50]. We have reviewed the related topic on ADF and modularity in **Section 2.4.3**.

Chapter 3

Genetic Knowledge Transfer

In the last chapter, we have reviewed several optimization problems that aim to solve multiple tasks together. In real life, Knowledge Transfer (KT) is an essential idea for solving multiple tasks. KT allows humans to learn from their experiences and improve themselves. In this dissertation, we focus on an agent that is able to do similar things.

This chapter starts with KT in a general sense and introduces a definition of Genetic Knowledge Transfer (GKT) which is the KT in the Evolutionary Computation context in **Section 3.1**. Then, we go through several examples of GKT in the literature, including GKT in Multi-Objective Optimization, GKT in Multi-Task Optimization, and GKT in Genetic Programming in **Section 3.2**, **Section 3.3**, and **Section 3.4**, respectively.

3.1 Definition

Knowledge Transfer (KT) or transfer of knowledge is a useful problem-solving technique when there are more than one tasks to solve and these tasks are related to each other. One example is our Evolutionary Computation (EC) algorithms. We apply our knowledge from the biological domain to solve engineering optimization problems since the optimization process is similar to an evolutionary process.

KT is useful in EC as well. In the real world, many problems are not isolated. However, the conventional EC methods solve them without considering this fact. Most of EC techniques employ random initialization, random variation, and semi-random selection based on fitness. However, when solving more than one task at the same time, KT techniques could leverage the knowledge from a related problem to enhance the search performance. This

chapter provides several existing KT methods in the EC literature, but before that, we give a definition of KT in EC.

In knowledge management studies, KT refers to "the process where one unit is affected by the experience of another" [79]. The term KT is also widely used in some topics of Machine Learning, such as Transfer Learning [80], Multi-Task Learning [81], and Knowledge Distillation [82].

In the EC literature, Multi-Task Optimization (MTO) [2] is a sub-field that often employs the idea of KT. Gupta *et al.* [83] have come up with a definition of knowledge as follows.

> *The knowledge extracted a posteriori from an unknown optimization task is identical to the prior knowledge required to spontaneously address the same task.*

However, this definition of "knowledge" is far from practice. In fact, the authors suggested in the same work [83] that the knowledge of the problem is usually embedded in the elite solutions in the population of an Evolutionary Algorithm (EA). Moreover, they did not provide a definition of "transfer" which is another important concept in KT.

Therefore, we propose a definition of KT in the context of EC in **Definition 1**. To differentiate the KT in EC from KT in other fields, we call it Genetic Knowledge Transfer (GKT). Our definition of GKT derives from the definition of KT in knowledge management studies in the first paragraph of this section [79].

Definition 1 *Genetic Knowledge Transfer is the process where one EA is affected by the dynamic of another.*

Clearly, the entire dynamic (experience) of an EA is considered as genetic knowledge. It includes all the individuals sampled during evolution and interactions between them. We name it as raw knowledge. The term transfer is the process or the act to affect an EA. The transfer process is usually done by specifically designed operators. The affecting EA is source EA and the affected EA is target EA.

Usually, using the entire dynamic of an EA (raw knowledge) is not efficient, since the solutions in the first beginning generations hold worse fitness

values and seldom contain the building blocks of the problem. Instead, we can select representatives from the entire dynamics, such as the best individual and the final population. We name these selected representatives as representative knowledge. Moreover, some prior studies use learned structures [84] or high-level statistics [85] of the high-quality solutions as knowledge. We call this type of knowledge processed knowledge.

Finally, GKT happens between two EAs. Therefore, methods that inject building blocks created by humans are not in this category. For the sake of convenience, we sometimes use descriptions such as "GKT between Task A and Task B" instead of "GKT between EA that solves Task A and EA that solves Task B". Similarly, we use "source task" and "target task" in place of "the task that is solved by source EA" and "the task that is solved by target EA", respectively.

In the following sections, we provide examples of GKT in different fields of EC, including Multi-Objective Optimization (**Section 3.2**), MTO (**Section 3.3**), and Genetic Programming (**Section 3.4**).

3.2 Examples in Multi-Objective Optimization

A very good example of GKT in Multi-Objective Optimization (MOO) happens in the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) [36]. We have reviewed MOEA/D in **Section 2.2.3**.

In MOEA/D [36], a MOO problem is decomposed into a set of single-objective optimization tasks (sub-problems). These tasks are solved in parallel using a single population. In this population, every individual solution serves as a solution to one sub-problem.

One may notice that there is only one individual for every sub-problem, rather than a population. Interestingly, MOEA/D selects parents from neighbor solutions (line 4 in **Algorithm 2.2**) to reproduce the child individual of a sub-problem by crossover.

If we take off the crossover operator, the mutation step will work on the single solution that is associated with the current sub-problem. In other words, those single-objective optimization problems in MOEA/D are solved by a single-solution metaheuristic; however, combined with a crossover operator that performs GKT from a second metaheuristic that solves neighbor

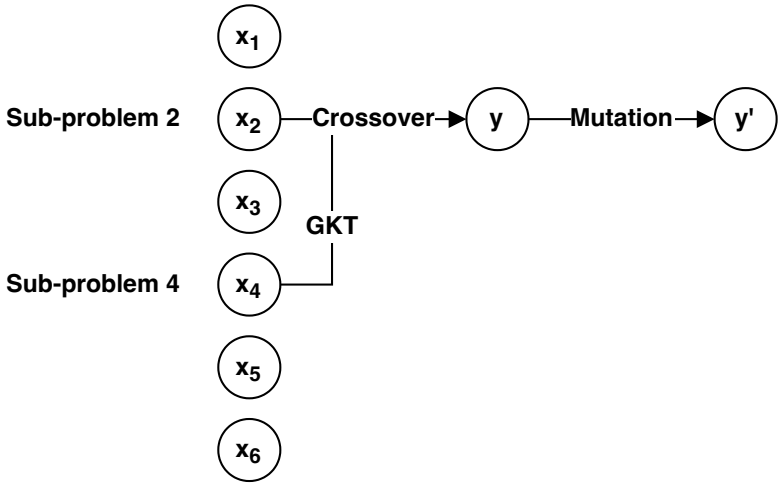

Figure 3.1: An example of Genetic Knowledge Transfer in MOEA/D. To generate the child individual of Sub-problem 2, the algorithm selects $\mathbf{x}_2$ and $\mathbf{x}_4$ to perform crossover. This step could be considered as transferring genetic knowledge from an EA that solves Sub-problem 4 to the current EA that solves Sub-problem 2.

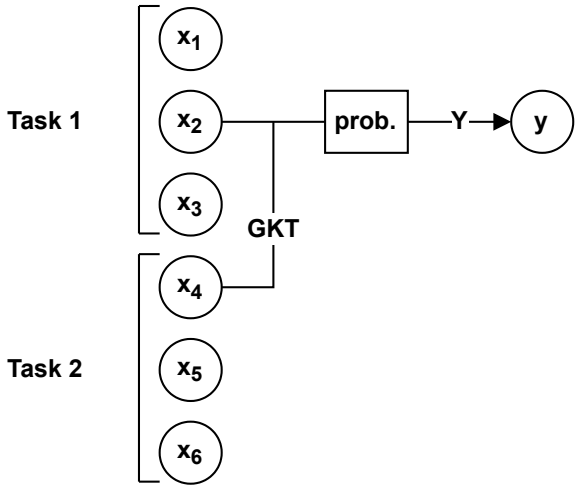

Figure 3.2: An example of Genetic Knowledge Transfer in Multi-Task Optimization. $\mathbf{x}_1$ to $\mathbf{x}_3$ are the solutions to Task 1 while $\mathbf{x}_4$ to $\mathbf{x}_6$ are the solutions to Task 2. Genetic Knowledge Transfer occurs when the algorithm crossovers $\mathbf{x}_2$ with $\mathbf{x}_4$ based on a probability.

sub-problem (**Figure 3.1**). Here, representative knowledge refers to the individual of a sub-problem in the current generation.

In spite of the above example, there are other techniques that could be considered as GKT in the literature of MOO, such as transferring a set of elite solutions from single-objective problems as representative knowledge to a MOO problem [86] and applying learned structure as processed knowledge to different sub-problems in MOEA/D [84].

3.3 Examples in Multi-Task Optimization

MTO [2] is naturally associated with the idea of GKT. MTO requires solving multiple tasks simultaneously and exploiting the common knowledge among these tasks. Therefore, almost all studies on MTO could be considered as examples of GKT. The reader could refer to a survey by Wei *et al.* [4].

In **Section 2.3.2**, we have introduced Multi-Factorial Evolutionary Al-

gorithm (MFEA) [2]. In MFEA, GKT is performed when the crossover operator takes two parents that hold different skill factors. The skill factor shows the best-performed task of a solution. Therefore, this crossover combines the genetic materials from the solutions that are good solutions to different tasks.

The Multi-Task Evolutionary Algorithms (MTEAs) with multi-population methods maintain several sub-populations to solve multiple tasks. The GKT in these algorithms is similar to what in MFEA [2]. The crossover operator takes two parents from different sub-populations (that are used to solve different tasks) and reproduces child solutions. In both MFEA [2] and the multi-population methods, this crossover is performed with a probability. The representative knowledge here is the individual with high fitness.

Among all MTEAs, we highlight Adaptive Archive-based Many-Task Evolutionary Algorithm (AAMTEA) [3]. In that work, many tasks are solved in parallel and these tasks are not necessarily to be similar. Most MTEAs transfer genetic knowledge between any pair of tasks. The authors have suggested that this type of "aimless" transfer might result in the waste of computational budget. To address this issue, they have come up with a novel method to select source tasks and target tasks based on a similarity measure between tasks and adaptive rewards from the evolutionary process.

3.4 Examples in Genetic Programming

In **Section 2.4.4**, we have mentioned the three levels of multiple tasks in Genetic Programming (GP), namely sub-problem level, IO-case level, and problem level.

GKT seldom occurs at the sub-problem level, since the crossover is not allowed to change the structure of the modules such as Automatically Defined Function (ADF) [50] during the reproduction period.

When looking at the IO-case level, it is hard to decouple the original evolutionary process into several separate EAs, if we only consider those methods that aggregate IO cases by simply taking the sum. However, there are a few studies that could be considered as examples of GKT. Multi-Objective Genetic Programming based on Decomposition [87, 88] is a variant of MOEA/D [36] that incorporates the encoding, crossover, and mutation of the GP algorithm. The GKT that happened in these methods is similar to

that happened in MOEA/D (**Section 3.2**). Lexicase Selection (LS) [19] is another way to aggregate the IO cases in GP algorithms. The GKT in GP with LS is discussed in **Chapter 5**.

Most examples of GKT in GP happen at the problem level. In other words, the GKT occurs among several GPs that solve different Program Synthesis (PS) problems. Some studies [5, 6, 7, 8, 9] have applied GP to solve multiple PS problems as MTO [2]. The GKT in these works is the same as the GKT that happens in MTO (**Section 3.3**).

Other than these works, several studies [10, 11, 12, 13, 14, 15, 16] have attempted to solve different tasks in a sequence. These tasks are usually similar and the number of the tasks is usually small. For example, Wick *et al.* [15] have proposed a method that solves a sequence of PS problems starting from simple problems and then harder ones. The final population of the just-solved problem will be used as the initial population of the next PS problem. Therefore, the genetic knowledge in that work is the final population. Other studies have utilized primitive sets [10, 11], subprograms [12, 13, 14], and best solutions during the whole run [16] as representative knowledge.

Chapter 4

Adaptive Transfer of Genetic Knowledge

Humans solve endless tasks in their whole lives. However, not all past tasks are helpful for a specific task to solve. Generally speaking, the experiences on similar tasks are usually more helpful than that on dissimilar ones. Trial and error is another common way to figure out the helpful tasks when the similarity of the tasks is not easy to compute.

This dissertation investigates an Evolutionary Algorithm that solves many tasks like a human. In **Chapter 3**, we have learned about several Genetic Knowledge Transfer (GKT) methods. However, most of them work on a few similar tasks. In this chapter, we propose a system design that allows efficient GKT among many tasks. Moreover, these tasks are not necessarily to be similar.

Section 4.1 describes Naive Genetic Knowledge Transfer. Most of the conventional GKT methods belong to this category. **Section 4.2** provides the details of the proposed design of our Adaptive Genetic Knowledge Transfer System.

4.1 Naive Genetic Knowledge Transfer

Figure 4.1 shows an intuition of Naive Genetic Knowledge Transfer (NGKT). The NGKT happens between two Evolutionary Algorithms (EAs) when some of the individuals from the offspring population of the source EA are selected and used as parents during the reproduction steps of the target EA.

Many prior studies that transfer the best individuals of current generation [4], final populations [15], and final solution [16] could be considered as special cases of this NGKT. Some studies, such as the work by Wick *et al.* [15],

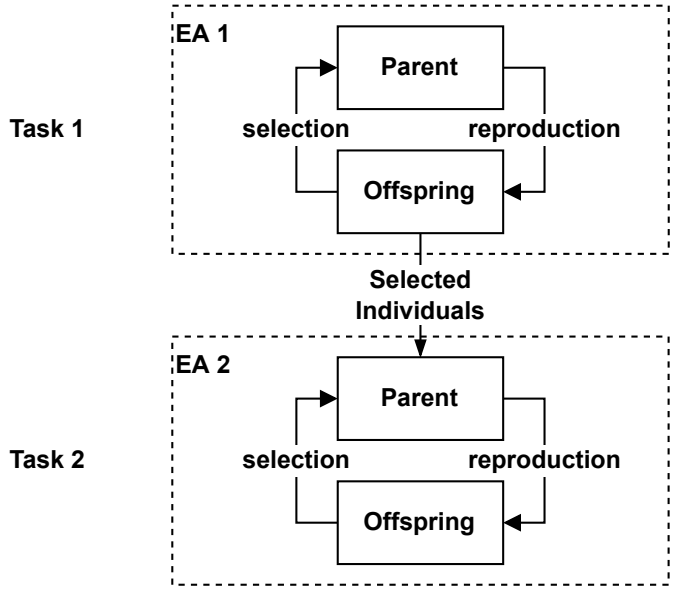

Figure 4.1: Naive Genetic Knowledge Transfer (NGKT). The NGKT happens between two Evolutionary Algorithms (EAs). It selects individuals from the offspring population of the first EA and uses them as parents of the second EA to reproduce child individuals.

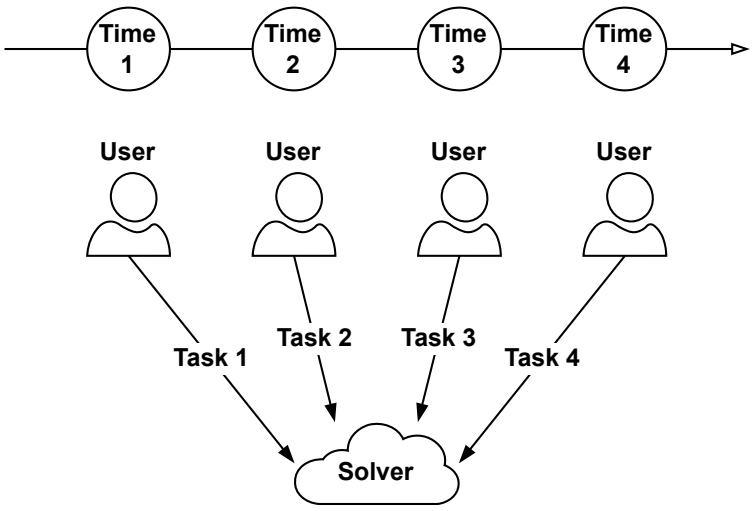

Figure 4.2: A cloud service solving multiple tasks in a sequence. These tasks are from the same user; however, at different time steps.

have proposed to perform GKT among more than two tasks. These works could be considered as a composite of several NGKT.

In addition, some researchers have proposed to transfer building blocks (partial individual) [12, 13] or high-level statistics [16] instead of the individuals. These methods are not exactly the same as NGKT but could be considered as NGKT with extraction methods (to extract building blocks or high-level statistics).

4.2 Adaptive Genetic Knowledge Transfer System

4.2.1 Solving Many Tasks in a Sequence

Multi-Task Optimization (MTO) [2] considers an optimization solver (EA) on the cloud service that could be accessed by multiple users at the same time. Therefore, MTO aims to solve several optimization problems simultaneously and exploit the common knowledge between tasks.

In this study, we assume a different scenario (**Figure 4.2**) where an EA

is run on the cloud service and a user consecutively poses a variety of tasks to this EA for a long time. In other words, the EA is required to solve a sequence of many problems (one at a time step) and use the knowledge gained from past tasks to improve performance when solving the current task. Moreover, these tasks are in a large number and are not necessarily to be similar. This scenario is similar to the case where a human solves tasks consecutively and improves himself/herself based on past problem-solving experiences.

This scenario is important and interesting for the following reasons. First, similar to MTO [2], this sequential problem-solving exploits the common knowledge between related tasks and therefore improves the performance as well. Second, there are endless optimization tasks in the real world, and it is then not possible to solve all of them by the means of MTO [2]. Therefore, studying the methods that work in this sequential problem-solving scenario definitely has its practical meaning. Third, the way that an agent solves problems consecutively and improves itself is similar to how humans develop intelligence.

4.2.2 System Design

In this section, we propose a system design that uses GKT to solve many tasks in a sequence and improves itself during problem-solving. Compared with NGKT, there are several issues to concern.

Many unrelated tasks Only a few tasks are related to the current task to solve. Therefore, it is important to design a method to automatically figure out helpful genetic knowledge. Assume an archive of genetic knowledge from different tasks, there are two ways to find helpful genetic knowledge.

1. The first method filters the archive based on the similarity between tasks. That is, only the genetic knowledge that is from similar tasks to the current task will be transferred.

2. The second method employs the idea of trial and error. This method is similar to the self-adaptation of parameters in an EA such as JADE [89]. The self-adaptation methods search for the proper parameters of an EA in parallel with the main search in the solution space. The second method starts using different pieces of genetic knowledge in the archive with equal probability; however, the probability to use a piece of the

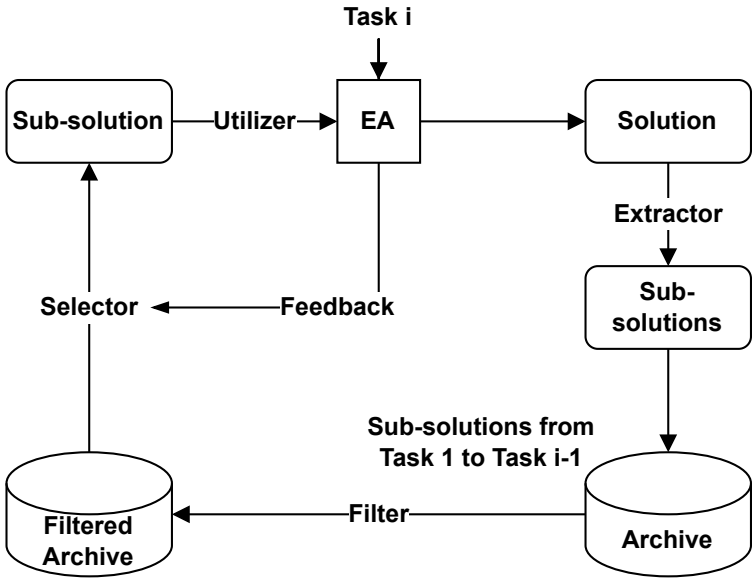

Figure 4.3: Design of Adaptive Genetic Knowledge Transfer (AGKT) system. The system consecutively solves tasks, extracts sub-solutions as genetic knowledge, and uses the genetic knowledge in future tasks.

genetic knowledge is then updated according to the feedback from the main EA solver.

3. The similarity-based method requires a way to compute the similarity between tasks, while the self-adaptive method might not work well when there is a large number of genetic knowledge.

Sub-tasks A task sometimes could be decomposed into several sub-tasks. Even if the two tasks are similar, they still have different sub-tasks in spite of common ones. Therefore, it is promising to have a technique that divides raw knowledge into components. For example, the system could use sub-solutions of the final solution of an EA as processed knowledge in the GKT.

Limited storage If the system is going to run for a long time, there will be a limit on the storage capacity. That is, we cannot store all individuals sampled during evolution for every run of EAs. Since we are focusing on a sequence of problems, a simple idea is to use the final solution that the EA generates for a task as the representative knowledge.

Our system design is provided in **Figure 4.3**. At the moment, the system has solved i-1 tasks and is going to solve the i-th task. The system solves Task i as follows.

1. The system generates a subset of the archive based on the similarity between the current task (Task i) and the tasks associated with sub-solutions. This subset is called a filtered archive.
2. The system selects sub-solutions based on the fitness feedback and utilizes the selected sub-solution in the reproduction steps of the EA in a similar sense to the self-adaptation.
3. The system generates the solution of Task i.
4. The system extracts sub-solutions from the solution of Task i and stores them in the archive.

We call this method Adaptive Genetic Knowledge Transfer (AGKT) System. Our system design shares a similar idea to Adaptive Archive-based Many-Task Evolutionary Algorithm (AAMTEA) [3]. However, there are at least two differences. AAMTEA [3] focuses on the case where many tasks are solved in parallel. Therefore, AAMTEA [3] keeps the historical individuals of all tasks, while our method can only access the final solutions of the past tasks. Moreover, our method considers sub-tasks by using sub-solutions as genetic knowledge. AAMTEA [3] does not consider sub-tasks. The genetic knowledge in that work [3] is a set of good individuals in the current generation.

This proposed system is rather conceptual. However, we first develop the general idea of this system of EA to solve a sequence of many tasks, extract genetic knowledge and improve itself. Moreover, we point out some possible designs of the important components. In the later chapters, we develop and test initial versions of the key components that demonstrate the viability of the system.

Chapter 5

Multi-Criteria Seismic History Matching

Seismic History Matching (SHM) is a key problem in the Geoscience community, requiring optimal parameters of a subsurface model that match the observed data from multiple in-situ measurements. Therefore, the SHM problems are usually solved with Multi-Objective Evolutionary Algorithms (MOEAs). This group of algorithms optimizes multiple objectives simultaneously, considering the trade-off between objectives. However, SHM requires solutions that are good on all objectives, rather than a trade-off.

In this case study, we propose a Differential Evolution algorithm using Lexicase Selection to solve the SHM problems. Unlike the MOEAs, this selection method pushes the solutions to perform well on all objectives. We explain how this selection method performs GKT. We compare this method with two MOEAs, namely Non-dominated Sorting Genetic Algorithm II and Reference Vector-guided Evolutionary Algorithm, on two SHM problems. The results show that this method generates more solutions near the ground truth.

5.1 Introduction of the Case Study

Optimization problems for subsurface flow processes, are a key problem in the Geoscience community, especially the Seismic History Matching (SHM) problem, which we will use here as a case study. It is about the matching of model parameters with data obtained from in-situ measurements. The objective of SHM is to find a model or small set of models which best match the in-situ measurements. This calibration process improves the prediction accuracy of the starting models and is a necessity for safe and economically sound development of subsurface energy systems (energy storage, geothermal, Oil and Gas). A small and accurate set of final models is needed because the

simulation of reservoir fluid flow is computationally expensive.

The SHM data assimilation problem is well-known as being a highly difficult inverse problem to solve [90, 17, 91, 92, 93, 94, 95]. The typical data to assimilate in a history matching exercise are wells and seismic data, which are usually represented as time-series and matrices, respectively. How to merge these two rather different attributes in a single objective function is a non-trivial problem [96]. One "classical" approach is to calculate the mismatch of wells and seismic maps and then compose a weighted single objective function. These weights are generally chosen by engineering judgement, consequently it is then questionable to sum directly these two objectives together, as they represent different entities and physical measurements.

To circumvent these problems, several authors have employed Multi-Objective Evolutionary Algorithms (MOEAs) [97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107], and Many-Objective Evolutionary Algorithms [108]. Typically, these algorithms work simultaneously on several conflicting objectives (i.e. the optimality on each objective cannot be achieved at the same time), looking for the best trade-off set of solutions, which is called the Pareto Front (PF) [20]. Therefore, selection methods that consider the trade-offs of multiple objectives at the same time are usually introduced in this group of algorithms. For example, the Reference Vector-guided Evolutionary Algorithm (RVEA) [21] uses a reference vector-guided selection based on the linear combinations of the objectives. Non-dominated Sorting Genetic Algorithm II (NSGA-II) [20] uses the fast non-dominated sorting that consider a partial order based on all objectives.

However, this trade-off approach is precisely what we propose to investigate in this paper. We postulate that a "trade-off of objectives" is not a good mental model to describe SHM problems, and therefore MOEAs are not well adapted for these problems. This is because in SHM, the different objectives are not strictly trade-offs, but rather different descriptions of a same physics. Eventual differences in objective values are explained by the sparseness and uncertainty of the available data, and not because of some inherent incompatibility of the two objectives (compare this with the more traditional financial portfolio optimization problem, where return and risk of an investment are objectives that are normally at odds with each other). Ideally, we are interested in solutions that show high performance in all objective measures, and a solution that is very good in one objective and very bad in others is likely unphysical.

Other problem formulations in the Evolutionary Computation (EC) literature are close to this "non trade-off" model, such as Multi-Task Optimization (MTO) [2] and Multi-Form Optimization. The readers may refer to Gupta's review work [24] for the general background. In particular, we highlight Program Synthesis (PS), where the goal is to optimize a computer program that can solve a generalized logical task. Each instance of the task is considered a separate objective, and an optimal program must solve as many of these instances as possible [19].

We consider the SHM problem to be more similar to this formulation, in the sense of objective aggregation, so we propose a method for aggregating multiple objectives for the SHM problem based on Lexicase Selection (LS) [19]. LS is a method originally proposed for PS tasks. The primary concept, is to filter the solutions based on a shuffled arrangement of all objectives, so that it can drive the solutions to a better quality in all objectives at the same time, while providing enough diversity during the search process. We provide a detailed background of this method in **Section 5.3**. We further provide an explanation on how this LS performs Genetic Knowledge Transfer (GKT). We introduce a new Differential Evolution algorithm using LS for the SHM problem, which is described in **Section 5.4**.

We tested our proposed algorithm on two SHM problems, referred in the following as: TS2N and Volve. TS2N [109] is a simple model containing a single injection and production well pair. This model contains very low trade-offs between the objectives, but it will serve as a calibration for the new implemented method. The Volve model [110], on the other hand, is a real-world case, and significantly more difficult to optimize, with multiple well and seismic objectives.

We compared the proposed algorithm with two well-known MOEAs, NSGA-II [20] and RVEA [21]. We discuss the results from various perspectives, including the distance to the ground truth, the difference on set coverage, the distribution of the non-dominated solutions, as well as the prediction performance. Our experiment shows that this method has the following characteristics:

1. A better optimization performance can be achieved for the SHM problem (**Table 5.3** to **Table 5.6** in **Section 5.5.3**).

2. The final solution set is concentrated in the center of the PF, with fewer extreme solutions which would possibly be non-physical (**Figure 5.4**

in **Section 5.5.3**).

3. Weighting or merging of independent objectives is not required, greatly simplifying the data assimilation processes (**Algorithm 5.2** in **Section 5.4**).

5.2 Seismic History Matching

5.2.1 Problem Description

Subsurface flow data assimilation problems [111, 112] and more precisely, SHM [90, 17, 91, 92, 93, 94, 95], developed in this paper, are very challenging to solve and very active area of research in the Geosciences community, which consists on merging predictions with observations. We aim in matching a multi-stream of data: map-based and "point scale" based (respectively, time-lapse seismic and multiple well data), to the reciprocal maps issued by the simulation model. The ultimate goal being to have at disposal a robust and reliable model update, able to inform on field connectivity [113, 114], geological features identification [115] or support to decision-making [116], for instance. The complexity lies in the fact that seismic contains uncertainty, such as structural errors and noise, which are very difficult to estimate. For this very reason it is hard to find wells and seismic data in agreement, therefore a multi-objective function approach is sound.

5.2.2 Multi-Objective Evolutionary Algorithms in Seismic History Matching Literature

In this section, we first provide background of MOEAs and their application in the seismic history matching literature. After that, we point out the limitations of the MOEAs and introduce a different approach, Lexicase Selection, to handle with the multiple objectives in the SHM problems.

There are two classical MOEAs that have been frequently cited in SHM literature, and illustrate two different approaches to find a PF: NSGA-II [20] uses a domination approach, while RVEA [21] uses a decomposition approach.

NSGA-II NSGA-II [20] is a traditional genetic algorithm where the selection operator, which selects the solutions to keep for subsequent iterations, is modified to take into account the notion of Pareto-optimality. This selection method contains two components, fast non-dominated sorting and crowding distance assignment. Fast non-dominated sorting assigns a rank to each solution depending on their dominance relationship to the rest of the solution set. The tie-breaker for solutions with the same rank is the crowding distance, which makes similar solutions in the objective space less likely to be selected.

Other than the selection method, NSGA-II [20] uses elitism strategy, and generates new solutions by simulated binary crossover and polynomial mutation.

RVEA RVEA [21] is a decomposition-based MOEA proposed for Many-Objective Optimization Problems (MaOPs). MaOPs contain more than four objectives. In these high dimensionality situations, domination-based selection does not provide enough selection pressure for the optimization process. To address this issue, RVEA [21] uses a novel selection method called reference vector-guided selection. It contains four steps: objective value transition, population partition, angle penalized distance (ADP) calculation, and the elitism selection. The objective value transition scales the objective values so that the length of the objective vectors is between 0 and 1. The next step, population partition, divides the whole population into several sub-populations based on the cosine similarity between the objective vectors of the individual and the unit reference vectors. After that, ADP of the individuals will be computed as the objective vector length with a penalty related to the angle between the objective vector and the reference vector. The algorithm selects the elites of each sub-population based on ADP.

The remainder of RVEA [21] contains random parent selection, simulated binary crossover and polynomial mutation, as well as an additional step to automatically adjust the reference vectors by generations.

The early studies in the SHM literature aggregated the objectives by weighted sum and solved with Single-Objective Evolutionary Algorithms such as Particle Swarm Optimization (PSO) [117, 118]. However, many recent studies started to solve the SHM problems using MOEAs.

Schulze applied Strength Pareto Evolutionary Algorithm (SPEA) to solve

the SHM problem [97], while Min and Negash modified and applied Multi-Objective GA (MOGA) in three separate works [119, 120, 121]. NSGA-II is the most frequently used algorithm in the SHM literature [122, 103, 105, 104, 106, 107]. Mohamed [99] and Christie [100] did the similar comparison studies between Multi-Objective PSO (MOPSO) and Single-Objective PSO (SOPSO) in two separate works. Hutahaean applied MOPSO in his two studies [101, 102]. Ilamah applied MOEA/DD [123]. Hutahean applied RVEA to solve Many-Objective SHM problems (4-objectives and 6-objectives) [108].

Based on our review, subsurface researchers have applied various MOEAs to solve the SHM problems, including domination-based approaches such as SPEA [97], SPEA2 [98], MOPSO [99, 100, 101, 102], and NSGA-II [103, 104, 105, 106, 107], as well as decomposition approaches such as RVEA [108] and MOEA/DD [123]. Most of these studies have indicated that the MOEAs can achieve better results on the SHM problems, compared to using single-objective EAs.

However, we suggest that there are at least three limitations of using MOEAs to solve the SHM problems.

1. MOO assumes that the objectives are strict trade-offs in competition with each other. We think, that for SHM problems the trade-off between competing objectives can be considered weak and that many objectives are either localized (do not impact each other) or are complimentary, where an improving fitness in one objective is usually complimentary of other objectives. This notion also fits with the overall desire to find models which best fit all objectives in SHM. The complexity and variability of SHM leads to a unique level of objective competition for each model, making generalization of this rule difficult.

2. Following from 1, it is usually not necessary (or desirable) to achieve the entire Pareto Front in SHM problems. Reservoir engineers are normally not interested in extreme points that hold high misfit on several objectives but low misfit on the rest, as such points usually represent unphysical solutions (data unbalanced solutions).

3. Some components of the MOEAs, such as the crowding distance assignment in NSGA-II [20], aggregate different objective values by the simple addition. This is a questionable practice when different objectives correspond to different physical entities, and requires arbitrary tuning of the scaling factors for these objectives.

In the next section, we introduce LS [19] that handles the multiple objectives differently from MOEAs. We argue that LS is able to overcome these limitations.

5.3 Lexicase Selection

Algorithm 5.1: Pseudocode of Automatic ϵ-Lexicase Selection

input : list of objective functions F and population X
output: selected individual $\mathbf{x}_{lex}$

```
F_r ← shuffle(F);
A ← X;
repeat
    f ← pop(F_r);
    for x_j in A do
        if f(x_j) > f_min + σ then
            A ← A \ {x_j};
until |F| = 0 or |A| = 1;
x_lex ← random-select(A);
return x_lex;
```

LS is first proposed for Genetic Programming (GP) to solve PS problems [19]. In a PS problem [19], the goal is to find a computer program that passes a set of example input and output. Therefore, in the PS problem [19], solutions are expected to have good performance on all objectives. This shows a difference from MOO, where solutions are allowed to perform sub-optimally on some objectives (i.e., extreme points). We think the SHM problems are more similar to the PS tasks rather than MOO, since a solution performing well in one objective should not, in principle, always cause it to perform worse in another.

The basic idea of LS [19] is to filter the population based on each of the objectives in a random order. Each time before the algorithm selects an individual, the order of the objectives will be shuffled. Then the algorithm keeps the best individuals based on the objectives in the previous shuffled order, until there is only one individual left or all the objectives have been used (in this case, the algorithm returns a random individual from the rest individuals).

One disadvantage of this selection scheme is that it can lead to a poor

performance on problems with a continuous fitness space, since few individuals share the same elitism unless they are exactly identical. Therefore, in this case, only one objective will be used to select a parent in the continuous fitness space.

To solve this problem, Cava *et al.* have proposed the Automatic ϵ-LS [124]. Automatic ϵ-LS [124] differs from the basic LS [19] by introducing an adaptive threshold parameter σ to solve the previous issue. For a minimization problem, the individuals whose fitness value is less than $f_{min}+\sigma$ are considered as the "best" individuals. f_{min} is the minimum objective value in the population. σ is calculated based on median absolute deviation (MAD) in (5.1), where median$(\cdot)$ takes the median of a set. A detailed procedure of this method is provided in **Algorithm 5.1**.

$$\sigma = \text{median}\left\{|\,\text{median}\{f(\mathbf{x}_k)\}_{k=1}^{|A|} - f(\mathbf{x}_i)|\right\}_{i=1}^{|A|} \tag{5.1}$$

In the Automatic ϵ-LS [124], each parent is elite on at least the first objective used to select it. Since each parent is selected by a random order of the objectives, the individuals are pressured to perform well on various combinations of the objectives, which enhances the diversity of the population. However, the disadvantage of this method is also obvious. When the number of objectives is small, there are not enough combinations of objectives to provide the diversity. Thus, the algorithm can become greedy. But since the available computational budget in the SHM problems is usually limited, we believe this greedy characteristic does little harm for this application case.

5.3.1 Genetic Knowledge Transfer in Lexicase Selection

In this section, we explain how GKT happens in LS. In **Section 4.1**, we provide a definition of Naive GKT (NGKT) where a part of individuals of an EA are selected as parents to reproduce the offspring of another EA.

Figure 5.1 shows the process of an EA using LS to solve two tasks (Task 1 and Task 2). This figure is divided in two parts showing the different views of the same process.

LS selects parents based on a shuffled list of objective functions. In this case, there are totally two possible orders (Order 1 and Order 2). Order

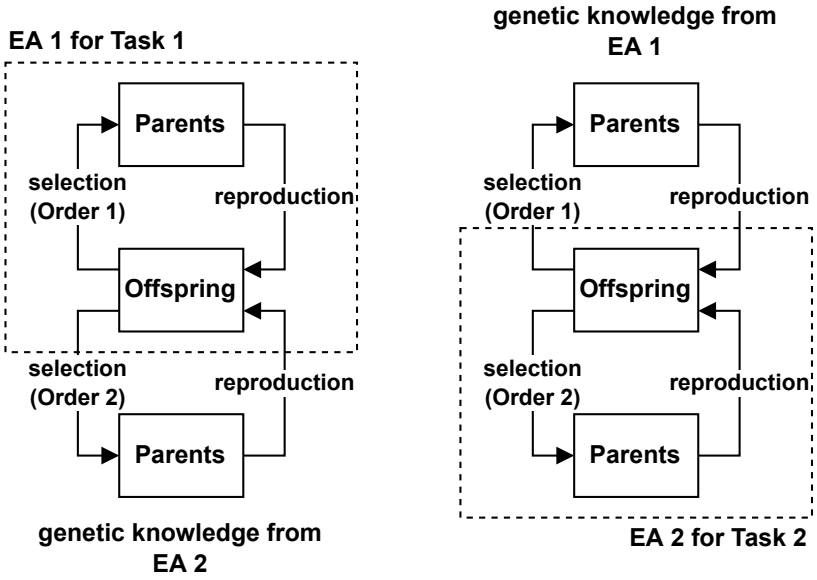

Figure 5.1: Genetic Knowledge Transfer (GKT) in Lexicase Selection (LS). The figure shows an example of LS with two tasks. On the left, the first Evolutionary Algorithm (EA 1) evolves population based on Task 1 and meanwhile LS transfers genetic knowledge from EA 2 that solves Task 2. On the right, EA 2 evolves based on Task 2 while LS transfers genetic knowledge from EA 1.

1 keeps the better individuals of Task 1 at first and then keeps the better individuals of Task 2 among the rest individuals. Similarly, Order 2 keeps the better individuals of Task 2 and then keeps the better individuals of Task 1 in the rest individuals. Therefore, Order 1 selects better individuals of Task 1 while Order 2 selects better individuals of Task 2.

On one hand, the components inside the dashed box in the left part could be considered as EA 1 that solves Task 1. On the other hand, the components inside the dashed box in the right part could be considered as EA 2 that solves Task 2.

Moreover, the components out of the dashed box in the left part are then transferring the genetic knowledge from EA 2 (good individuals of Task 2) to EA 1. Similarly, the components out of the dashed box in the right part are transferring the genetic knowledge from EA 1 (good individuals of Task 1) to EA 2.

Therefore, LS solves all tasks with only one population and implicitly performs GKT among tasks. When the tasks are similar, this implicit GKT could enhance the performance of LS.

5.4 Differential Evolution based on Lexicase Selection

In this study, we propose a novel method to solve the SHM problems based on the Differential Evolution (DE) [27] and the Automatic ϵ-LS [124]. DE [27] is a simple yet powerful optimization algorithm, especially on continuous domain. Its superiority has been proven in many prior studies [95, 125, 126].

In most MOEAs, there are some components doing arithmetic operations on the objective values of the different objective functions. For example, the crowding distance in NSGA-II [20] is computed as a Manhattan distance that adds objective values of different objective functions directly. This step is influenced by the different scales of the objective functions. Therefore, a proper weight assigning step is necessary. This step is not trivial and usually based on an engineering judgement. However, by using Automatic ϵ-LS [124], there is no need to do arithmetic operations on the objective values of the different objective functions. Therefore, this selection method is not influenced by the different scales of the objectives and there is no need to do weighting.

Algorithm 5.2: Pseudocode of Differential Evolution based on Automatic ϵ-Lexicase Selection

input : population size N
output: Final population X

$X \leftarrow$ initialize(N);
repeat
  $X' \leftarrow \emptyset$;
  for i in $1 \dots N$ **do**
    $\mathbf{x}_{lex} \leftarrow$ automatic-epsilon-lexicase-select(X);
    $\mathbf{x}_1, \mathbf{x}_2 \leftarrow$ random-select(X);
    $\mathbf{y} \leftarrow$ differential-mutate-crossover($\mathbf{x}_{lex}, \mathbf{x}_1, \mathbf{x}_2$);
    $\mathbf{y}' \leftarrow$ polynomial-mutate($\mathbf{y}$);
    $X' \leftarrow X' \cup \{\mathbf{y}'\}$;
  $X \leftarrow X'$;
until termination criteria are satisfied;
return Xs;

While there are multiple proposed DE variants, one of the most frequently used versions is DE/rand/1/bin [27]. In this method, three parents $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ are selected randomly to generate an offspring individual $\mathbf{y}$ based on (5.2). In (5.2), the subscript j means the j-th dimension of a vector. α is a scaling factor to control the mutation strength, and CR controls the binary crossover rate. j_{r_2} is a randomly selected dimension that ensures at least one dimension in the solution is mutated.

$$y_j = \begin{cases} x_{1,j} + \alpha \cdot (x_{2,j} - x_{3,j}), & r_1 \leq CR \text{ or } j = j_{r_2} \\ x_{1,j}, & \text{otherwise} \end{cases} \tag{5.2}$$

In this study, we propose DE/lexicase/1/bin in **Algorithm 5.2**. This method is similar to DE/rand/1/bin [27], however, we replace the first parent with a "good" individual $\mathbf{x}_{lex}$ selected by Automatic ϵ-LS [124]. This method performs mutation on the selected individual, by adding a differential vector between two random individuals.

We further modify the original DE algorithm [27] as follows. We omit the survival selection in the original DE procedure, since the LS can provide sufficient selection pressure. We also introduce the polynomial mutation after the differential mutation to prevent premature convergence. We provide the

procedure of our proposed method in **Algorithm 5.2**.

$$y_j = \begin{cases} x_{lex,j} + \alpha \cdot (x_{1,j} - x_{2,j}), & r_1 \leq CR \text{ or } j = j_{r_2} \\ x_{lex,j}, & \text{otherwise} \end{cases} \tag{5.3}$$

As pointed out in **Section 5.3**, when the number of the objectives in the problem is small, LS can become greedy. However, for a SHM problem, the available number of iterations is usually small, since it costs several minutes to hours per evaluation. Therefore, the greedy performance may not harm the optimization results in the SHM problems.

In the remaining sections, we abbreviate our method to Lex-DE.

5.5 Experiments

To test the proposed Lex-DE algorithm, we prepared two SHM problems, TS2N [109] and Volve [110]. The optimal parameters, also known as ground truth, for both problems are known for us (of course not used by the algorithms). The datasets generated and analysed during the current study are available in a public repository [1].

5.5.1 Test Problems

SHM models based on real world datasets, while unique, are broadly similar in their objectives and the challenges they present for optimization. We have selected two test problems which cover different model scales and complexities. Additionally the models are open and available for additional research.

The TS2N model [109] simulates a reservoir located in the Gulf of Mexico with a single production well. The model includes monthly production data for Oil, Gas and Water volumes from 1996 to 1999. Within the model, there are five geological layers with uniform properties. It is a real life example, albeit a simple one. It includes five objectives, namely FWPR, FOPR, FWPT, FOPT, and FGPR. The detail of the five objectives is provided in **Table 5.1**. The production totals (PT) metrics are time integrals of the production rates (PR). Including them as an objective emphasizes the requirement for the model to match the actual produced volumes of the field,

[1] https://github.com/Y1fanHE/lexde-subsurface-model

Table 5.1: Details of the Objectives in the TS2N problem

Objective	**Details**
FWPR	Field Water Production Rate
FOPR	Field Oil Production Rate
FWPT	Field Water Production Total $\int_0^t FWPR(t)dt$
FOPT	Field Oil Production Total $\int_0^t FOPR(t)dt$
FGPR	Field Gas Production Rate

Table 5.2: Details of the Objectives in the Volve problem

Objective	**Details**
Seis-mean	Seismic metric using Mean Attribute
Seis-spa	Seismic metric using SPA Attribute
P-F-14	Composite well F-14 fitness metric
P-F-12	Composite well F-12 fitness metric
P-F-15C	Composite well F-15C fitness metric

rather than just the production rates. This is a key requirement in history matching. Production totals tend towards increasing error overtime, which places more weight on matching the total produced volume at the end of the optimization period.

The TS2N model [109] has 13 model parameters that include horizontal and vertical permeability multipliers for each layer, the oil-water contact depth, porosity and the reservoir compressibility.

To increase the uncertainty of the TS2N model [109], we added randomized noise to the production rate history and reintegrated the rates to create production totals which are slightly different to the truth case. The noise is not correlated between objectives, and this creates a small degree of trade-off in the objective function that would otherwise be absent.

The Volve field is an open dataset [110]. This problem also includes five objectives, namely Seis-mean, Seis-spa, P-F-14, P-F-12, and P-F-15C. The detail of the five objectives are provided in **Table 5.2**. The Volve model [110] has 63 model parameters to generate a large search space. The parameters of the model include the oil-water contact, fault transmissibilities, region and zone permeability and porosity multipliers and aquifer volume. Unlike the

TS2N problem, this problem includes data from multiple wells and multiple seismic.

5.5.2 Experimental Methods

We compared Lex-DE with two MOEAs, namely RVEA [21] and NSGA-II [20]. Before the formal experiments, we tuned several important parameters but set the rest based upon experience in prior studies [20, 21] except for population size. For each algorithm, we run five repetitions.

We set the total number of evaluations as 1500 for all three algorithms on TS2N, and 2000 on Volve. The population size is set as 20 (75 generations on TS2N and 100 generations on Volve). For Lex-DE, we set the scaling vector α=0.5 and the crossover rate CR=0.5 without tuning. The mutation rate p_m is set to $1/n$ (n is the dimension of the problem). For NSGA-II, the crossover rate p_c is set to 0.9 (tuned from {0.6, 0.7, 0.8, 0.9, 1.0}). The mutation rate p_m is set to $1/n$ (n is the dimension of the problem). For RVEA, the crossover rate and mutation rate are the same as NSGA-II. We generate 15 weight vectors based on Das-Dennis method [37]. We set the rest parameters, α=2.0 and f_r=0.1, based on the original RVEA paper [21].

We include two numerical metrics of performance, average distance to the ground truth and the difference on set coverage. Before computing the metrics, we scaled the objective values into $[0, 1]$ based on the non-dominated solutions over all evaluations in the five repetitions of the three algorithms.

- **Average distance to the ground truth ($\bar{d}_{gt}$).** This metric shows the scaled Euclidean distance between the non-dominated solution set A and the ground truth $\mathbf{x}^*$ in the parameter space.

$$\bar{d}_{gt} = \frac{\Sigma_{\mathbf{x}\in A}||\mathbf{x} - \mathbf{x}^*||}{|A|} \tag{5.4}$$

- **Difference on set coverage ($\Delta C(A, B)$).** Let A and B be two non-dominated sets, the set coverage $C(A, B)$ is defined as the percentage of the solutions in B that are dominated by at least one solution in A. When computing this set coverage, the non-dominated solution set of every method is the non-dominated set of union of the solutions from five repetitions. The difference on set coverage is computed as in (5.5).

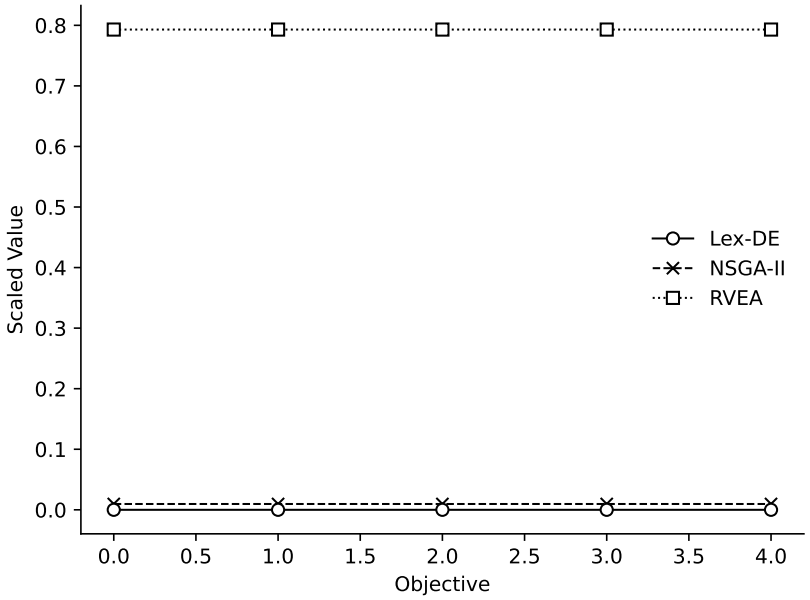

Figure 5.2: Parallel Objective plots for the best run on TS2N. Each line corresponds to the five (scaled) objective values of a non-dominated solution.

A positive value of $\Delta C(A, B)$ shows that A is better than B considering all the objectives.

$$\Delta C(A, B) = C(A, B) - C(B, A) \tag{5.5}$$

$$C(A, B) = \frac{|\{u \in B | \exists v \in A : v \prec u\}|}{|B|} \tag{5.6}$$

5.5.3 Experimental Results

Table 5.3 and **Table 5.5** provide the average distance to the ground truth $\bar{d}_{gt}$ on the TS2N and Volve problems. On both problems, the solution set found by Lex-DE is closest on average to the ground truth. **Table 5.4** and **Table 5.6** shows the difference on set coverage on the two problems. On both problems, ΔC(Lex-DE,RVEA) and ΔC(Lex-DE,NSGA-II) are positive, indicating that a high proportion of solutions by Lex-DE dominates the solution sets of both RVEA and NSGA-II.

Table 5.3: Average distance to the ground truth $\bar{d}_{g.t.}$ of three algorithms on TS2N

Method	Best	Median	Worst	Mean	Std.
Lex-DE	0.157	0.339	0.623	**0.352**	0.170
RVEA	0.684	0.719	0.953	0.793	0.123
NSGA-II	0.256	**0.319**	0.498	0.359	0.099

Table 5.4: Difference on set coverage $\Delta C(A, B)$ of three algorithms on TS2N

A	B	ΔC(A,B)
Lex-DE	RVEA	100%
Lex-DE	NSGA-II	100%
RVEA	NSGA-II	-100%

Table 5.5: Average distance to the ground truth $\bar{d}_{gt}$ of three algorithms on Volve

Method	Best	Median	Worst	Mean	Std.
Lex-DE	2.102	**2.368**	3.076	**2.464**	0.376
RVEA	2.947	3.127	3.222	3.086	0.110
NSGA-II	2.383	2.764	2.840	2.675	0.187

Table 5.6: Difference on set coverage $\Delta C(A, B)$ of three algorithms on Volve

A	B	ΔC(A,B)
Lex-DE	RVEA	100%
Lex-DE	NSGA-II	95%
RVEA	NSGA-II	-98%

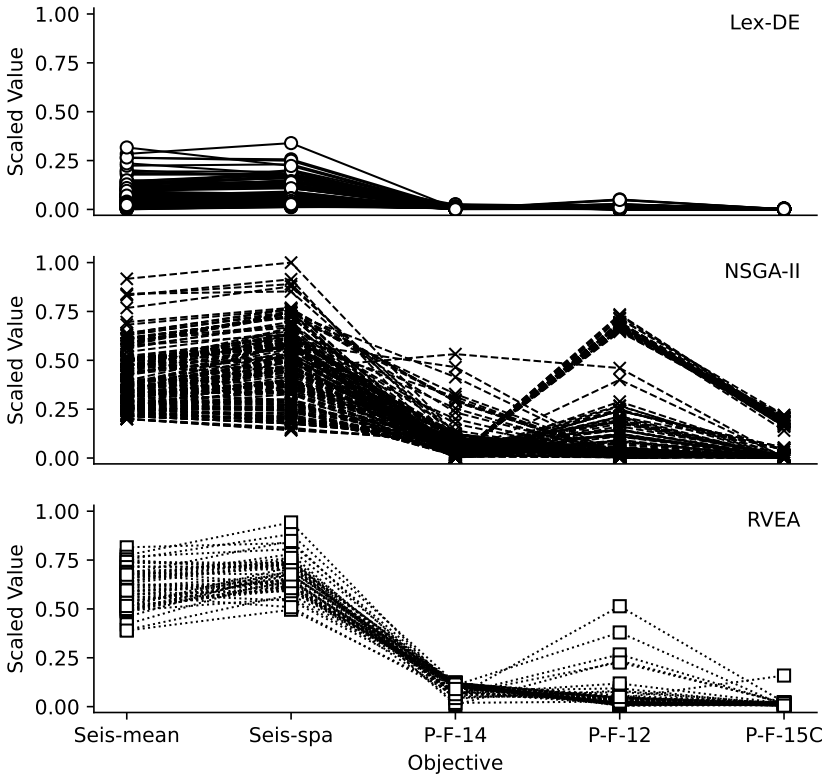

Figure 5.3: Parallel objective plot of non-dominated solutions in the best run on Volve. Each line corresponds to the five (scaled) objective values of a non-dominated solution. The solutions generated by Lex-DE hold smaller objective values and are more concentrated, compared with the solutions generated by NSGA-II and RVEA.

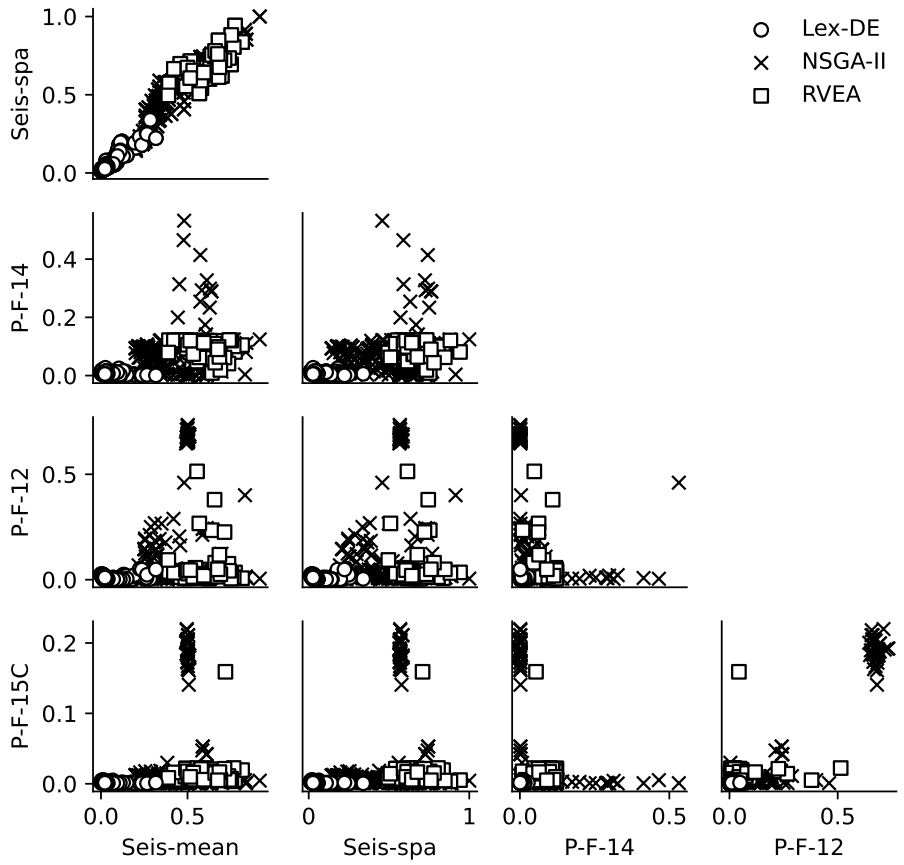

Figure 5.4: Scatter plot of the objective values of non-dominated solutions in the best run on Volve. The solutions generated by Lex-DE hold smaller objective values and are more concentrated, compared with the solutions generated by NSGA-II and RVEA.

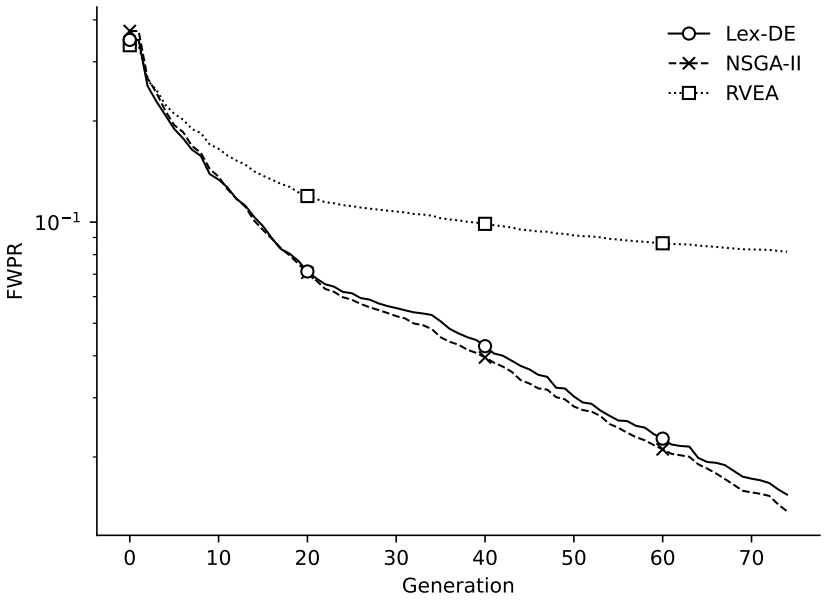

Figure 5.5: FWPR by generations (y-axis in log scale) on TS2N

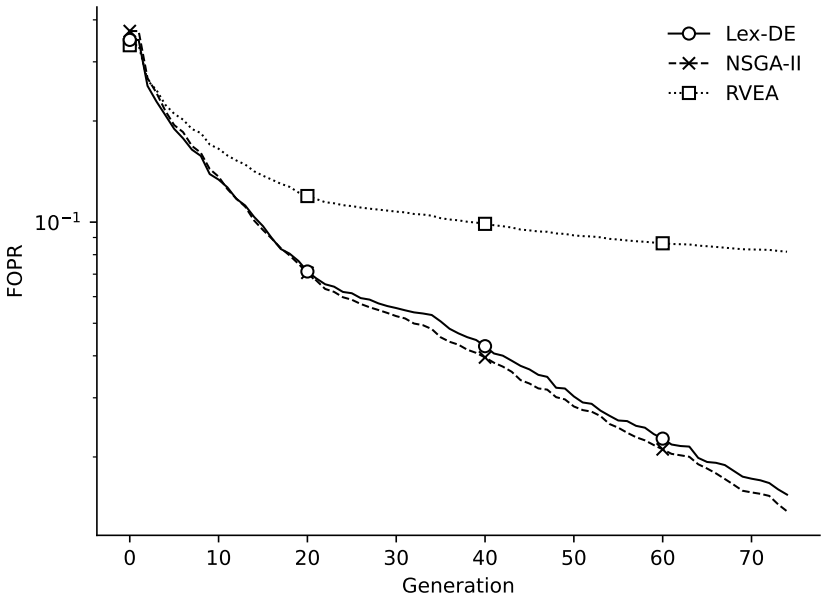

Figure 5.6: FOPR by generations (y-axis in log scale) on TS2N

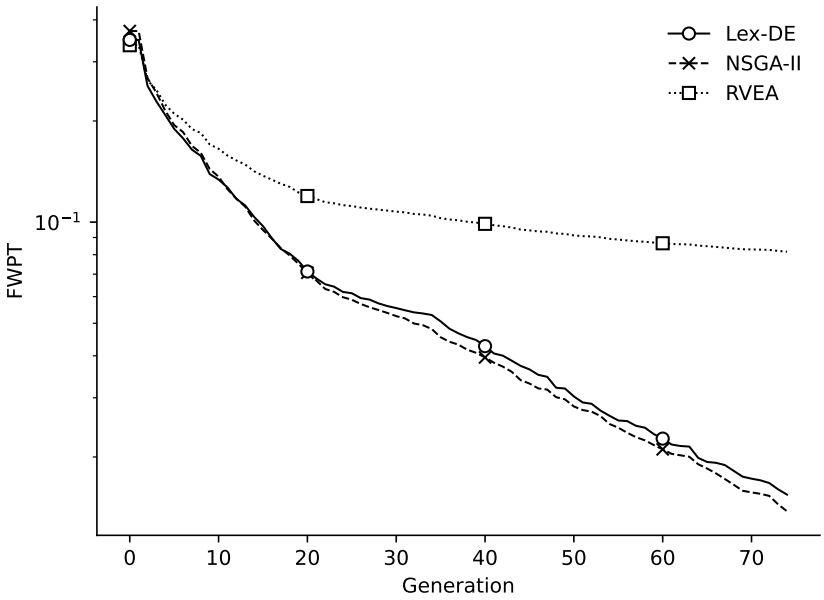

Figure 5.7: FWPT by generations (y-axis in log scale) on TS2N

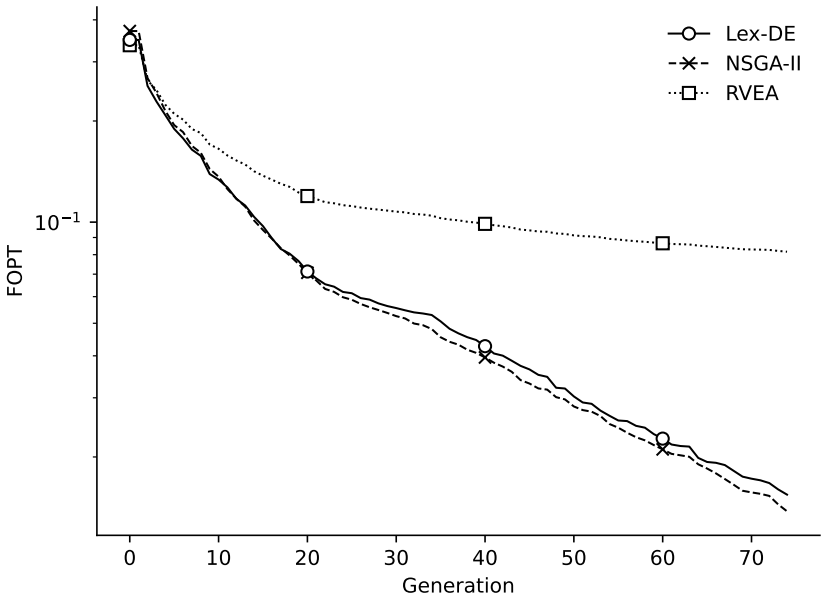

Figure 5.8: FOPT by generations (y-axis in log scale) on TS2N

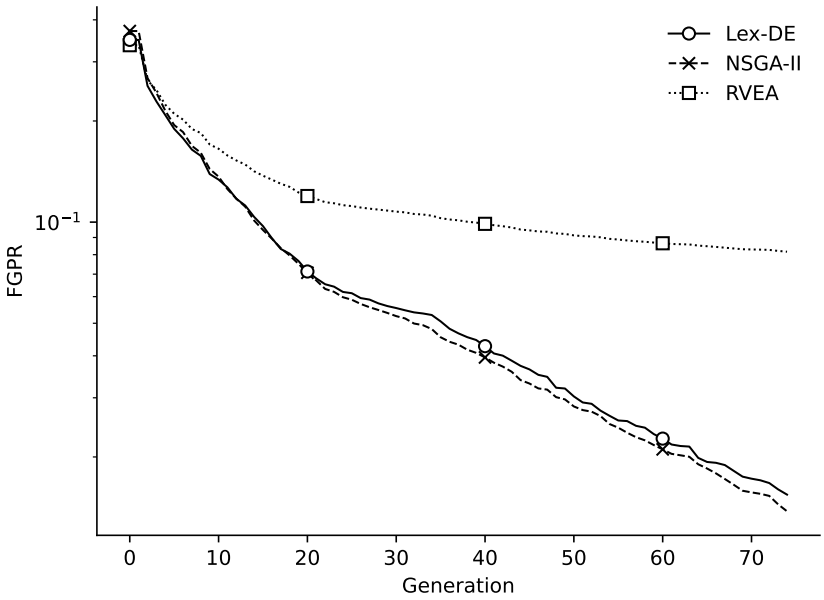

Figure 5.9: FGPR by generations (y-axis in log scale) on TS2N

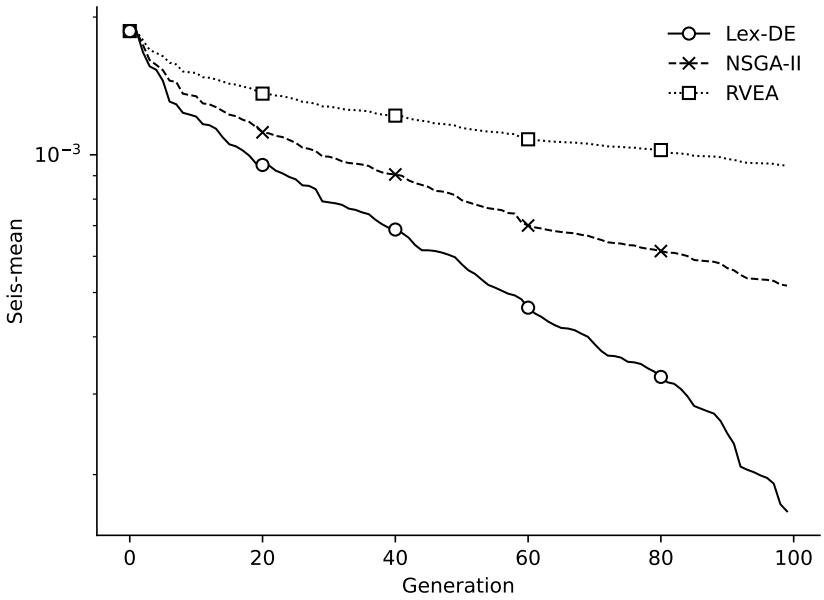

Figure 5.10: Seis-mean by generations (y-axis in log scale) on Volve

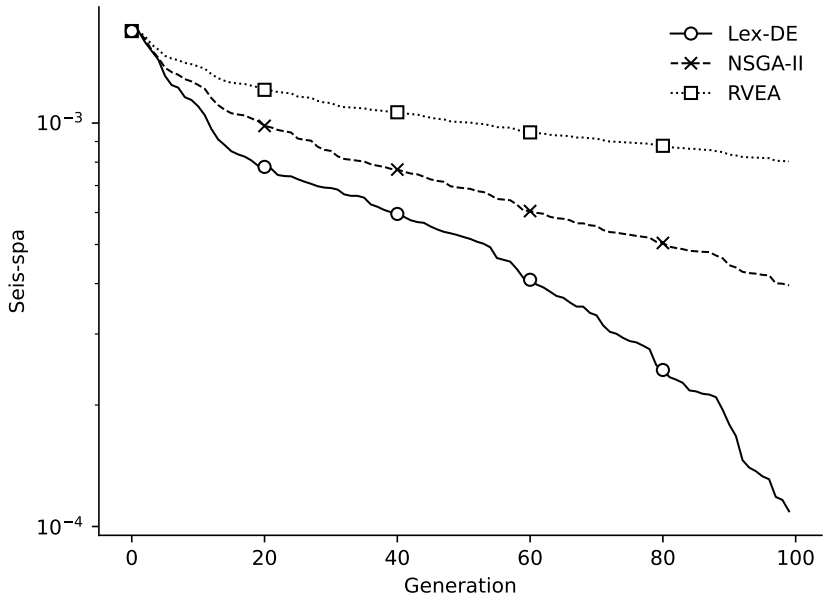

Figure 5.11: Seis-spa by generations (y-axis in log scale) on Volve

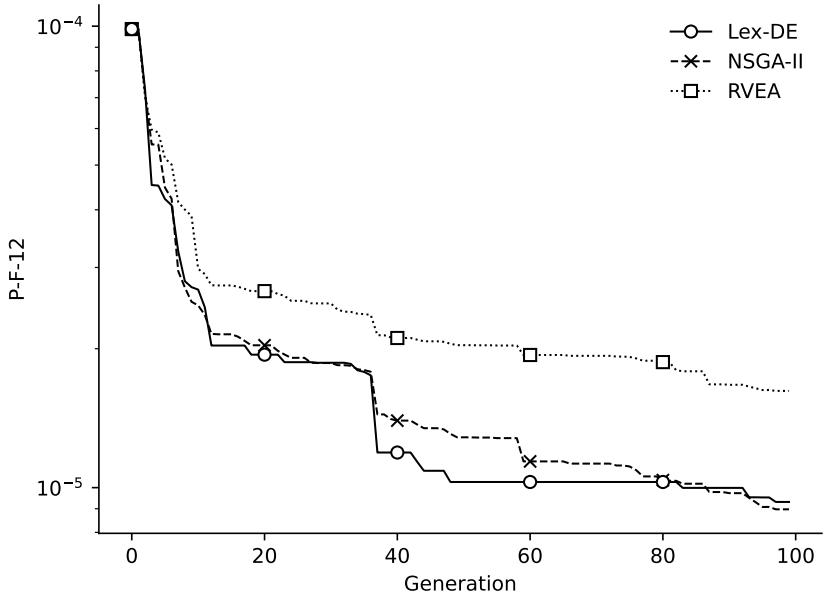

Figure 5.12: P-F-12 by generations (y-axis in log scale) on Volve

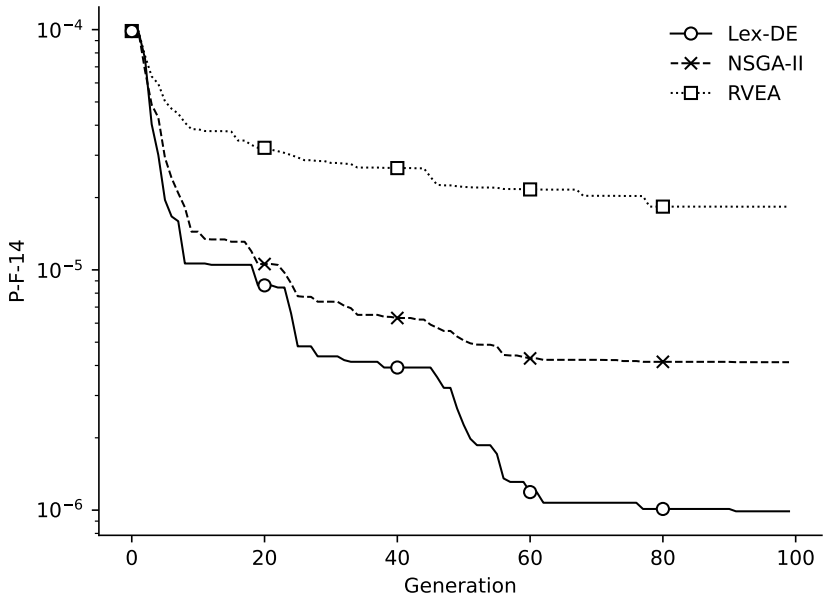

Figure 5.13: P-F-14 by generations (y-axis in log scale) on Volve

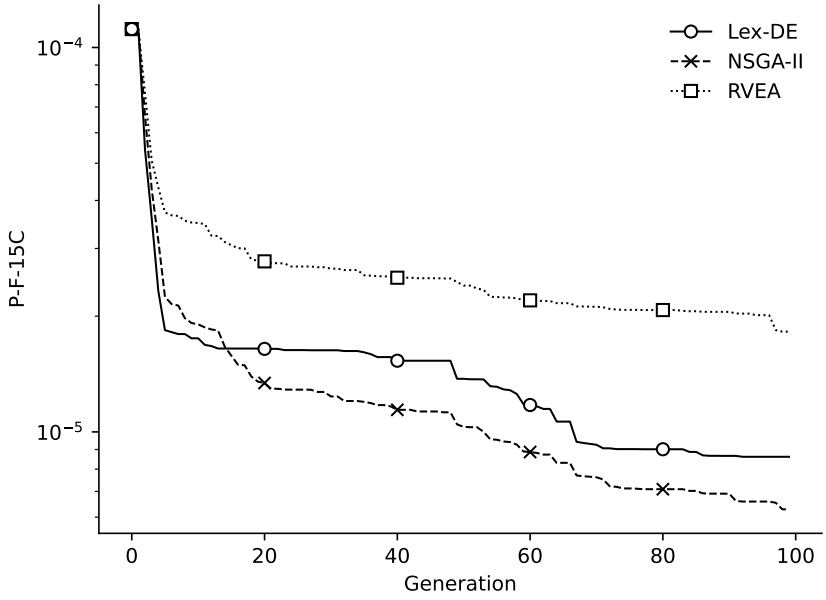

Figure 5.14: P-F-15C by generations (y-axis in log scale) on Volve

Figure 5.5 to **Figure 5.9** and **Figure 5.10** to **Figure 5.14** show the change in the best objective values by generation on the TS2N and Volve problem, indicating the convergence of the algorithms on these problems. The objective values are scaled into $[0, 1]$ based on all the solutions found in five repetition of three algorithms. These figure shows that Lex-DE and NSGA-II approaches the best solutions faster than RVEA in TS2N, and that on Volve Lex-DE finds better solutions faster on seis-mean, seis-spa and P-F-14, while NSGA-II finds better solutions faster on P-F-15C.

Figure 5.2 and **Figure 5.3** provide the parallel coordinate graph of the TS2N and Volve problems, respectively. These figures show the non-dominated solutions over all evaluations in the run with best average distance to the ground truth. In the graph, a non-dominated solution is represented by five points connected by a line. The five points show the five objective values (scaled into $[0, 1]$) of this solution. We find that on the TS2N problem (**Figure 5.2**), there is only one non-dominated solution for each algorithm. This shows the lack of trade-off between the objectives in TS2N.

On TS2N, the non-dominated solution of Lex-DE is slightly better than that of NSGA-II, but much better than that of RVEA. On the Volve problem, more than one non-dominated solutions have been found for all three methods. The parallel objectives plot in **Figure 5.3** shows that Lex-DE generates solutions with clearly better values than the other methods for the objectives Seis-mean and Seis-spa, and somewhat better values on P-F-14, P-F-12, and P-F-15. This shows that the proposed method finds solutions that are generally good across all objectives, including objectives of different nature (seismic and well) when compared to the other two MOEAs.

To better understand these results, **Figure 5.4** provides a scatter plot of the solutions in **Figure 5.3**. This figure shows how the non-dominated solutions of the Lex-DE algorithm are distributed in a central area in the non-dominated front. However, for the other two MOEAs (especially the NSGA-II), their solution sets are spread over a larger area, including some "extreme points" that perform sub-optimally on some objectives.

5.6 Discussion

In **Section 5.5.3**, we find that the performance of Lex-DE and NSGA-II on the simple TS2N problem are close. However, on the harder Volve problem, Lex-DE has a better result compared to NSGA-II and RVEA. In this section,

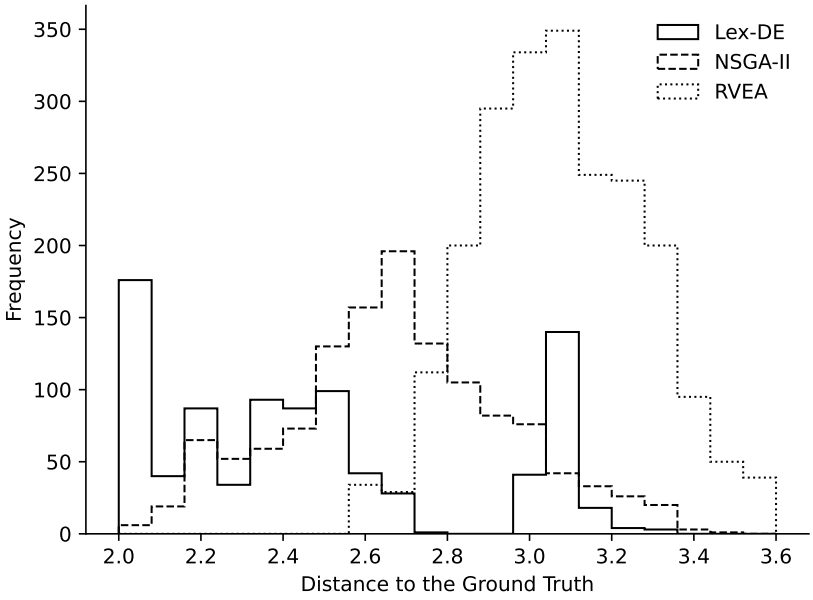

Figure 5.15: Frequency of scaled Euclidean distance between non-dominated solutions to the ground truth in the parameter space. Lex-DE holds more solutions that are close to the ground truth in terms of the decision space. However, there is one run of Lex-DE might get into local optima.

we provide further discussion based on the Volve results.

5.6.1 Distribution of distance to the ground truth

Lex-DE uses the LS. Therefore, the solutions are pressured to perform well on all objectives. This feature brings two main benefits.

1. For the final solution set, LS centralizes the solutions in a small area. To illustrate this point, in **Figure 5.15**, we show the histogram of the scaled distance between every non-dominated solution and the ground truth in the parameter space in all repetitions. We clearly find that Lex-DE generates more solutions than the other two MOEAs in the area that is close to the ground truth (distance less than 2.2). In real-world engineering tasks, the final set of models when using Lex-DE as an optimization tool is closer to the truth and contains less nonphysical

Table 5.7: Difference on set coverage $\Delta C(A, B)$ in prediction of three algorithms on Volve

A	**B**	**ΔC(A,B)**
Lex-DE	RVEA	-45%
Lex-DE	NSGA-II	-28%
RVEA	NSGA-II	-23%

results.

2. For the evolutionary process itself, a centralized solution set usually holds a stronger exploitation, and thus the algorithm can converge to a better result in a shorter amount of time. This is advantageous for SHM problems where evaluations a usually limited due to computation cost.

However, the disadvantages are also obvious. Focusing on a specific area may lead to many solutions within a local optimum. For example, in **Figure 5.15**, there are solutions of Lex-DE distributed between 3.0 and 3.2. We suspect this run falls within a local optimum, and thus results are worse than other runs.

5.6.2 Performance in the prediction period

In the real world, the subsurface model is used to do forecasting on the field production. We perform prediction based on every individual generated during the optimization (2000 individuals per run). We only compute the misfit on P-F-12 and P-F-14, since the other three objectives are not available in the prediction period. The difference on set coverage based on the fitness in prediction is provided in **Table 5.7**. Regarding the set coverage in prediction and the evolution of the prediction fitness (**Figure 5.16** and **Figure 5.17**), NSGA-II performs the best. We consider the following three possible reasons.

- Our Lex-DE performs much better on the two seismic related objectives. However, they are not available in the prediction. In the optimization period (**Figure 5.3**), NSGA-II also generated several solutions that are good in P-F-14 and P-F-12. Therefore, it is not strange for NSGA-II to have a good prediction result.

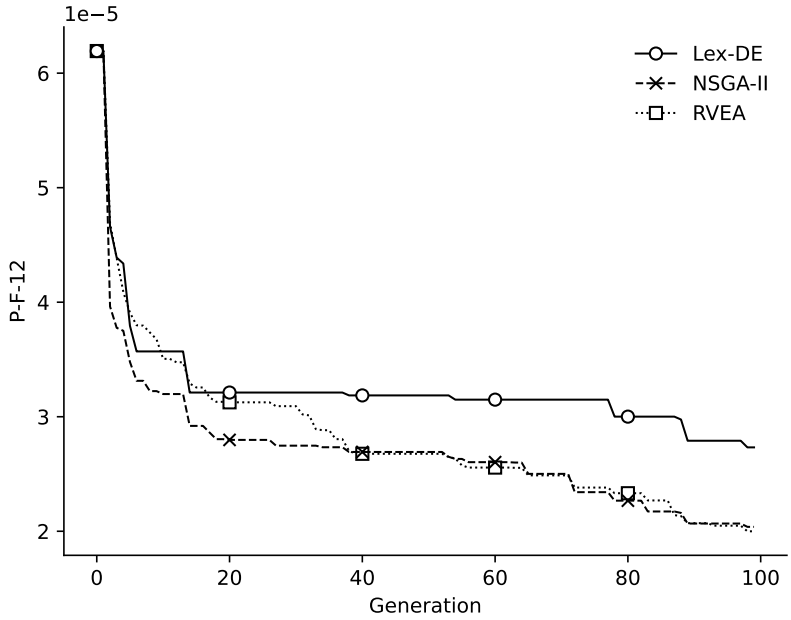

Figure 5.16: P-F-12 by generations (y-axis in log scale) in prediction on Volve

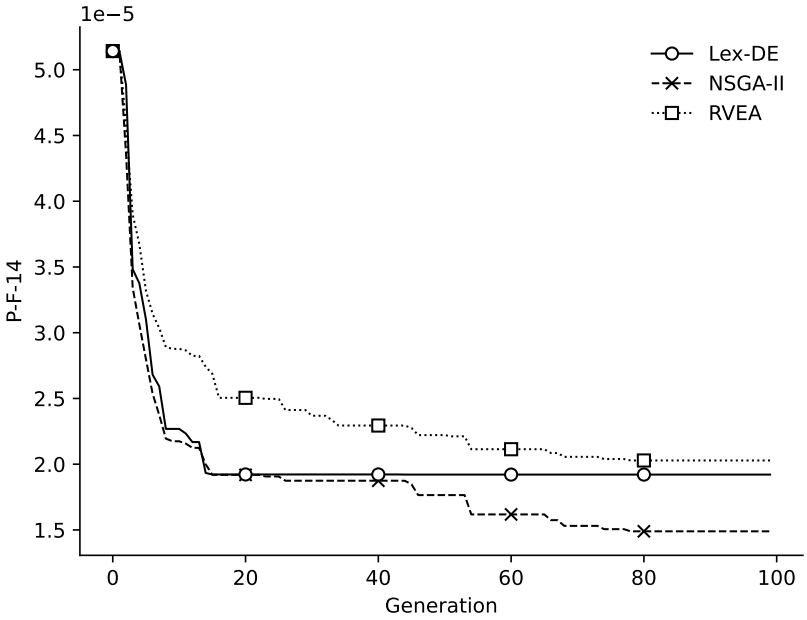

Figure 5.17: P-F-14 by generations (y-axis in log scale) in prediction on Volve

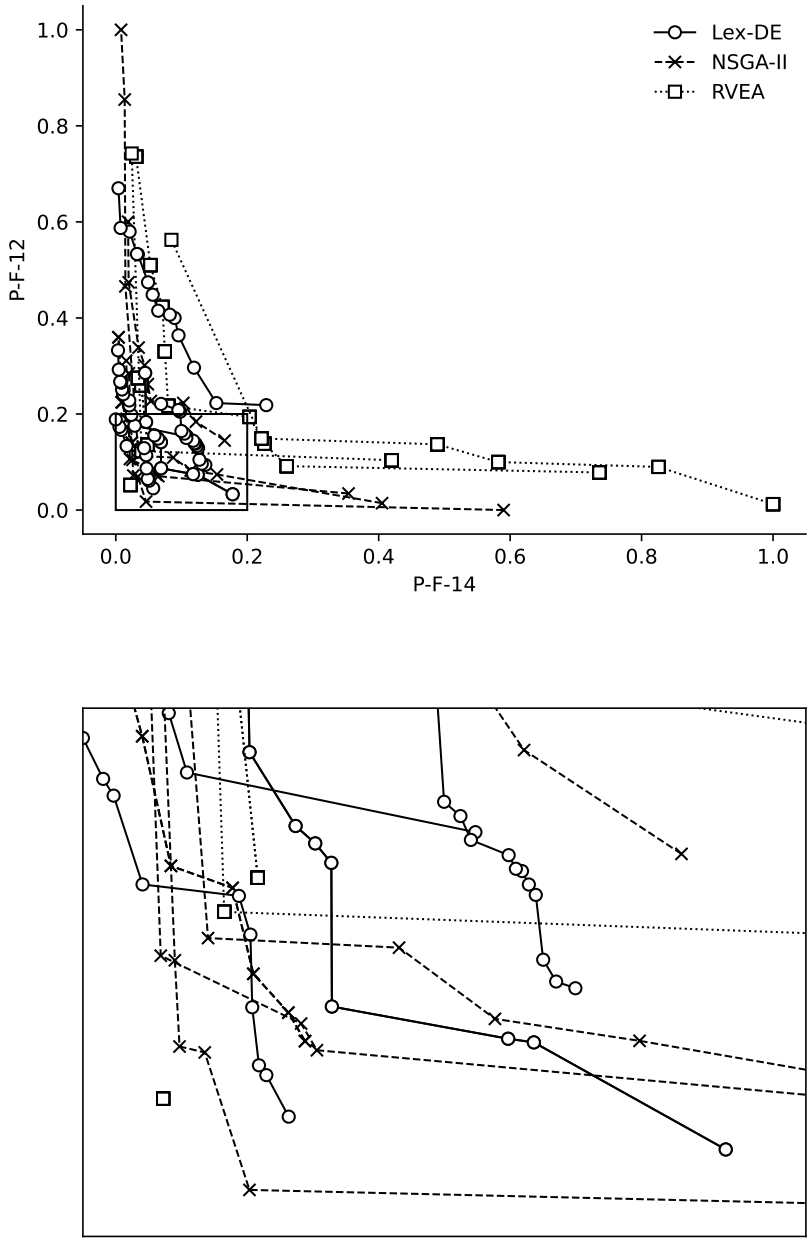

Figure 5.18: Scatter plot of non-dominated solutions in prediction period on Volve. The lower subplot is the zoom-in of the rectangle area in the upper subplot.

- **Figure 5.18** shows the non-dominated solutions in the prediction period of three algorithms in all five runs, as well as a zoom-in graph of the center part of the non-dominated front. The solutions in the same run are connected by a line. In most runs, NSGA-II and RVEA generate several solutions that perform well on some objectives, but sub-optimally on the others. These "sub-optimal" solutions may not be dominated by any of the solutions of Lex-DE, but they are less useful in the SHM problem.

- Our method may overfit the problem during the optimization phase. Though all three algorithms do not contain any explicit way to overcome overfitting, NSGA-II and RVEA tend to maintain a more diverse solution set. This may lead to the better results in the prediction period. To enhance the diversity of Lex-DE without generating sub-optimal solutions, we can use other strong global mutation methods or restart strategies, and keep the LS.

5.7 Conclusions of the Case Study

In this study, we have introduced the Lexicase Selection method [19, 124] and proposed the Differential Evolution based on Automatic ϵ-Lexicase Selection algorithm to solve the history matching problems. We have explained how LS performs GKT. We have compared the proposed algorithm with two other literature methods, the NSGA-II [20] and the RVEA [21], on two real-world examples [109, 110]. The results have shown the superiority of the proposed method with better optimization results (i.e., positive difference on set coverage and smaller average distance to the ground truth) and a more centralized solution set. What is more, we have found that this centralized set usually provides more solutions close to the ground truth in the parameter space. For an engineering problem, this feature generates a final ensemble of models which better characterize the true model and parameter uncertainty.

Despite the above advantages, this method sometimes falls into local optimum. What is more, the prediction performance (difference on set coverage in prediction) of Lex-DE is not as good as in the optimization phase. This may be caused by the following reasons: 1) some objectives are not available in the prediction period; 2) the set coverage may be affected by the extreme points; 3) our Lex-DE may get overfitting in the optimization phase. In the future, we are going to increase the diversity of Lex-DE by applying strong global mutation methods. This can solve the local optimum and the

overfitting issue, without generating sub-optimal solutions.

In addition, the main outcome of this paper is developed from questioning whether we should model the history matching problem as Multi-Objective Optimization. In the recent optimization literature, Multi-Form Optimization [24] has been proposed to reconcile multiple alternate formulations of a single target task of interest. Part of our future research will consider the history matching problem as a Multi-Form Optimization task.

Chapter 6

Knowledge-Driven Program Synthesis

In this chapter, we show a practice of our idea on the Adaptive Genetic Knowledge Transfer system in **Chapter 4**. We introduce Knowledge-Driven Program Synthesis (KDPS) as a variant of the Program Synthesis (PS) task that requires the agent to solve a sequence of program synthesis problems. In KDPS, the agent should use knowledge from the earlier problems to solve the later ones. We propose a novel method based on PushGP to solve the KDPS problem, which takes subprograms as processed knowledge. The proposed method extracts subprograms from the solution of previously solved problems by the Even Partitioning (EP) method and uses these subprograms to solve the upcoming programming task using Adaptive Replacement Mutation (ARM). We call this method PushGP+EP+ARM. With PushGP+EP+ARM, no human effort is required in the knowledge extraction and utilization processes. We compare the proposed method with PushGP, as well as a method using subprograms manually extracted by a human. Our PushGP+EP+ARM achieves better train error, success count, and faster convergence than PushGP. Additionally, we demonstrate the superiority of PushGP+EP+ARM when consecutively solving a sequence of six PS problems.

6.1 Introduction of the Case Study

Program Synthesis (PS) are techniques that automatically compose computer programs to solve a certain task. PS is useful in fields such as automatic bug fixing, automatic program completion, and low-level code development. PS is a key issue in Artificial General Intelligence [127]. Genetic Programming (GP) [50] is an Evolutionary Algorithm that searches for computer programs by selecting and updating a population of program candidates. Some

GP variants [23, 128, 129] can solve problems in a famous PS benchmark suite [130] efficiently.

However, the difference between GP and a human programmer is still obvious. As an Evolutionary Algorithm, GP heavily utilizes random sampling; while a human programmer does not write random programs. Humans write programs based on their knowledge, either the domain knowledge about the problem or the programming skills from previous experiences.

Recently, several studies [131, 132, 16, 15, 133] have attempted to incorporate knowledge in PS, improving the synthesis performance. However, some of these methods have drawbacks in requiring extra information [131, 132] and human efforts [133].

In our prior study, we proposed the Adaptive Replacement Mutation (ARM) [133]. The ARM is a mutation method designed for a well-known GP variant called PushGP [23]. ARM uses subprograms from an archive as knowledge, automatically selecting useful subprograms from the archive according to the search history. Although the ARM provides a way to use existing knowledge from an archive, the archive itself was made by a human. Moreover, it is questionable whether the subprograms written by humans are included in the programs generated by PushGP.

In this study, we focus on the task where a GP is required to solve a sequence of PS problems. The GP should learn genetic knowledge from each problem in the sequence and apply this genetic knowledge to improve its performance in later problems. Ideally, this procedure should not require human interference or extra external information. We call this task the Knowledge-Driven Program Synthesis (KDPS) problem (**Section 6.4**).

To this end, we propose a novel method to solve the KDPS problem based on PushGP [23]. The overall design is similar to our Adaptive Genetic Knowledge Transfer system in **Chapter 4**. This method takes subprograms as processed knowledge. The proposed method consecutively solves programming tasks, extracts subprograms from the solution of solved problems, and uses subprograms to solve an upcoming problem. To extract subprograms, we propose Even Partitioning (EP) which divides a solution into several parts with equal lengths. To use these subprograms, we apply ARM [133]. We name our method PushGP+EP+ARM. The details of the proposed methods, including EP and ARM, are given in **Section 6.4**.

We analyze the proposed method in two KDPS tasks. The "compos-

ite task" (**Section 6.5**) includes three "composite" PS problems. For each problem, PushGP+EP+ARM prepares the knowledge archive based on the component problems. The "sequential task" (**Section 6.6**) has six problems that must be solved in sequence, including the composite and component problems of the previous "composite task". PushGP+EP+ARM updates the knowledge archive at each step of the sequence. PushGP+EP+ARM achieves a better overall success rate and convergence speed in the composite task, and in the later stages of the sequential task, showing that it can create a useful knowledge archive. However, the comparison with a human-curated archive shows that there is still room for improvement.

Our main contributions are as follows.

1. We introduce a new type of task called the KDPS problem. KDPS includes a sequence of single PS problems. The agent is required to solve the single PS problems, extract knowledge, and use it in the later PS problems.
2. We propose EP to extract subprograms from the solution of a solved problem. We propose PushGP+EP+ARM to solve KDPS problems based on EP and our previous work on ARM [133].
3. Our implementation of the proposed method based on PyshGP [134] and experimental scripts are in an online repository [1].

6.2 PushGP

In **Section 2.4**, we have introduced Koza's tree-based GP [50]. Despite the tree-based GP, several variants of GP [135, 67, 136, 23] have been applied to solve PS problems. Among these variants, we highlight PushGP [67, 23, 129], which generates programs based on a Turing-complete language called Push. PushGP supports generating computer programs with multiple data types and control flows, such as loop and recursion.

6.2.1 Push Language

Precisely, Push is the name of a family of languages that are developed for Evolutionary Computation (EC) to generate computer programs. Versions

[1] https://github.com/Y1fanHE/ssci2022

of Push have been implemented by many different researchers in different languages (Clojure, Python, Java, Julia, etc.). New versions generally add some new features, delete some old features, and change name conventions.

Here we introduce the version that is used in our experiments. It is embedded in a Python implementation of PushGP called PyshGP [134].

In PyshGP [134], there are two types of Push programs, linear Push programs and normal Push programs. The linear Push programs are exactly the genomes that are modified through evolutionary process. However, the Push interpreter cannot directly run these linear Push programs before they are translated into normal Push programs. A vital difference between the two types of the programs is that normal Push programs allow nested structures called "code blocks" while linear Push programs implement equivalent programs with extra instructions such as "closer".

For example, the following two programs are exactly equivalent; however, the first one uses linear Push and the second one uses normal Push.

1. `input_0 exec_dup int_inc exec_dup int_inc closer closer`
2. `input_0 exec_dup (int_inc exec_dup (int_inc))`

After translated from the genome (linear Push program), a normal Push program is run by an interpreter using multiple stacks of different data types based on the following steps.

1. To execute an instruction, the interpreter pops the required arguments from the corresponding stacks.
2. After executed the instruction, the results are pushed to the corresponding stacks.
3. If the interpreter cannot find enough arguments from the stacks, the instruction will be skipped.

Based on the third rule, any Push program is valid to run. Therefore, we do not need to consider the constraint handling techniques when developing new mutation or crossover methods.

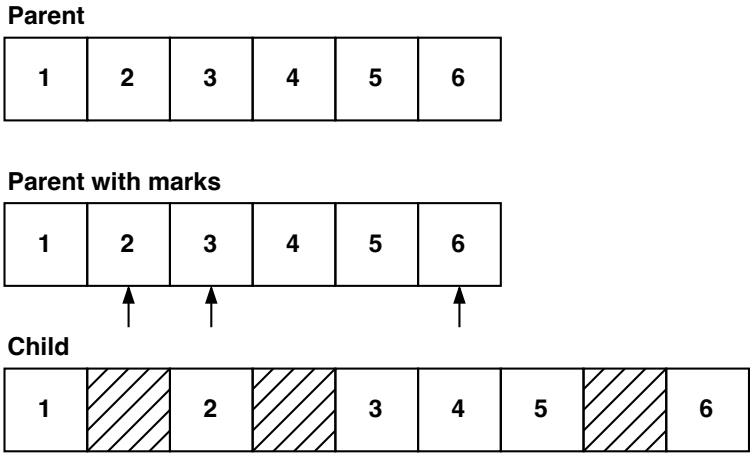

Figure 6.1: Addition Mutation in PushGP [23]. The Addition Mutation traverses the parent solution and adds a random instruction before the current instruction with a probability.

6.2.2 PushGP with Uniform Mutation by Addition and Deletion

Helmuth *et al.* have proposed a variant of PushGP [23] that employs random initialization, Lexicase Selection (**Section 5.3**), and Uniform Mutation by Addition and Deletion (UMAD).

The UMAD contains two steps: Addition Mutation (AM) and Deletion Mutation (DM). AM takes one individual as the parent. AM traverses all instructions in the genome and inserts a random instruction before the current instruction with a probability. Similarly, DM takes one individual as the parent as well. DM traverses all instructions in the genome and deletes the current instruction with a probability (not necessary to be equal with the probability in AM). **Figure 6.1** and **Figure 6.2** illustrates the examples of AM and DM.

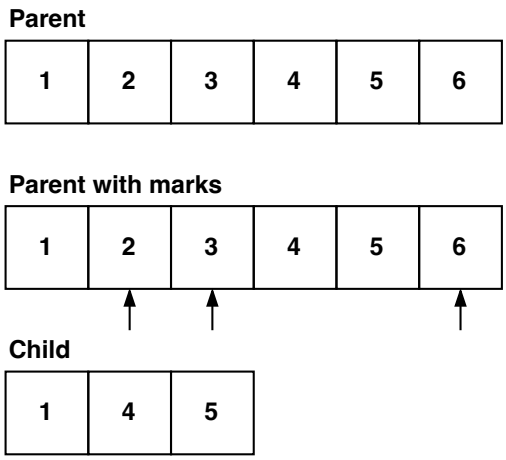

Figure 6.2: Deletion Mutation in PushGP [23]. The Deletion Mutation traverses the parent solution and deletes the current instruction with a probability.

6.3 Incorporating Knowledge in Program Synthesis

Several recent works have attempted to incorporate knowledge in PS [132, 131, 16, 15, 133, 137]. Some studies require extra information, such as text description [131] and human-written code [132, 133, 137]. Notably, these prior studies try to use human knowledge with GP, and thus cannot be considered as Genetic Knowledge Transfer (GKT) methods, since GKT happens among several Evolutionary Algorithms.

Helmuth *et al.* have proposed to transfer the instructions from the solution of other problems to construct the instruction set of PushGP [16]. Wick *et al.* have proposed to use the whole individuals of a problem as a part of the initial population when solving similar problems [15]. Both studies [16, 15] could be considered as GKT.

Recently, we came up with a mutation method that allows using subprograms in a prepared archive with PushGP [133]. Compared with the study by Helmuth [16], a subprogram can capture more information from the solution than a single instruction. Compared to Wick *et al.*'s study [15], the way to use an external archive might be more efficient when the number of the

past problems is large. However, in our prior study [133], the subprogram is extracted by hand from human-written solutions. This step is non-trivial and requires a lot of human effort. Also, it is questionable whether the programs generated by PushGP would look similar to human written ones.

In this case study, we aim to apply GKT methods to solve a sequence of PS problems, so that the algorithm could improve itself through problem-solving. We suggest that this idea is helpful for generating complex programs. In the next section, we introduce the problem description of Knowledge-Driven Program Synthesis and demonstrate a simple algorithm based on PushGP to solve it.

6.4 Knowledge-Driven Program Synthesis

6.4.1 Problem Description

We introduce a type of problem where a GP algorithm is required to solve a sequence of PS problems. These PS problems are at a large amount and not necessary to be similar. Moreover, when solving a new task, the algorithm should use the genetic knowledge learned from previously solved tasks. We call this type of problem Knowledge-Driven Program Synthesis (KDPS). Clearly, KDPS is an application of the sequential problem-solving that we have described in **Section 4.2.1**.

The formalization of KDPS is presented in (6.1). T^j is one of the M PS tasks to solve, containing N_j pairs of I/O. $\hat{\mathbf{p}}^j$ and S^j are the solution program and the genetic knowledge that extracted from the dynamics of solving T^j, respectively. T^j is solved by Solve$(\cdot)$ based on the genetic knowledge from the previously solved problems $S^{j-1} \cup \ldots S^0$. S^0 is the initial genetic knowledge before solving the first problem T^1. By default, S^0 is empty.

$$
\begin{aligned}
& \hat{\mathbf{p}}^j, S^j \leftarrow \text{Solve}(T^j | S^{j-1} \cup \cdots \cup S^0) \\
s.t. \quad & T = \{T^1, \ldots, T^M\} \\
& T^j = \{(in_1^j, out_1^j), \ldots, (in_{N_j}^j, out_{N_j}^j)\} \\
& S^0 = \emptyset
\end{aligned}
\tag{6.1}
$$

As an initial step, we use subprograms of the final solution program $\hat{\mathbf{p}}^j$ as the processed knowledge. The subprograms hold partial information about

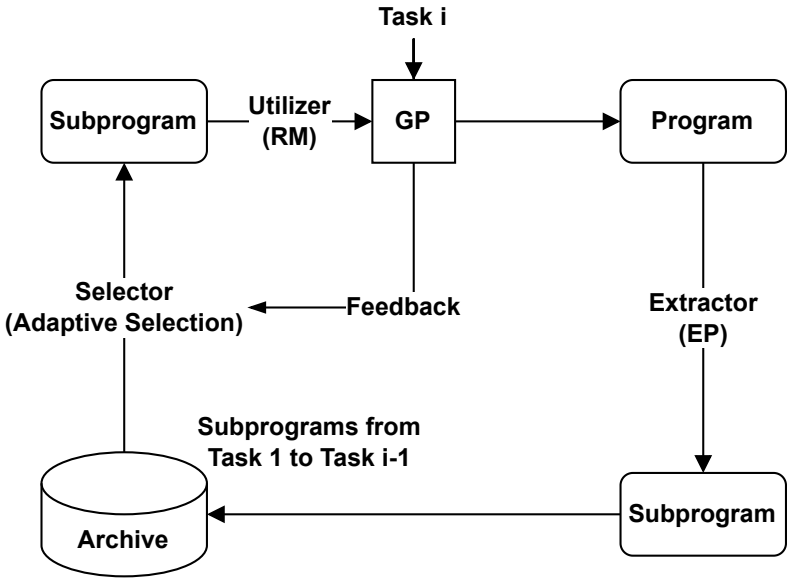

Figure 6.3: Knowledge-Driven Program Synthesis (KDPS) system. The KDPS system is an implementation of the idea of Adaptive Genetic Knowledge Transfer system. The KDPS system consecutively solves Program Synthesis tasks, extracts sub-programs as genetic knowledge, and uses them in the future tasks.

the original program. Moreover, any sequence of Push instructions is valid to run. Therefore, we can easily take a subprogram and use it to mutate another Push program.

$$S^j \leftarrow \text{Extract}(\hat{\mathbf{p}}^j) \tag{6.2}$$

6.4.2 Overview of the System Design

To solve KDPS problems, we propose a method based on PushGP [23] to consecutively solve programming tasks, extract subprograms from the final solutions (processed knowledge), and utilize these subprograms in the next problem. This entirely automated method is based on Adaptive Genetic Knowledge Transfer (AGKT) system in **Section 4.3**.

Figure 6.3 illustrates the AGKT system for KDPS problem. At the moment, the system has solved i-1 PS problems. The system is going to solve the i-th PS problem through the following steps.

1. The system selects subprograms from the archive that contains subprograms from all the previous tasks with an adaptive strategy.
2. The selected subprograms are utilized by Replacement Mutation (RM) in PushGP algorithm to reproduce new individuals.
3. The feedback in fitness improvement will be sent back to the adaptive selector to update the selection probability of every subprogram.
4. After PushGP finds the final solution, the extractor called Even Partitioning (EP) will extract subprograms from this solution program and store in the archive.

We leave the implementation of the filter which creates a subset of the archive as a future work.

6.4.3 Even Partitioning

EP is a simple method that divides a genome (linear Push program) into n parts with equal lengths. For example, a program with 15 instructions is divided into subprograms with lengths of (3, 3, 3, 3, 3) if $n = 5$; the same program is divided into subprograms with lengths of (4, 4, 4, 3) if $n = 4$.

Before dividing the solution program, a simplification operation is performed to remove the redundant instructions (i.e., instructions without enough arguments to execute). This simplification step is implemented in PyshGP library [134]. After the dividing step, the subprograms are stored into an archive for the future use.

6.4.4 Replacement Mutation

RM requires a parent candidate (of length l_1) from the PushGP population and a subprogram (of length l_2). RM replaces a random partition (of length l_2) of the parent candidate using the subprogram. If $l_1 < l_2$, the entire parent is replaced by the subprogram.

6.4.5 Adaptive Selection

The adaptive selection is designed to automatically select helpful subprograms for the current task when an archive contains both helpful and unhelpful subprograms.

The idea behind is similar to the parameter adaptation in many Self-Adaptive Evolutionary Algorithms (SAEAs) such as JADE [89]. In SAEAs, parameters such as mutation rate are randomly initialized for every individual. The individuals are then mutated based on the parameters associated with them. After mutation, the parameters that are associated with improved individuals will have more chance to survive to the next generation. In other words, the parameters are searched along with the solutions to the problem.

In this study, we exploit helpful subprograms so far with proportional selection to select subprograms based on their quality measure. The proportion to select a subprogram depends on how many times that it improves parents. Meanwhile, we sometimes randomly select subprograms to perform exploration.

We call RM with adaptive selection Adaptive Replacement Mutation (ARM). We provide the pseudocode of PushGP with ARM in **Algorithm 6.1**. $Q(\mathbf{s})$ is the count that a subprogram $\mathbf{s}$ improves the parents during the search. The probability in proportional selection is computed as in (6.3).

$$p(\mathbf{s}) = \frac{Q(\mathbf{s})}{\Sigma_{\mathbf{s}_i \in S} Q(\mathbf{s}_i)} \tag{6.3}$$

r_{arm} is the probability to perform ARM, and r_{prop} is the probability to perform the proportional selection of subprograms. In Line 12 of **Algorithm 6.1**, the symbol "$\prec$" means "better than". In our implementation, a solution is better than another if it solves more I/O cases (i.e., contains more "0" in its error vector).

In some cases, the subprograms may include more inputs than the current problem (e.g., a subprogram contains `input_3` while the current problem only takes two inputs). We replace the input in the subprograms with a random input of the current problem.

Algorithm 6.1: PushGP with Adaptive Replacement Mutation

input : subprogram archive S where every subprogram $\mathbf{s}$ holds a quality metric $Q(\mathbf{s}) = 0$
output: final population X

$X \leftarrow$ initialize();
repeat
  $X' \leftarrow \emptyset$;
  for $i \leftarrow 1 \dots |P|$ **do**
    $\mathbf{x} \leftarrow$ lexicase-select(X);
    if rand() $< r_{arm}$ **then**
      if rand() $< r_{prop}$ **then**
        $\mathbf{s} \leftarrow$ proportional-select(S);
      else
        $\mathbf{s} \leftarrow$ random-select(S);
      $\mathbf{x}' \leftarrow$ replacement-mutate($\mathbf{x}, \mathbf{s}$);
      if $f(\mathbf{x}') \prec f(\mathbf{x})$ **then**
        $Q(\mathbf{s}) \leftarrow Q(\mathbf{s}) + 1$;
    else
      $\mathbf{x}' \leftarrow$ umad-mutate($\mathbf{x}$);
    $X' \leftarrow X' \cup \{\mathbf{x}'\}$;
  $X \leftarrow X'$;
until termination criteria are satisfied;
return X;

6.5 Experiments on Composite Problems

6.5.1 Experimental Methods

In this experiment, we focus on an intermediate step of the KDPS problem (**Figure 6.3**). That is, to solve a composite problem with our proposed method after solving its sub-problems. We compare the following three methods.

PushGP+EP+ARM: PushGP with ARM. The subprogram archive contains the subprograms extracted from the final solutions by EP.

PushGP+HP+ARM: PushGP with ARM. The subprogram archive contains the subprograms extracted from the final solutions by human (HP stands for human partitioning).

PushGP: the original PushGP proposed by Helmuth *et al.* [23].

We use PushGP [23] to solve three problems in GPSB [130]. They are "small or large" (SL), "compare string lengths" (CSL), and "median" (MD). We take the best and shortest solutions among 25 runs (after 5000 steps of simplification) of the three problems to generate subprogram archives. For PushGP+EP+ARM, We use EP to get five equal-length subprograms for every best and shortest solution automatically. The subprograms used by PushGP+HP+ARM are partitions of the same solutions, however, devised by a human.

We then solve the composite problems of SL, CSL, and MD. When solving a composite problem, PushGP+EP+ARM and PushGP+HP+ARM will use subprogram archives generated from solutions of the corresponded subproblems by EP and HP, respectively. We compare the three methods on three composite problems.

Median String Length (MSL): given 3 strings, print the median of their lengths.

Small or Large Median (SLM): given 4 integers a, b, c, d, print "small" if median(a,b,c) $< d$ and "large" if median(a,b,c) $> d$ (and nothing if median(a,b,c) $= d$).

Small or Large String (SLS): given a string n, print "small" if len(n) $<$

100 and "large" if $\text{len}(n) \geq 200$ (and nothing if $100 \leq \text{len}(n) < 200$).

MSL is the composite problem of MD and CSL; SLM is composed of SL and MD; SLS is a composite of SL and CSL.

For all three methods, we use a population size of 1000 and a maximum generation of 300. The UMAD mutation in all three methods is set with addition rate of 0.09 and deletion rate of 0.0826 based on Helmuth's study [23]. For PushGP+EP+ARM and PushGP+HP+ARM, the rate to perform ARM r_{arm} is 0.1 and the rate to perform the proportional selection of subprograms r_{prop} is 0.5 based on the original paper of ARM [133]. For every algorithm, we run 25 repetitions on every problem.

6.5.2 Experimental results

Table 6.1 to **Table 6.3** present the error in the training period of the three methods in 25 runs. The mean value is marked with an underline if the difference between the method and PushGP [23] is significant with a 95% family confidence level through a Wilcoxon rank sum test (i.e., the individual confidence level is computed by Šidák correction.[2]). The proposed PushGP+EP+ARM outperforms the original PushGP with a significant difference in the training error. However, compared with PushGP+HP+ARM, the difference is not statistically significant.

Table 6.4 shows the success counts in the test period of the three comparison methods in 25 runs. We count a run as a successful run only when it passes all I/O cases in both training and testing data. The value is marked with an underline if the difference between the method and PushGP [23] is significant with a 95% family confidence level through a Fisher's exact test (i.e., the individual confidence level is computed by Šidák correction[2]). Compared to PushGP, our PushGP+EP+ARM achieves higher success counts, however, without statistical significance. Compared to the method using human-made subprograms (PushGP+HP+ARM), our proposed method gets a lower success count on MSL, a higher success count on SLM, and an equal success count on SLS. However, these differences are not significant.

We provide a comparison of the best train error by generations from **Figure 6.4** to **Figure 6.6**. PushGP+EP+ARM holds a much faster con-

[2]The individual confidence levels in **Section 6.5** and **Section 6.6** are 99.4% and 99.1%, respectively.

Table 6.1: Train error on MSL in Experiment I

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**0.00**	3.00	**0.12**	0.60
PushGP+HP+ARM	0.00	**0.00**	4.00	0.28	0.98
PushGP	0.00	3.00	59.00	12.00	19.14

Table 6.2: Train error of SLM in Experiment I

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	15.00	41.00	14.24	14.20
PushGP+HP+ARM	0.00	**5.00**	40.00	**12.80**	13.62
PushGP	5.00	35.00	70.00	37.40	17.15

Table 6.3: Train error on SLS in Experiment I

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	5.00	59.00	17.96	22.19
PushGP+HP+ARM	0.00	**0.00**	50.00	**7.76**	13.17
PushGP	0.00	69.00	85.00	59.40	24.01

Table 6.4: Success count in Experiment I

Method	MSL	SLM	SLS
PushGP+EP+ARM	12	**6**	**8**
PushGP+HP+ARM	**15**	5	**8**
PushGP	4	0	1

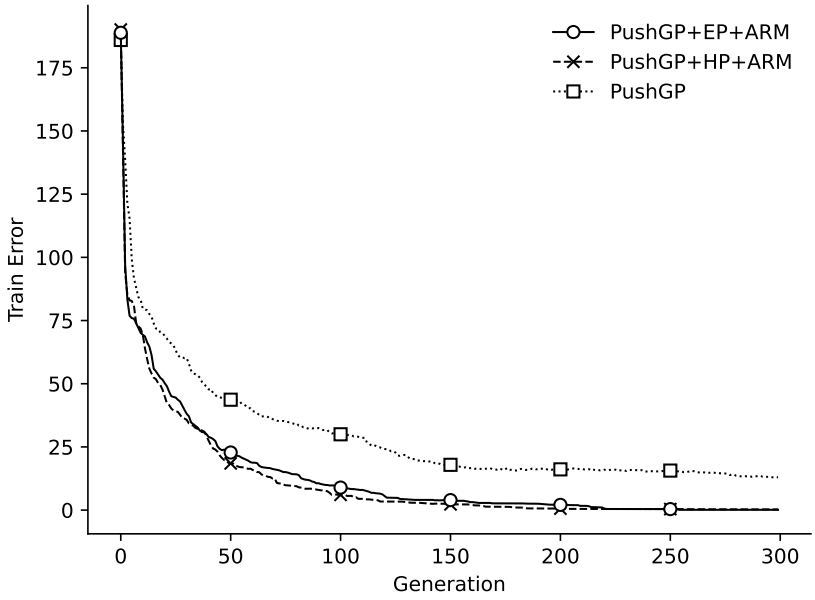

Figure 6.4: Anytime train error on MSL in Experiment I

vergence speed compared to PushGP; however, it is slightly slower than PushGP+HP+ARM.

In the case of solving the sub-problems and then the composite problems, PushGP+EP+ARM achieves a better performance in train error, success count, and faster convergence, compared to the original PushGP [23]. However, its performance in both success count and convergence speed is worse than PushGP+HP+ARM without statistical significance.

6.6 Experiments on Sequential Problems

6.6.1 Experimental Methods

In this second experiment, we test the entire KDPS process in **Figure 6.3**. That is, using PushGP+ARM+EP to solve a sequence of problems. They are the six problems in the last experiment, namely SL, CSL, MD, MSL, SLM, and SLS. Every time a problem is solved, we extract subprograms from its solution and store them in the archive. This archive will be used

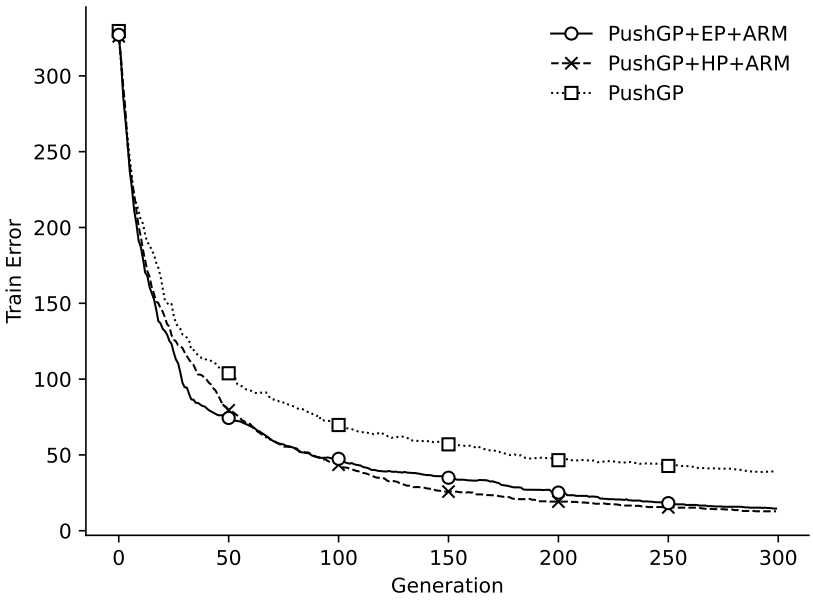

Figure 6.5: Anytime train error on SLM in Experiment I

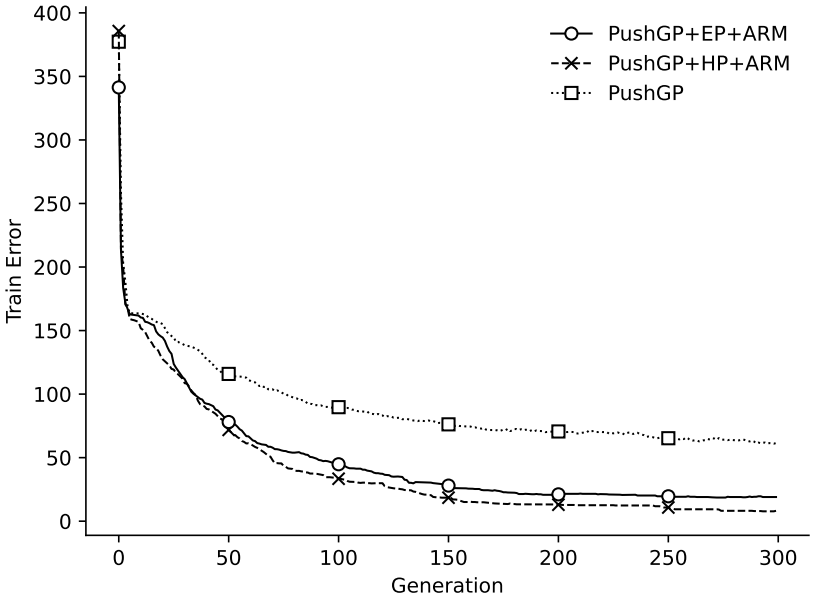

Figure 6.6: Anytime train error on SLS in Experiment I

when solving the next problem.

PushGP+EP+ARM: solving a sequence of PS problems consecutively by PushGP+EP+ARM in the way as in **Figure 6.3**; every time a problem is solved, the subprograms are extracted by EP from its solution and added to the archive. This archive is used by PushGP+ARM when solving the next problem.

PushGP: solving a sequence of PS problems independently using the original PushGP [23].

We solve six problems, namely MD, CSL, SL, MSL, SLM, and SLS. The first three problems are from GPSB [130]. They do not share any sub-problems. The last three problems are the composite problems of MD, CSL, and SL (**Section 6.5**). Any pair of the three composite problems share a sub-problem.

For PushGP+EP+ARM, we run a procedure as in **Figure 6.3**. We solve the first problem (MD) with the original PushGP (i.e., PushGP+ARM with an empty archive) and the rest problems with PushGP+ARM. We initialize an empty subprogram archive when solving the first problem. For every problem, we run 25 repetitions and take the best and shortest program. We use EP to extract five subprograms from the best and shortest programs. These subprograms are stored in the archive that we initialized before and used in solving the later problems by PushGP+ARM. For PushGP, we solve the six problems independently in the same order with PushGP+EP+ARM. However, no subprograms are stored and used.

We provide results of solving the six problems in two different orders. Order 1 solves simple problems at first and later harder ones, while Order 2 is a reverse order of Order 1.

- **Order 1**: MD → CSL → SL → MSL → SLM → SLS
- **Order 2**: SLS → SLM → MSL → SL → CSL → MD

We use the same parameter settings as in **Section 6.5**. We present the results and the statistical test similarly as in **Section 6.5**.

Table 6.5: Train error on MD in Experiment II with Order 1

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**0.00**	8.00	0.56	1.96
PushGP	0.00	**0.00**	5.00	**0.20**	1.00

Table 6.6: Train error of CSL in Experiment II with Order 1

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**0.00**	6.00	1.56	2.02
PushGP	0.00	**0.00**	8.00	**1.48**	2.40

Table 6.7: Train error on SL in Experiment I with Order 1

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**25.00**	80.00	**23.08**	19.14
PushGP	0.00	30.00	75.00	29.84	28.89

Table 6.8: Train error on MSL in Experiment II with Order 1

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**0.00**	14.00	**1.04**	3.10
PushGP	0.00	3.00	59.00	12.00	19.14

Table 6.9: Train error of SLM in Experiment II with Order 1

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**20.00**	45.00	**19.20**	12.39
PushGP	5.00	35.00	70.00	37.40	17.15

Table 6.10: Train error on SLS in Experiment I with Order 1

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**5.00**	85.00	**18.40**	23.75
PushGP	0.00	69.00	85.00	59.40	24.01

Table 6.11: Train error on SLS in Experiment I with Order 2

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**30.00**	92.00	**35.00**	31.67
PushGP	0.00	69.00	85.00	59.40	24.01

Table 6.12: Train error of SLM in Experiment II with Order 2

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**35.00**	75.00	**34.72**	19.43
PushGP	5.00	**35.00**	70.00	37.40	17.15

Table 6.13: Train error on MSL in Experiment II with Order 2

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**0.00**	27.00	**2.88**	6.35
PushGP	0.00	3.00	59.00	12.00	19.14

Table 6.14: Train error on SL in Experiment I with Order 2

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**0.00**	60.00	**10.08**	18.15
PushGP	0.00	30.00	75.00	29.84	28.89

Table 6.15: Train error of CSL in Experiment II with Order 2

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**0.00**	6.00	**0.36**	1.25
PushGP	0.00	0.00	8.00	1.48	2.40

Table 6.16: Train error on MD in Experiment II with Order 2

Method	Best	Median	Worst	Mean	Std.
PushGP+EP+ARM	0.00	**0.00**	0.00	**0.00**	0.00
PushGP	0.00	**0.00**	5.00	0.20	1.00

Table 6.17: Success count in Experiment II with Order 1

Method	MD	CSL	SL	MSL	SLM	SLS
PushGP+EP+ARM	**19**	6	4	**9**	**1**	**5**
PushGP	18	**8**	**7**	4	0	1

6.6.2 Experimental results of Order 1

According to **Table 6.5** to **Table 6.10**, PushGP+EP+ARM holds a significantly lower train error than PushGP on the three composite problems, while the difference on the rest three problems is not significant. **Table 6.17** shows the test success count of PushGP+EP+ARM and PushGP. The test success count of PushGP+EP+ARM is lower than PushGP on CSL and SL but higher than PushGP on MSDLEN, SLM, and SLS. All these difference is not statistically significant through Fisher's exact test. On MD, the difference between the two methods is very small, since it is solved by two equivalent methods. **Figure 6.7** to **Figure 6.12** provide the best error in the population by generations of both methods. It is obvious that PushGP+EP+ARM converges faster than PushGP on all problems except MD.

6.6.3 Experimental results of Order 2

From **Table 6.11** to **Table 6.16**, PushGP+EP+ARM holds a lower train error than PushGP on all problems without statistical significance. **Table 6.18** shows the test success count of PushGP+EP+ARM and PushGP. The test success count of PushGP+EP+ARM is higher than PushGP on most of the problems except MD. Especially on CSL, PushGP+EP+ARM gets 16 success while PushGP only gets 8. However, all these difference is not statistically significant through Fisher's exact test. **Figure 6.13** to **Figure 6.18** provide the best error in the population by generations of both methods. It is obvious that PushGP+EP+ARM converges faster than PushGP on all problems except SLM.

Therefore, when solving a sequence of problems, PushGP+EP+ARM achieves a better optimization performance (i.e., train error and convergence speed). This performance finally leads to a higher test success count, however, without statistical significance.

Table 6.18: Success count in Experiment II with Order 2

Method	SLS	SLM	MSL	SL	CSL	MD
PushGP+EP+ARM	**2**	**3**	**6**	**12**	**16**	17
PushGP	1	0	4	7	8	**18**

6.7 Discussion

We find that the PushGP+EP+ARM in **Section 6.5** is better than the PushGP+EP+ARM in **Section 6.6** with Order 1, in terms of test success on the three composite problems. Though their algorithms are the same, they have at least two differences.

1. In **Section 6.6**, PushGP+EP+ARM adds five subprograms to the archive after solving one problem. Therefore, the size of the archive is 15, 20, and 25 when solving MSL, SLM, and SLS, respectively. However, in **Section 6.5**, PushGP+EP+ARM uses archives with 10 subprograms (five for one sub-problem). A larger subprogram archive makes it harder to select helpful subprograms by the adaptive strategy in **Algorithm 6.1**.

2. CSL and SL are solved in different conditions in the two experiments. In **Section 6.6**, CSL is solved with subprograms from MD; SL is solved with subprograms from MD and CSL. However, no subprogram is used when solving MD, CSL, and SL in **Section 6.5** (i.e., they are solved by the original PushGP [23]). Moreover, MD, CSL, and SL do not share the same sub-problems. Therefore, solving CSL and SL with PushGP+EP+ARM is not as good as with the original PushGP (as shown in **Table 6.17**). Thus, the subprograms extracted in **Section 6.6** is not as good as in **Section 6.5**. These subprograms further influence the performance of PushGP+EP+ARM in **Section 6.6** in solving the later problems MSL, SLM, and SLS. This issue is called "negative transfer".

In **Section 6.6** (problems in Order 2), we find that PushGP+EP+ARM holds a lower success count on MD. However, on MD, the training error of all runs with PushGP+EP+ARM is 0 (**Table 6.16**). Moreover, **Figure 6.18** shows PushGP+EP+ARM converges much faster than PushGP. This observation may indicate an over-fitting issue with the proposed method.

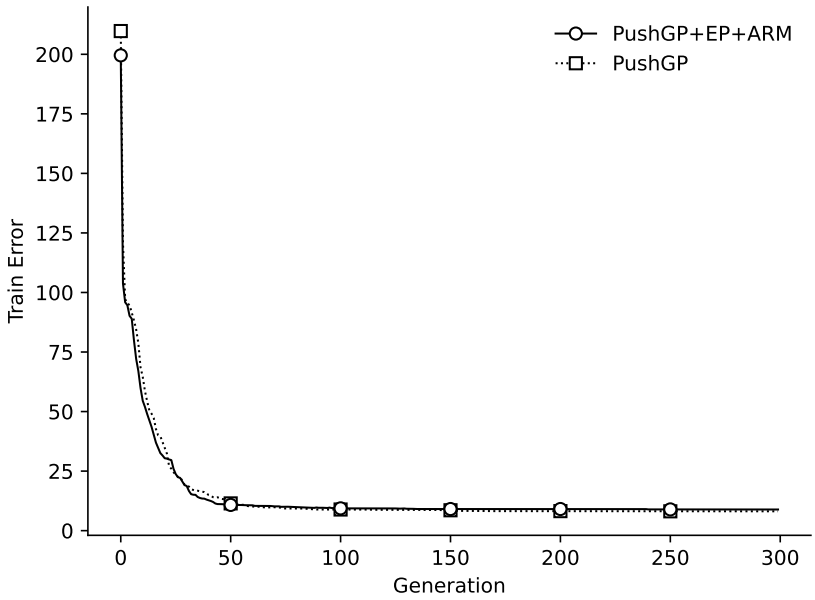

Figure 6.7: Anytime train error on MD in Experiment II with Order 1

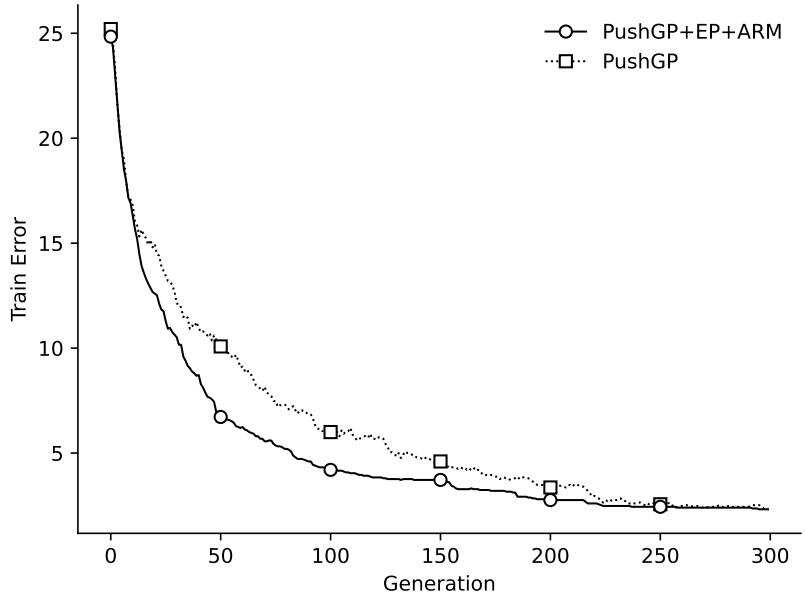

Figure 6.8: Anytime train error on CSL in Experiment II with Order 1

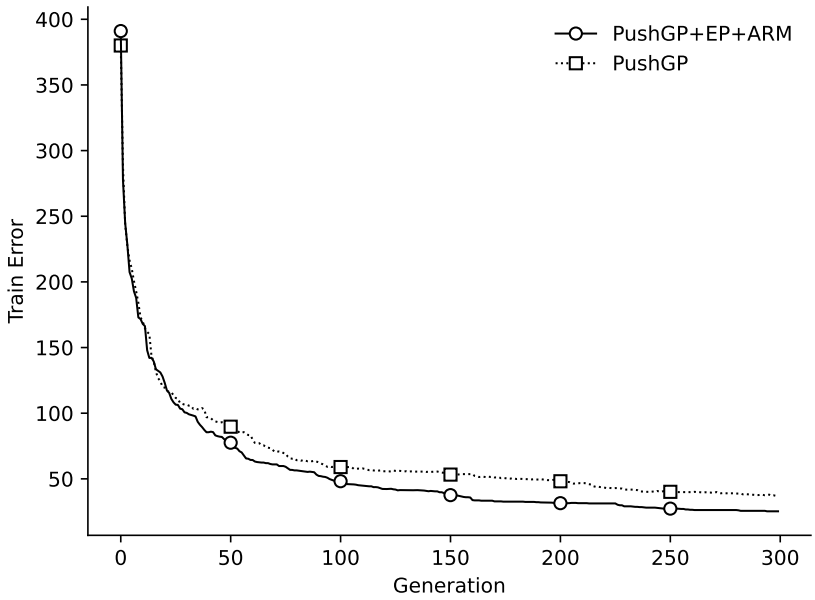

Figure 6.9: Anytime train error on SL in Experiment II with Order 1

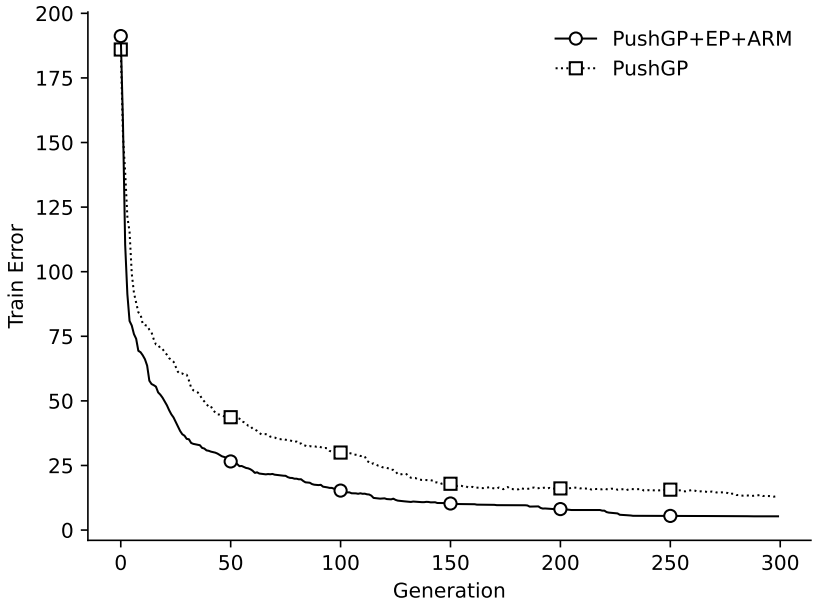

Figure 6.10: Anytime train error on MSL in Experiment II with Order 1

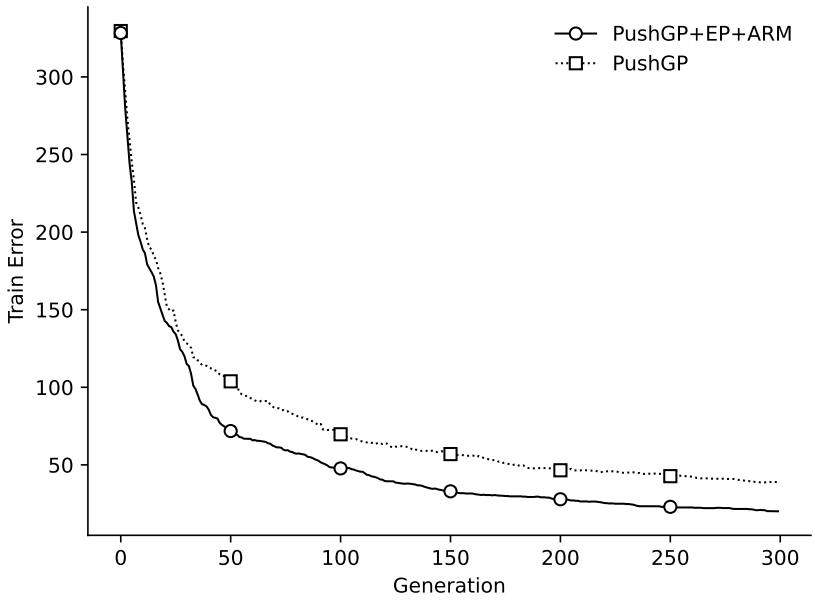

Figure 6.11: Anytime train error on SLM in Experiment II with Order 1

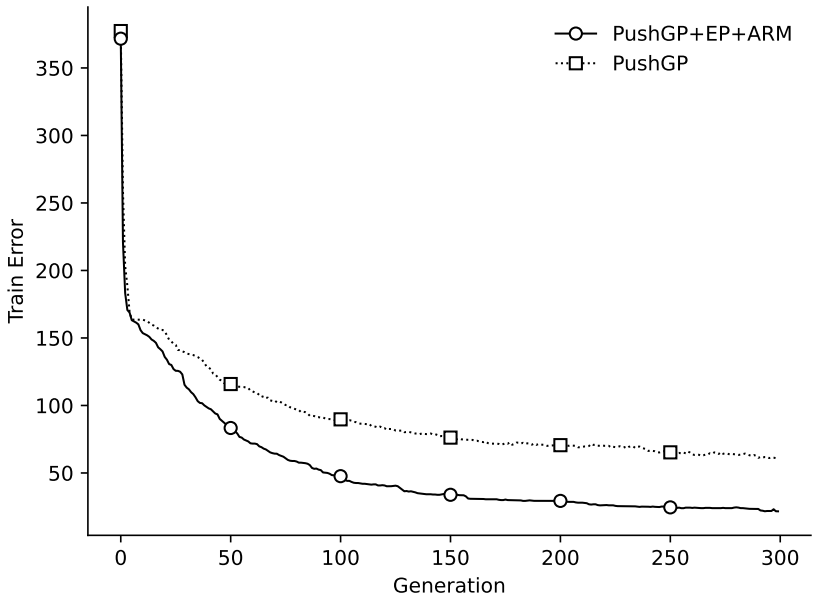

Figure 6.12: Anytime train error on SLS in Experiment II with Order 1

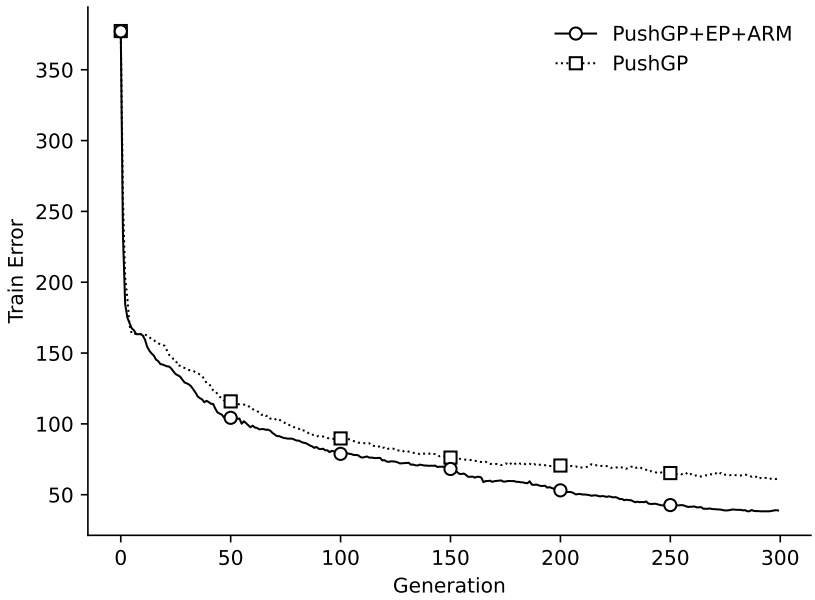

Figure 6.13: Anytime train error on SLS in Experiment II with Order 2

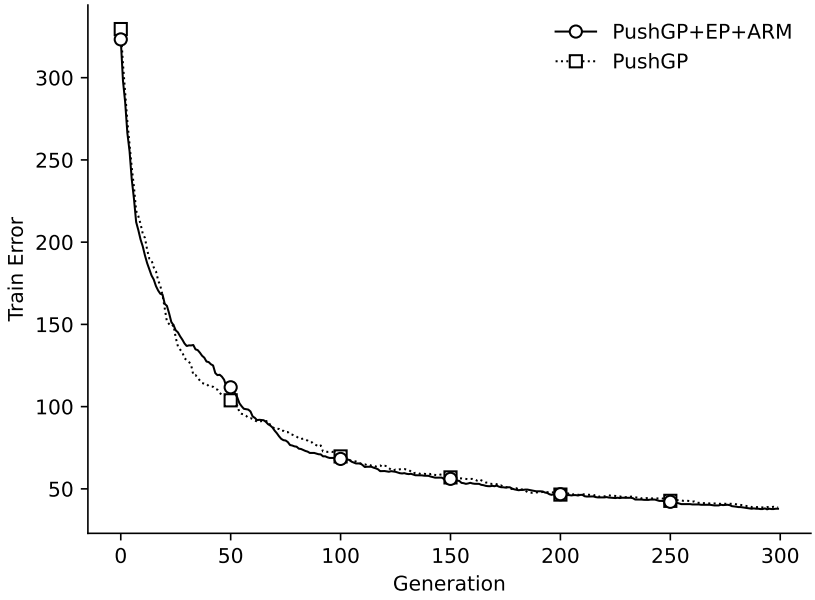

Figure 6.14: Anytime train error on SLM in Experiment II with Order 2

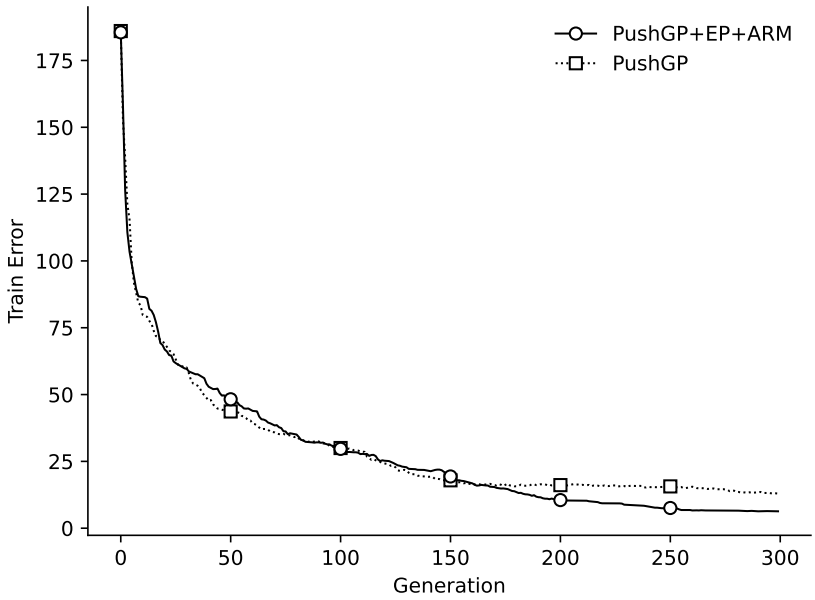

Figure 6.15: Anytime train error on MSL in Experiment II with Order 2

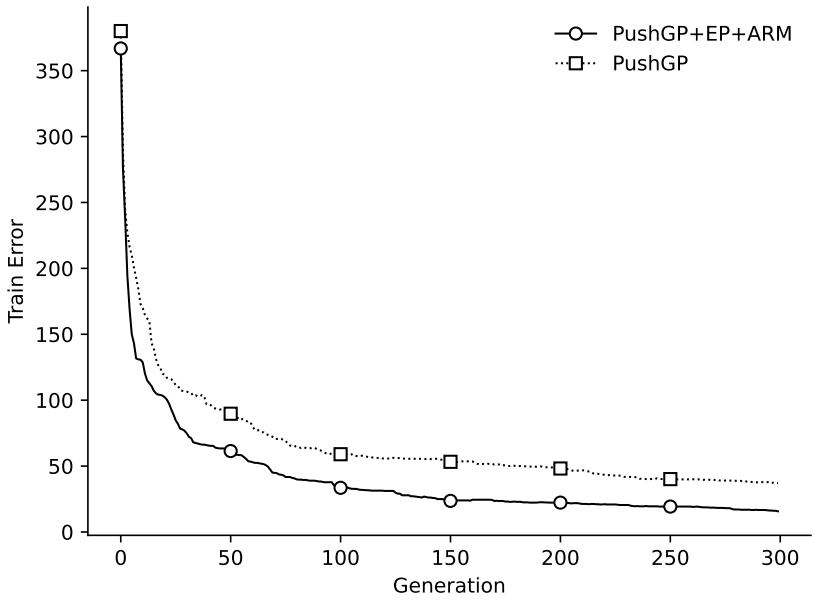

Figure 6.16: Anytime train error on SL in Experiment II with Order 2

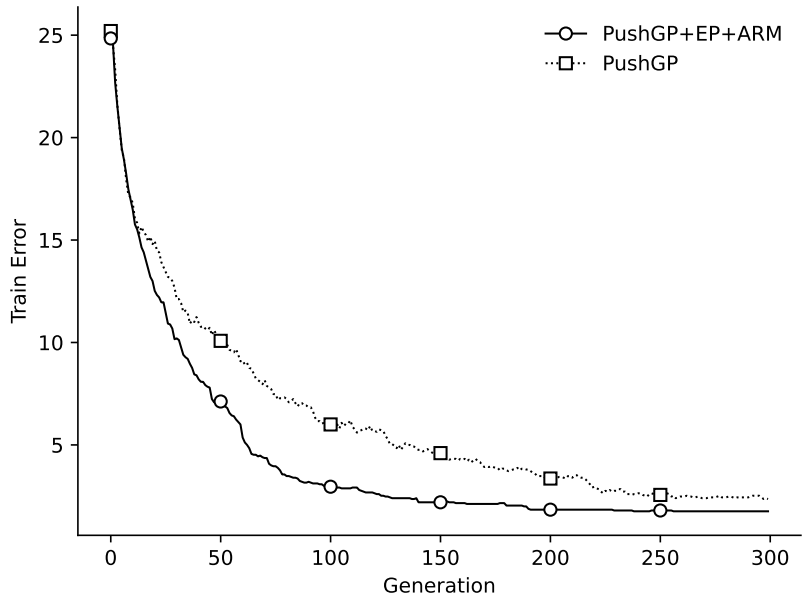

Figure 6.17: Anytime train error on CSL in Experiment II with Order 2

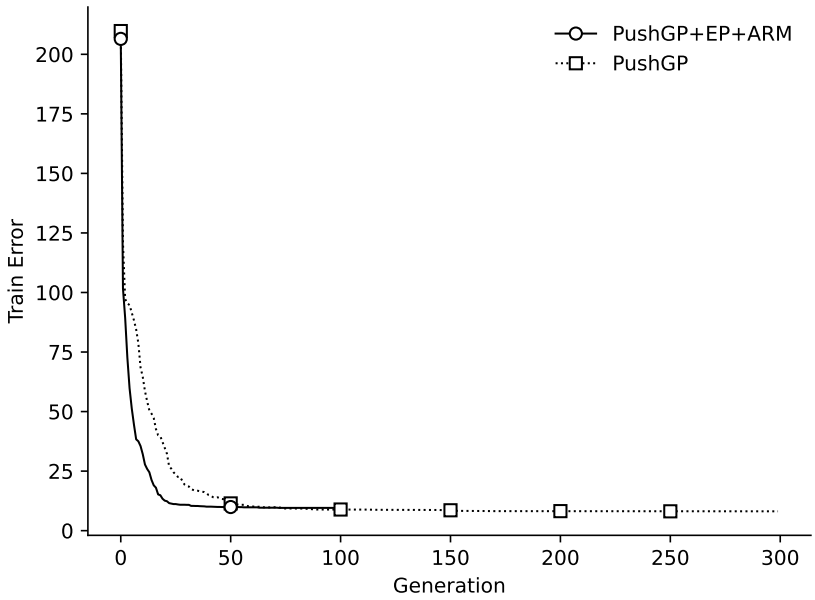

Figure 6.18: Anytime train error on MD in Experiment II with Order 2

6.8 Conclusions of the Case Study

In this study, we have introduced a problem called Knowledge-Driven Program Synthesis (KDPS) problem. KDPS requires an agent to solve a sequence of related PS problems. To solve KDPS, we have proposed a method based on PushGP [23]. This method consecutively solves programming tasks, extracts subprograms from the solutions as knowledge, and uses these subprograms to solve the next problem. To extract subprograms from the solution of a solved problem, we have proposed the Even Partitioning (EP) method; to use these subprograms, we applied Adaptive Replacement Mutation (ARM) [133].

We have compared our proposed method (PushGP+EP+ARM) with the original PushGP [23] and a method extracting subprograms by humans (PushGP+HP+ARM). Our PushGP+EP+ARM has achieved a significantly better train error, success count, and convergence speed than PushGP. The performance of our proposed method is slightly worse than PushGP+HP+ ARM. We have further compared our PushGP+EP+ARM with the original PushGP in solving a sequence of problems. Our method has achieved a better train error and convergence speed. Our PushGP+EP+ARM also holds a higher test success count, however, without statistical significance.

The current method to automatically construct the subprogram archive is rather naive. We would like to improve this method in our future work. In the discussion section in **Section 6.6**, PushGP+EP+ARM has limitations in dealing with the growing subprogram archive after solving more problems. Moreover, PushGP+EP+ARM also suffers from the "negative transfer" of the subprograms from the unrelated problems. A part of our future work is to fix these issues. Methods such as a more efficient adaptation or filtering strategy on subprograms are promising to solve the two limitations. Additionally, the strategy of extracting and using knowledge could be applied to problems other than program synthesis, such as training soft robotics [138].

Chapter 7

Conclusions

7.1 Automatic Discovery of Sub-tasks

The idea of our Adaptive Genetic Knowledge Transfer (AGKT) system derives from two separate projects that are not mentioned in this dissertation. Both projects are trying to bring some characteristics of human intelligence to Evolutionary Algorithms (EAs).

In the first project, we have attempted to build a set of subtrees to help the search of Genetic Programming (GP). Based on our own programming experiences, there are many code snippets that are frequently used in different tasks. This first project aims to bring a similar thing to GP. However, we finally have failed to find this promising set of subtrees.

In our second project, we have come up with a research question on whether a GP can automatically discover sub-problems of the given problem. Decomposing a task into several sub-tasks is vital for solving complex problems. At the very beginning, we have tried a method that evolves a population of I/O datasets (that represents sub-problems) and uses the solutions to these datasets to solve the original task. These datasets are randomly initialized but evolved based on the quality of their solutions. However, this method cannot perform efficiently since we have to run an entire GP to get the solution to an evolved dataset.

Fortunately, we have realized that we can combine the two research projects. If a GP has solved the sub-problems of the current task, we can simply store the solutions to these sub-problems in an archive (like the first project). When solving the current task, we select and use these solutions in the evolutionary process of GP (like the second project). We have no-

ticed the similarity between this design and a human who solves tasks and improves himself/herself. Based on this idea, we have come up with our Knowledge-Driven Program Synthesis (KDPS) problem and system.

However, we have not proposed the idea of Genetic Knowledge Transfer (GKT) till now. We have started to think about GKT when we were trying to summarize all the research projects during these three years. Before the KDPS system, we have studied a method using Lexicase Selection (LS) [19] to solve the Seismic History Matching (SHM) problems. We have surprisingly found the connection between the two works: in both methods, an EA is affected by the dynamics of another EA. We have further investigated similar techniques in the existing literature, including MOEA/D [36] and Multi-Task Optimization [2].

After that, we have summarized the common parts of these methods and proposed GKT. We have also come up with a Naive GKT model that migrates selected individuals from one EA to another. Many GKT techniques could be considered as modifications of this naive model. Finally, we have realized the adaptive selection of the genetic knowledge in our KDPS is one of the most important differences between those naive methods in the literature. This adaptive selection allows us to automatically find suitable sub-problems from past problems.

7.2 Summary of the Research

In this dissertation, we start with Genetic Knowledge Transfer (GKT) in Evolutionary Computation with multiple tasks. We define GKT as the process where an Evolutionary Algorithm is affected by the dynamics of another EA in **Chapter 3**. We further show several examples of GKT in the existing optimization paradigms, including Multi-Objective Optimization (MOO), Multi-Task Optimization (MTO), and Genetic Programming (GP).

Based on the above examples, we summarize a basic model of GKT called Naive GKT (NGKT) in **Section 4.1**. NGKT happens when an EA sends selected individuals to another EA, and these individuals are used as parents to reproduce child individuals in the second EA.

We then discuss the possible limitations of the NGKT method, regarding a future scenario where an EA is required to solve an endless sequence

of distinct tasks. That is, the naive method can transfer genetic knowledge from an unrelated task which might not be helpful to the current task. To address this issue, this work proposes a method that allows the adaptive transfer of genetic knowledge called the Adaptive Genetic Knowledge Transfer (AGKT) system in **Section 4.2**. The AGKT system consecutively solves tasks, extracts sub-solutions as genetic knowledge, and reuses these sub-solutions properly when the user poses a new task. To select a proper sub-solution, the system uses the similarity between tasks and trial-and-error methods. We provide general design for every single component.

In **Chapter 5**, we present a case study on using GKT to solve the Seismic History Matching (SHM) problem. SHM requires finding a subsurface model that matches a set of simulated data with real-world records. We proposed a Differential Evolution algorithm based on Lexicase Selection (LS) [19] to solve the SHM problems. We explain how GKT happens in the LS in an implicit manner. The results on two SHM benchmarks indicated the superiority of our proposed method in terms of closer distance to the ground truth and a more concentrated solution set with fewer unphysical solutions.

We provide a case study to demonstrate our AGKT system in **Chapter 6**. We first set up a new type of problem called the Knowledge-Driven Program Synthesis (KDPS) problem. A KDPS problem requires a GP algorithm to solve a sequence of distinct Program Synthesis (PS) tasks. Moreover, the GP should use the genetic knowledge extracted from previous tasks to solve the current PS task. We propose a method to solve this KDPS problem based on PushGP [23]. This method extracts subprograms as genetic knowledge by Even Partitioning (EP) and stores them in an archive. Adaptive Replacement Mutation (ARM) is designed to reuse these subprograms based on the feedback from the evolutionary process of PushGP. We test our proposed method on three composite problems as well as two sequences of six PS problems (i.e., two KDPS problems). Our proposed method achieves a better success rate.

We show some possible future directions in the next section.

7.3 Future Directions

First, we would like to focus our main attention on developing new methods to solve the KDPS problem. In the current method to solve the KDPS problem, subprograms are extracted by a simple EP method. A method

that considers the logical connection between instructions is promising. For example, a program could be represented as a graph. Therefore, techniques of graph theory, such as community detection, could be used to address this issue.

We have not implemented the filter for the KDPS system. A possible design is to measure the similarity between tasks by computing the similarity between text descriptions. Related to this topic, Large Language Models (LLMs) are another method to solve PS problems, however, using a different specification of natural language. A part of the future works is to discuss the possibility of combining KDPS with LLMs or other Neural Networks (NNs). One possible way is to build a recommendation system that recommends subprograms from the archive for reusing in the current PS problem.

The current method of subprogram selection could be further improved, considering a different scenario where a large set of subprograms are included. Possible improvements could be a new type of feedback to use or a new strategy to keep the balance of the exploitation and exploration of the subprograms.

We are also interested in studying how different problems influence the results of our KDPS system. One way is to perform a fitness landscape analysis to show some insights on GKT from a view of optimization. Moreover, it is also interesting to apply the KDPS system to solve some real-world problems such as training soft robotics and extracting image features.

In spite of new methods to solve KDPS problems, another important topic is communication and collaboration between two KDPS systems. If we have two KDPS systems that have solved two sequences of problems in different categories, it is possible to build a method that allows collaboration between two systems to solve a complex problem. In this case, the genetic knowledge is transferred from one KDPS system to another.

Acknowledgements

First, I would like to thank Prof. Tetsuya Sakurai and Prof. Claus Aranha from the University of Tsukuba for being my supervisors and providing financial support. I would also like to thank other members of my thesis committee, Prof. Hitoshi Iba from the University of Tokyo, Prof. Yukiyoshi Kameyama, and Prof. Koji Hasebe from the University of Tsukuba, for their insightful questions and comments.

The Seismic History Matching case study was done in collaboration with Prof. Romain Chassagne and Dr. Antony Hallam from Herriot-Watt University, Scotland. I would like to thank them for providing benchmark problems and computational devices for this research project. I am also grateful for their suggestions on this case study.

I would like to thank the members of the Evolutionary Computation group in the Mathematical Modeling and Algorithm Lab at the University of Tsukuba. This work cannot be finished without their kind help. I would like to express my thanks to Mr. Yuri Lavinas, Mr. Jair Pereira, and Mr. Fabio Tanaka for their technical comments on the Knowledge-Driven Program Synthesis case study.

I would like to thank my parents for their financial support in my life in Japan, as well as the rental server for experiments. Finally, I would like to thank those who have provided mental support for me during these three years.

Bibliography

[1] Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449. Springer, 2014.

[2] Abhishek Gupta, Yew-Soon Ong, and Liang Feng. Multifactorial evolution: toward evolutionary multitasking. *IEEE Transactions on Evolutionary Computation*, 20(3):343–357, 2015.

[3] Yongliang Chen, Jinghui Zhong, Liang Feng, and Jun Zhang. An adaptive archive-based evolutionary framework for many-task optimization. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 4(3):369–384, 2019.

[4] Tingyang Wei, Shibin Wang, Jinghui Zhong, Dong Liu, and Jun Zhang. A review on evolutionary multi-task optimization: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 2021.

[5] Wojciech Jaskowski, Krzysztof Krawiec, and Bartosz Wieloch. Multi-task code reuse in genetic programming. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*, pages 2159–2164, 2008.

[6] Eric O Scott and Kenneth A De Jong. Automating knowledge transfer with multi-task optimization. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 2252–2259. IEEE, 2019.

[7] Ahmed Kattan, Faiyaz Doctor, Yew-Soon Ong, and Alexandros Agapitos. Genetic programming multitasking. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1004–1012. IEEE, 2020.

[8] Mazhar Ansari Ardeh, Yi Mei, and Mengjie Zhang. A novel multi-task genetic programming approach to uncertain capacitated arc routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 759–767, 2021.

[9] Fangfang Zhang, Yi Mei, Su Nguyen, Kay Chen Tan, and Mengjie Zhang. Multitask genetic programming-based generative hyperheuristics: A case study in dynamic scheduling. *IEEE Transactions on Cybernetics*, 2021.

[10] Gregory Seront. External concepts reuse in genetic programming. In *working notes for the AAAI Symposium on Genetic programming*, pages 94–98. MIT/AAAI Cambridge, 1995.

[11] Maarten Keijzer, Conor Ryan, Gearoid Murphy, and Mike Cattolico. Undirected training of run transferable libraries. In *European Conference on Genetic Programming*, pages 361–370. Springer, 2005.

[12] Muhammad Iqbal, Bing Xue, and Mengjie Zhang. Reusing extracted knowledge in genetic programming to solve complex texture image classification problems. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 117–129. Springer, 2016.

[13] Muhammad Iqbal, Bing Xue, Harith Al-Sahaf, and Mengjie Zhang. Cross-domain reuse of extracted knowledge in genetic programming for image classification. *IEEE Transactions on Evolutionary Computation*, 21(4):569–587, 2017.

[14] Damien O'Neill, Harith Al-Sahaf, Bing Xue, and Mengjie Zhang. Common subtrees in related problems: A novel transfer learning approach for genetic programming. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1287–1294. IEEE, 2017.

[15] Jordan Wick, Erik Hemberg, and Una-May O'Reilly. Getting a head start on program synthesis with genetic programming. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 263–279. Springer, 2021.

[16] Thomas Helmuth, Edward Pantridge, Grace Woolson, and Lee Spector. Genetic source sensitivity and transfer learning in genetic programming. In *ALIFE 2020: The 2020 Conference on Artificial Life*, pages 303–311. MIT Press, 2020.

[17] Romain Chassagne and Claus Aranha. A pragmatic investigation of the objective function for subsurface data assimilation problem. *Operations Research Perspectives*, 7:100143, 2020.

[18] Yifan He, Claus Aranha, Antony Hallam, and Romain Chassagne. Optimization of subsurface models with multiple criteria using lexicase selection. *Operations Research Perspectives*, 9:100237, 2022.

[19] Thomas Helmuth, Lee Spector, and James Matheson. Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643, 2014.

[20] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[21] Ran Cheng, Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. A reference vector guided evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 20(5):773–791, 2016.

[22] Yifan He, Claus Aranha, and Tetsuya Sakurai. Knowledge-driven program synthesis via adaptive replacement mutation and auto-constructed subprogram archives. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 14–21. IEEE, 2022.

[23] Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. Program synthesis using uniform mutation by addition and deletion. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1127–1134, 2018.

[24] Abhishek Gupta and Yew-Soon Ong. Back to the roots: Multi-x evolutionary computation. *Cognitive Computation*, 11(1):1–17, 2019.

[25] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[26] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[27] Kenneth V Price. Differential evolution. In *Handbook of optimization*, pages 187–214. Springer, 2013.

[28] Takehisa Kohira, Hiromasa Kemmotsu, Oyama Akira, and Tomoaki Tatsukawa. Proposal of benchmark problem based on real-world car structure design optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 183–184, 2018.

[29] Harry M Markowits. Portfolio selection. *Journal of finance*, 7(1):71–91, 1952.

[30] Antonio C Briza and Prospero C Naval Jr. Stock trading system based on the multi-objective particle swarm optimization of technical indicators on end-of-day market data. *Applied Soft Computing*, 11(1):1191–1201, 2011.

[31] Ron Janssen. *Multiobjective decision support for environmental management*, volume 2. Springer Science & Business Media, 2012.

[32] Thibaut Lust and Jacques Teghem. The multiobjective traveling salesman problem: a survey and a new approach. In *Advances in Multi-Objective Nature Inspired Computing*, pages 119–141. Springer, 2010.

[33] Kalyanmoy Deb. Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*, pages 3–34. Springer, 2011.

[34] Carlos A Coello Coello and Margarita Reyes Sierra. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In *Mexican international conference on artificial intelligence*, pages 688–697. Springer, 2004.

[35] Jesús Guillermo Falcón-Cardona and Carlos A Coello Coello. Indicator-based multi-objective evolutionary algorithms: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 53(2):1–35, 2020.

[36] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

[37] Indraneel Das and John E Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM journal on optimization*, 8(3):631–657, 1998.

[38] Kalyanmoy Deb, Ram Bhushan Agrawal, et al. Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148, 1995.

[39] Kalyanmoy Deb, Karthik Sindhya, and Tatsuya Okabe. Self-adaptive simulated binary crossover for real-parameter optimization. In *Proceedings of the 9th annual conference on genetic and evolutionary computation*, pages 1187–1194, 2007.

[40] Jinghui Zhong, Linhao Li, Wei-Li Liu, Liang Feng, and Xiao-Min Hu. A co-evolutionary cartesian genetic programming with adaptive knowledge transfer. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 2665–2672. IEEE, 2019.

[41] Abhishek Gupta, Jacek Mańdziuk, and Yew-Soon Ong. Evolutionary multitasking in bi-level optimization. *Complex & Intelligent Systems*, 1(1):83–95, 2015.

[42] Aritz D Martinez, Javier Del Ser, Eneko Osaba, and Francisco Herrera. Adaptive multifactorial evolutionary optimization for multitask reinforcement learning. *IEEE Transactions on Evolutionary Computation*, 26(2):233–247, 2021.

[43] Ying Bi, Bing Xue, and Mengjie Zhang. Learning and sharing: A multitask genetic programming approach to image feature learning. *IEEE Transactions on Evolutionary Computation*, 26(2):218–232, 2021.

[44] Gen Yokoya, Heng Xiao, and Toshiharu Hatanaka. Multifactorial optimization using artificial bee colony and its application to car structure design optimization. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 3404–3409. IEEE, 2019.

[45] Yongliang Chen, Jinghui Zhong, and Mingkui Tan. A fast memetic multi-objective differential evolution for multi-tasking optimization. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

[46] Genghui Li, Qingfu Zhang, and Weifeng Gao. Multipopulation evolution framework for multifactorial optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 215–216, 2018.

[47] Maoguo Gong, Zedong Tang, Hao Li, and Jun Zhang. Evolutionary multitasking with dynamic resource allocating strategy. *IEEE Transactions on Evolutionary Computation*, 23(5):858–869, 2019.

[48] Shijia Huang, Jinghui Zhong, and Wei-Jie Yu. Surrogate-assisted evolutionary framework with adaptive knowledge transfer for multi-task optimization. *IEEE transactions on emerging topics in computing*, 9(4):1930–1944, 2019.

[49] Tingyang Wei and Jinghui Zhong. A preliminary study of knowledge transfer in multi-classification using gene expression programming. *Frontiers in Neuroscience*, 13:1396, 2020.

[50] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[51] Michael A Lones. Optimising optimisers with push gp. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 101–117. Springer, 2020.

[52] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[53] Douglas Adriano Augusto and Helio JC Barbosa. Symbolic regression via genetic programming. In *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks*, pages 173–178. IEEE, 2000.

[54] Steven Gustafson, Edmund K Burke, and Natalio Krasnogor. On improving genetic programming for symbolic regression. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 912–919. IEEE, 2005.

[55] John R Koza, Forrest H Bennett, David Andre, and Martin A Keane. Automated design of both the topology and sizing of analog electrical circuits using genetic programming. In *Artificial intelligence in design'96*, pages 151–170. Springer, 1996.

[56] Julian F Miller, Peter Thomson, and Terence Fogarty. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. *Genetic algorithms and evolution strategies in engineering and computer science*, pages 105–131, 1997.

[57] Shotaro Kamio and Hitoshi Iba. Adaptation technique for integrating genetic programming and reinforcement learning for real robots. *IEEE Transactions on Evolutionary Computation*, 9(3):318–333, 2005.

[58] John Rieffel, Davis Knox, Schuyler Smith, and Barry Trimmer. Growing and evolving soft robots. *Artificial life*, 20(1):143–162, 2014.

[59] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*, pages 497–504, 2017.

[60] Adam Gaier and David Ha. Weight agnostic neural networks. *Advances in neural information processing systems*, 32, 2019.

[61] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[62] Jean-Yves Potvin, Patrick Soriano, and Maxime Vallée. Generating trading rules on the stock markets with genetic programming. *Computers & Operations Research*, 31(7):1033–1047, 2004.

[63] Shu-Heng Chen, Tzu-Wen Kuo, and Kong-Mui Hoi. Genetic programming and financial trading: how much about" what we know". In *Handbook of financial engineering*, pages 99–154. Springer, 2008.

[64] C Aranha, O Kasai, U Uchide, and H Iba. Day-trading rules development by genetic programming. In *Information Sciences 2007*, pages 515–521. World Scientific, 2007.

[65] Ying Bi, Bing Xue, and Mengjie Zhang. Genetic programming with image-related operators and a flexible program structure for feature learning in image classification. *IEEE Transactions on Evolutionary Computation*, 25(1):87–101, 2020.

[66] Ying Bi, Bing Xue, and Mengjie Zhang. *Genetic programming for image classification: An automated approach to feature learning*, volume 24. Springer Nature, 2021.

[67] Lee Spector. Autoconstructive evolution: Push, pushgp, and pushpop. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, volume 137, 2001.

[68] John R Woodward. Modularity in genetic programming. In *European Conference on Genetic Programming*, pages 254–263. Springer, 2003.

[69] Peter J Angeline and Jordan Pollack. Evolutionary module acquisition. In *Proceedings of the second annual conference on evolutionary programming*, pages 154–163, 1993.

[70] Simon C Roberts, Daniel Howard, and John R Koza. Evolving modules in genetic programming by subtree encapsulation. In *European Conference on Genetic Programming*, pages 160–175. Springer, 2001.

[71] Lee Spector. Evolving control structures with automatically defined macros. In *Working Notes of the AAAI Fall Symposium on Genetic Programming*, pages 99–105. American Association for Artificial Intelligence Menlo Park, Calif, 1995.

[72] Wolfgang Banzhaf, Dirk Banscherus, and Peter Dittrich. Hierarchical genetic programming using local modules. In *Unifying Themes in Complex Systems*, pages 321–330. CRC Press, 2018.

[73] Lee Spector, Brian Martin, Kyle Harrington, and Thomas Helmuth. Tag-based modules in genetic programming. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1419–1426, 2011.

[74] Lee Spector, Kyle Harrington, and Thomas Helmuth. Tag-based modularity in tree-based genetic programming. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 815–822, 2012.

[75] Aniko Ekart and Sandor Z. Nemeth. Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 2(1):61–73, 2001.

[76] Stefan Bleuler, Martin Brack, Lothar Thiele, and Eckart Zitzler. Multiobjective genetic programming: Reducing bloat using spea2. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, volume 1, pages 536–543. IEEE, 2001.

[77] Edwin D De Jong, Richard A Watson, and Jordan B Pollack. Reducing bloat and promoting diversity using multi-objective methods. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 11–18, 2001.

[78] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008.

[79] Linda Argote and Paul Ingram. Knowledge transfer: A basis for competitive advantage in firms. *Organizational behavior and human decision processes*, 82(1):150–169, 2000.

[80] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[81] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[82] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[83] Abhishek Gupta, Yew-Soon Ong, and Liang Feng. Insights on transfer optimization: Because experience is the best teacher. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):51–64, 2017.

[84] Clément Legrand, Diego Cattaruzza, Laetitia Jourdan, and Marie-Eléonore Kessaci. Enhancing moea/d with learning: application to routing problems with time windows. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 495–498, 2022.

[85] Martin Pelikan and Mark W Hauschild. Learn from the past: Improving model-directed optimization by transfer learning based on distance-based bias. *Missouri Estimation of Distribution Algorithms Laboratory, University of Missouri in St. Louis, MO, United States, Tech. Rep*, 2012007, 2012.

[86] Lingyu Huang, Liang Feng, Handing Wang, Yaqing Hou, Kai Liu, and Chao Chen. A preliminary study of improving evolutionary multi-objective optimization via knowledge transfer from single-objective problems. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1552–1559. IEEE, 2020.

[87] Edgar Galván and Fergal Stapleton. Promoting semantics in multi-objective genetic programming based on decomposition. *arXiv preprint arXiv:2012.04717*, 2020.

[88] Fergal Stapleton and Edgar Galván. Semantic neighborhood ordering in multi-objective genetic programming based on decomposition. In

2021 IEEE Congress on Evolutionary Computation (CEC), pages 580–587. IEEE, 2021.

[89] Jingqiao Zhang and Arthur C Sanderson. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5):945–958, 2009.

[90] P Mitchell and R Chassagne. 4d assisted seismic history matching using a differential evolution algorithm at the harding south field. In *81st EAGE Conference and Exhibition 2019*, volume 2019, pages 1–5. European Association of Geoscientists & Engineers, 2019.

[91] Dennis Obidegwu, Romain Chassagne, and Colin MacBeth. Seismic assisted history matching using binary maps. *Journal of Natural Gas Science and Engineering*, 42:69–84, 2017.

[92] Dean S Oliver, Kristian Fossum, Tuhin Bhakta, Ivar Sandø, Geir Nævdal, and Rolf Johan Lorentzen. 4d seismic history matching. *Journal of Petroleum Science and Engineering*, 207:109119, 2021.

[93] Qi Zhang, Romain Chassagne, and Colin MacBeth. Seismic history matching uncertainty with weighted objective functions. In *ECMOR XVI-16th European conference on the mathematics of oil recovery*, volume 2018, pages 1–12. European Association of Geoscientists & Engineers, 2018.

[94] Karl D Stephen, Juan Soldo, Colin MacBeth, and Mike Christie. Multiple-model seismic and production history matching: a case study. *SPE Journal*, 11(04):418–430, 2006.

[95] Claus Aranha, Ryoji Tanabe, Romain Chassagne, and Alex Fukunaga. Optimization of oil reservoir models using tuned evolutionary algorithms and adaptive differential evolution. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 877–884. IEEE, 2015.

[96] Qi Zhang, Romain Chassagne, and Colin MacBeth. 4d seismic and production history matching, a combined formulation using hausdorff and fréchet metric. In *SPE Europec featured at 81st EAGE Conference and Exhibition*. OnePetro, 2019.

[97] Ralf Schulze-Riegert, Markus Krosche, Abul Fahimuddin, and Shawket Ghedan. Multiobjective optimization with application to model validation and uncertainty quantification. In *SPE Middle East oil and gas show and conference*. OnePetro, 2007.

[98] F Verga, M Cancelliere, and D Viberti. Improved application of assisted history matching techniques. *Journal of Petroleum Science and Engineering*, 109:327–347, 2013.

[99] Linah Mohamed, Mike Christie, and Vasily Demyanov. History matching and uncertainty quantification: multiobjective particle swarm optimisation approach. In *SPE EUROPEC/EAGE annual conference and exhibition*. OnePetro, 2011.

[100] Mike Christie, Dmitry Eydinov, Vasily Demyanov, Jack Talbot, Dan Arnold, and Vassili Shelkov. Use of multi-objective algorithms in history matching of a real field. In *SPE reservoir simulation symposium*. OnePetro, 2013.

[101] JJ Hutahaean, V Demyanow, and Michael Andrew Christie. Impact of model parameterisation and objective choices on assisted history matching and reservoir forecasting. In *SPE/IATMI Asia Pacific oil & gas conference and exhibition*. OnePetro, 2015.

[102] Junko Hutahaean, Vasily Demyanov, and Michael A Christie. On optimal selection of objective grouping for multiobjective history matching. *SPE Journal*, 22(04):1296–1312, 2017.

[103] Mohammad Sayyafzadeh and Manouchehr Haghighi. Regularization in history matching using multi-objective genetic algorithm and bayesian framework (spe 154544). In *74th EAGE Conference and Exhibition incorporating EUROPEC 2012*, pages cp–293. European Association of Geoscientists & Engineers, 2012.

[104] Mohammed S Kanfar and Christopher R Clarkson. Reconciling flowback and production data: A novel history matching approach for liquid rich shale wells. *Journal of Natural Gas Science and Engineering*, 33:1134–1148, 2016.

[105] Han-Young Park, Akhil Datta-Gupta, and Michael J King. Handling conflicting multiple objectives using pareto-based evolutionary algorithm during history matching of reservoir performance. In *SPE Reservoir Simulation Symposium*. OnePetro, 2013.

[106] Jaejun Kim, Joe M Kang, Changhyup Park, Yongjun Park, Jihye Park, and Seojin Lim. Multi-objective history matching with a proxy model for the characterization of production performances at the shale gas reservoir. *Energies*, 10(4):579, 2017.

[107] Zheng Zhang, Hye Young Jung, Akhil Datta-Gupta, and Mojdeh Delshad. History matching and optimal design of chemically enhanced oil recovery using multi-objective optimization. In *SPE Reservoir Simulation Conference*. OnePetro, 2019.

[108] Junko Hutahaean, Vasily Demyanov, and Mike Christie. Many-objective optimization algorithm applied to history matching. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

[109] Mike Christie, Colin MacBeth, and Sam Subbey. Multiple history-matched models for teal south. *The Leading Edge*, 21(3):286–289, 2002.

[110] Tony Hallam, Colin MacBeth, Romain Chassagne, and Hamed Amini. 4d seismic study of the volve field-an open subsurface-dataset. *First Break*, 38(2):59–70, 2020.

[111] Jude Lubega Musuuza, David Gustafsson, Rafael Pimentel, Louise Crochemore, and Ilias Pechlivanidis. Impact of satellite and in situ data assimilation on hydrological predictions. *Remote Sensing*, 12(5):811, 2020.

[112] E Essouayed, E Verardo, A Pryet, RL Chassagne, and O Atteia. An iterative strategy for contaminant source localisation using glma optimization and data worth on two synthetic 2d aquifers. *Journal of contaminant hydrology*, 228:103554, 2020.

[113] G Corte, R Chassagne, and C MacBeth. Seismic history matching in the pressure and saturation domain for reservoir connectivity assessment. In *82nd EAGE Annual Conference & Exhibition*, volume 2021, pages 1–5. European Association of Geoscientists & Engineers, 2021.

[114] Zhen Yin, Colin MacBeth, and Romain Chassagne. Joint interpretation of interwell connectivity by integrating 4d seismic with injection and production fluctuations. In *EUROPEC 2015*. OnePetro, 2015.

[115] Daniel Rahon, Paul Francis Edoa, and Mohamed Masmoudi. Identification of geological shapes in reservoir engineering by history matching production data. *SPE Reservoir Evaluation & Engineering*, 2(05):470–477, 1999.

[116] Cristina CB Cavalcante, Célio Maschio, Antonio Alberto Santos, Denis Schiozer, and Anderson Rocha. History matching through dynamic decision-making. *PloS one*, 12(6):e0178507, 2017.

[117] Jérôme E Onwunalu and Louis J Durlofsky. Application of a particle swarm optimization algorithm for determining optimum well location and type. *Computational Geosciences*, 14(1):183–198, 2010.

[118] Richard Rwechungura, Mohsen Dadashpour, and Jon Kleppe. Application of particle swarm optimization for parameter estimation integrating production and time lapse seismic data. In *SPE offshore Europe Oil and Gas Conference and Exhibition*. OnePetro, 2011.

[119] Baehyun Min, Joe M Kang, Sunghoon Chung, Changhyup Park, and Ilsik Jang. Pareto-based multi-objective history matching with respect to individual production performance in a heterogeneous reservoir. *Journal of Petroleum Science and Engineering*, 122:551–566, 2014.

[120] Baehyun Min, Joe M Kang, Hoyoung Lee, Suryeom Jo, Changhyup Park, and Ilsik Jang. Development of a robust multi-objective history matching for reliable well-based production forecasts. *Energy Exploration & Exploitation*, 34(6):795–809, 2016.

[121] BM Negash, Mohammed A Ayoub, Shiferaw Regassa Jufar, and Aban John Robert. History matching using proxy modeling and multi-objective optimizations. In *ICIPEG 2016*, pages 3–16. Springer, 2017.

[122] YM Han, Changhyup Park, and Joo Myung Kang. Estimation of future production performance based on multi-objective history matching in a waterflooding project. In *SPE EUROPEC/EAGE annual conference and exhibition*. OnePetro, 2010.

[123] Osho Ilamah. A multiobjective dominance and decomposition algorithm for reservoir model history matching. *Petroleum*, 5(4):352–366, 2019.

[124] William La Cava, Lee Spector, and Kourosh Danai. Epsilon-lexicase selection for regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 741–748, 2016.

[125] Arash Mirzabozorg, Long Nghiem, Zhangxing Chen, and Chaodong Yang. Differential evolution for assisted history matching process: Sagd case study. In *SPE Heavy Oil Conference-Canada*. OnePetro, 2013.

[126] Yasin Hajizadeh, Mike Christie, and Vasily Demyanov. History matching with differential evolution approach; a look at new search strategies. In *SPE EUROPEC/EAGE annual conference and exhibition*. OnePetro, 2010.

[127] Michael O'Neill and Lee Spector. Automatic programming: The open issue? *Genetic Programming and Evolvable Machines*, 21(1):251–262, 2020.

[128] Jonathan Kelly, Erik Hemberg, and Una-May O'Reilly. Improving genetic programming with novel exploration-exploitation control. In *European Conference on Genetic Programming*, pages 64–80. Springer, 2019.

[129] Thomas Helmuth and Lee Spector. Explaining and exploiting the advantages of down-sampled lexicase selection. In *ALIFE 2020: The 2020 Conference on Artificial Life*, pages 341–349. MIT Press, 2020.

[130] Thomas Helmuth and Lee Spector. General program synthesis benchmark suite. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1039–1046, 2015.

[131] Erik Hemberg, Jonathan Kelly, and Una-May O'Reilly. On domain knowledge and novelty to improve program synthesis performance with grammatical evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1039–1046, 2019.

[132] Dominik Sobania and Franz Rothlauf. Teaching gp to program like a human software developer: using perplexity pressure to guide program synthesis approaches. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1065–1074, 2019.

[133] Yifan He, Claus Aranha, and Tetsuya Sakurai. Incorporating subprograms as knowledge in program synthesis by pushgp and adaptive replacement mutation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 554–557, 2022.

[134] Edward Pantridge and Lee Spector. Pyshgp: Pushgp in python. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1255–1262, 2017.

[135] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[136] Stefan Forstenlechner, David Fagan, Miguel Nicolau, and Michael O'Neill. Extending program synthesis grammars for grammar-guided genetic programming. In *International Conference on Parallel Problem Solving from Nature*, pages 197–208. Springer, 2018.

[137] Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models. *arXiv preprint arXiv:2206.08896*, 2022.

[138] Federico Pigozzi, Yujin Tang, Eric Medvet, and David Ha. Evolving modular soft robots without explicit inter-module communication using local self-attention. *arXiv preprint arXiv:2204.06481*, 2022.