SPATIAL GENETIC PROGRAMMING

By

Iliya Miralavy

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science—Doctor of Philosophy

2024

ABSTRACT

Space, while inherent to the natural world, often finds itself omitted in bio-inspired computational system designs. Spatial Genetic Programming (SGP) is a Genetic Programming (GP) paradigm that incorporates space as a fundamental dimension, evolving alongside Linear Genetic Programming (LGP) programs. In SGP, each individual model is represented by a 2D space consisting of one or many LGP programs. These programs execute in an order controlled by their spatial position. The contributions of this work are: Introducing SGP as a tool for studying evolution of space in GP. Application of the proposed system to a range of problems including symbolic regression, classic control and decision-making problems and a comparison to other common GP paradigms. A study on how spatial dimension influences generational diversity, on emergence of spatially-induced localization within the system, and on the emergence of iterative structures within the system. The findings of this research open new avenues towards a better understanding of natural evolution and how the dimension of space could be useful as a handle for controlling important aspects of evolution.

ACKNOWLEDGMENTS

First off, I want to thank my advisor, Dr. Wolfgang Banzhaf, who welcomed me into his lab, was always patient with me, and supported me through every single step of my journey. His exemplary passion for knowledge and science, along with his attitude toward constructive collaboration and his guidance shaped my way of thinking and was extremely helpful in finishing this manuscript. I would like to thank my committee, Dr. Punch, Dr. Ofria, and Dr. Hintze. Through their constructive criticism, kind attitude, and guidance, I managed to choose a proper pathway toward selecting the correct research topic. I am thankful for my wife, Pegah, and my friend, Nima. They were always there for me, understood me, and backed me up no matter how hard it got sometimes. I am thankful for my family and friends who supported me even though they were miles away. And last but not least, I'm thankful for my friends, Reza and Mohammad. I cherish every memory that we made together; Mohammad's kindness and Reza's understanding of life were uncanny. Even though they are gone now, they always believed in me, and their belief made me a stronger person.

TABLE OF CONTENTS

Chapter 1

Introduction

Parts of this chapter have been adapted from [Miralavy and Banzhaf, 2023a], and some others are reproduced with permission from Springer Nature from [Miralavy and Banzhaf, 2023c].

1.1 Background and Motivation

Nature presents a wide range of complex problems. Over millions of years, natural evolution has developed biological systems capable of solving these problems, allowing organisms to survive, reproduce, and pass down their genetic information to the next generation. Even the highly sophisticated human brain, the locus and source of our intelligence, is not the limit to what can evolve through this process. For example, in the past 3,000 years, the continuous process of evolution has caused the spatial volume of human brains to significantly decrease in order to enhance its efficiency [Hofman, 2014] which shows that even such highly evolved biological systems are still under evolution. Biological systems are often intricate, composed of numerous diverse elements. At times, multiple biological systems collaborate to achieve a unified goal. Consider how the immune system, the respiratory system, the skeletal system, the nervous system, and the digestive system in a mammal work together to ensure the host's survival. The capabilities of natural systems have caused many researchers to study them and draw inspiration from them to solve problems in various frameworks,

including computational ones. While the complexity of biological organisms serves a purpose and often arises out of necessity, computational models inspired by them are abstractions. These abstractions aim not only to simplify the complexity of biological systems but also to adapt them to a computational environment rather than a biological one. However, it is a challenge in creating bio-inspired algorithms to determine the necessary level of abstraction without diminishing the power of their natural counterparts [Vieu, 1997].

Space and time constitute the fundamental framework within which natural evolution unfolds. Between the two, space is often the element abstracted away in bio-inspired systems, due to the inherent complexity stemming from its multidimensional nature. However, in natural evolution space plays a critical role in determining the habitat of organisms, their morphology, and access to environmental resources. It also naturally enables organisms to work in parallel and facilitates interaction among natural systems [Vieu, 1997]. For instance, within the human body, various biological systems operate concurrently in distinct spatial locations, such as the nervous system. An intriguing aspect of biological systems is their spatial locality [Chen and Konstantinides, 2022]. These systems exhibit high modularity, with elements within each module typically situated in close spatial proximity. In organisms like plants, the spatial arrangement of components significantly influences their function. For example, roots must be underground to access resources, while leaves need to be positioned to capture sunlight for photosynthesis. An unresolved question arises: should space be abstracted away when developing new bio-inspired systems, or should it be retained in computational models? It is evident that, thus far, achieving the same functional prowess as natural systems in their computational counterparts has proven challenging. This fact, in addition to the lack of related research in previous literature, justifies the ongoing exploration of space within computational frameworks as a valid and promising approach.

When we abstract space from a bio-inspired system, the result becomes simpler and more suitable for implementation within a computational framework. This abstraction also entails the removal of the capabilities that space contributes to the system, preventing

spatial phenomena from emerging. One of the primary reasons for abstracting space away in classical algorithms has historically been the limitations of technology. However, with recent advancements, these computational constraints have become significantly less pronounced, prompting a re-evaluation of the conventional perspective about such abstractions. It is noteworthy that space, as a dimension, inherently facilitates parallelization and can, in some cases increase the computational speed at which the evolutionary system operates. Another challenge arises when designing more complex algorithms that are less abstract: the models can become too complicated to understand. One of the crucial benefits of incorporating space in computational problem-solvers is the natural facilitation of classification and management of entities, as well as that of the visualization and understanding of the elements contained within. Spatial analysis has been an approach incorporated in the vast majority of scientific fields through visualizations, generation of graphs, maps and so on potentially making it less of an obstacle when it comes to the complexity of algorithms.

My general interest is in studying bio-inspired problem solvers and investigating approaches to enhance them. Specifically, I am interested in examining the role of space in natural evolution and utilizing it as a primary dimension in the evolution of computational problem solvers and specially in Genetic Programming (GP).

To achieve this goal, this dissertation introduces Spatial Genetic Programming (SGP), a Genetic Programming (GP) paradigm in which individuals are spatial, and space significantly influences the execution of individual models. I present how Spatial Genetic Programming (SGP) can be configured and utilized as a tool for studying spatial GPs. As a proof of concept, I apply SGP to various problem classes and, using spatial analysis, I explore the effects of space on the evolution of SGP models. To initiate an investigation in this vast area of research, I target two primary criteria — diversity and localization — when analyzing the impact of space on SGP individuals. Maintaining a healthy diversity in an evolutionary algorithm can enhance the algorithm's performance by enabling it to cover more parts of the search space in a population of individual models. Thus, it is beneficial to investigate

whether the dimension of space can add diversity to a computational framework, similar to what occurs in nature through environmental conditions and their changes. Observing nature, the localization of natural entities that serve distinct purposes is a common phenomenon. I investigate whether similar patterns can emerge in the computational framework of SGP. Additionally, I investigate how spatial organizations provide a unique method for evolving iterative structures.

1.2 Evolutionary Computation

Evolutionary Computation (EC) is a sub-field of Artificial Intelligence (AI) in which inspirations from natural evolution are incorporated as a metaphor for solving computational problems through search and optimization [Dumitrescu et al., 2000]. The advent of the general idea of EC dates back to the invention of computers, with Evolutionary Programming [Fogel, 1962], Evolution Strategies [Rechenberg, 1973, Schwefel, 1981], and Genetic Algorithms [Holland, 1975, Goldberg, 1989] emerging between the 1960s and 1980s, followed by the introduction of GP [Koza, 1992a] in the early 1990s [Banzhaf et al., 1998]. Models that are studied in EC are usually referred to as an Evolutionary Algorithms (EA).

Natural evolution [Darwin, 1909] adapts populations of organisms to challenges posed by their surrounding environment through variation and reproduction. The fittest organisms are those that can survive under environmental selection pressures and can pass their genetic markers to the next generation through reproduction. Variations in these genetic markers accidentally occur during reproduction and might result in deleterious, neutral, or beneficial outcomes for the phenotypic traits of the offspring. Over generations of evolution, organisms undergo changes to become adapted for surviving in their environments.

In EAs, the fundamental essence of natural evolution is harnessed to evolve computational representations through trial and error, serving specific purposes such as solving tasks or computationally modeling a system. Figure 1.1 presents an overview of the core processes commonly used in EA, where a population of individual solutions is evolved to find an

optimized solution to a given problem. It is crucial to determine a representation of individuals capable of solving the specified problem. In the *initialization* step, a population of individuals is randomly initialized as the first generation. Next, in the *evaluation* step, the fitness of these individuals is assessed. In terms of natural evolution, fitness refers to an organism's ability to survive; however, in EA, fitness is defined as a quantified measurement of how well an individual solution performs in relation to the given problem. Since the first generation is randomly initialized, it typically does not contain individuals with satisfactory fitness levels. Nonetheless, the population undergoes a *selection* phase, where, following a selection scheme, a set of the best individuals, based on their fitness values, are chosen to create the next generation. In the *variation* phase, the selected individuals, serving as parents, undergo crossover and mutation. During crossover, usually two parent individuals are selected to create one or more offspring by combining parts of their genetic markers. An offspring created from crossover inherits traits from its parents, meaning crossover works towards converging the population of individuals, making the offspring more similar in their representation. Conversely, mutation, as an evolutionary operator, introduces diversity into the population and allows evolution to explore different regions of the fitness landscape by introducing random alterations in the genetic markers of the offspring. Variation causes diversity in the population, and maintaining diversity is a crucial task in evolutionary computation. Evaluation, selection, and variation form an iterative process, where in each iteration, a new generation of individuals replaces the former one. This process continues until a termination condition is met, such as reaching a limit on the number of generations or finding an adequately optimized solution.

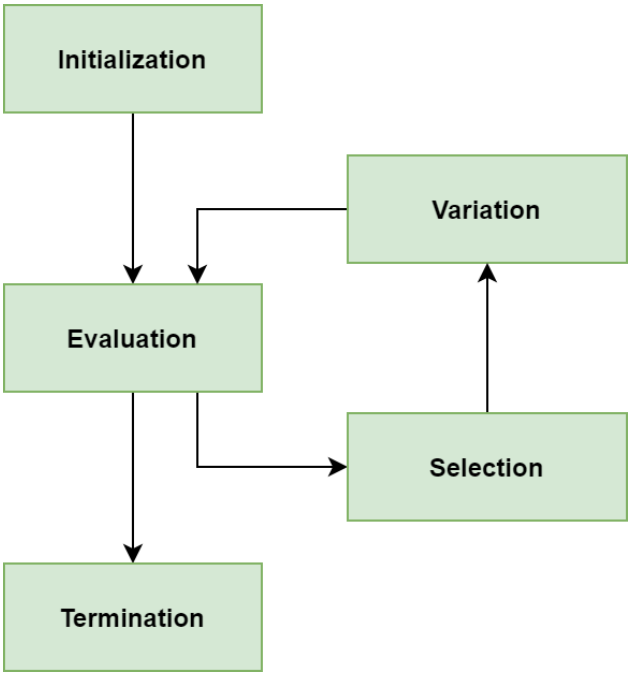


Figure 1.1: General principle of Evolutionary Algorithms

1.3 Genetic Algorithm

Genetic Algorithm (GA) [Sastry et al., 2005, Holland, 1975] is a search-based optimization algorithm that is inspired by the Darwinian conception of natural evolution. A GA follows the core features of EAs. Figure 1.2 shows a flowchart that represents how a GA operates. First, a generation of individuals is randomly generated as the initial population. Individuals should be represented by a data structure suitable for defining the target optimization problem in a computational framework. A common approach is to use bit strings, as illustrated in the figure. The next step involves evaluating the fitness or the performance level of every individual. This step highly depends on the target problem, and various metrics can be incorporated to quantify the performance level of the individuals. Subsequently, a termination condition is checked. This condition can be a certain number of evolutionary generations or finding a satisfactory solution. If the termination condition is not reached, then a selection mechanism is applied to the population to select a number of individuals to recombine and mutate to form the next generation. For example, a conventional selection algorithm used in GAs is

the proportional selection, roulette wheel [Holland, 1975]. In this algorithm, a proportional fitness value is calculated for every individual by dividing their fitness by the sum of all fitness values in the population. Each individual's proportional fitness corresponds to a slice of the roulette wheel (represented as a pie chart in Figure 1.2). A random number between 0 and 1 is then selected to determine the slice representing an individual that is chosen to be a parent for recombination. The whole evolutionary process described in the figure iterates until the termination condition is met, and an output is produced by the system.

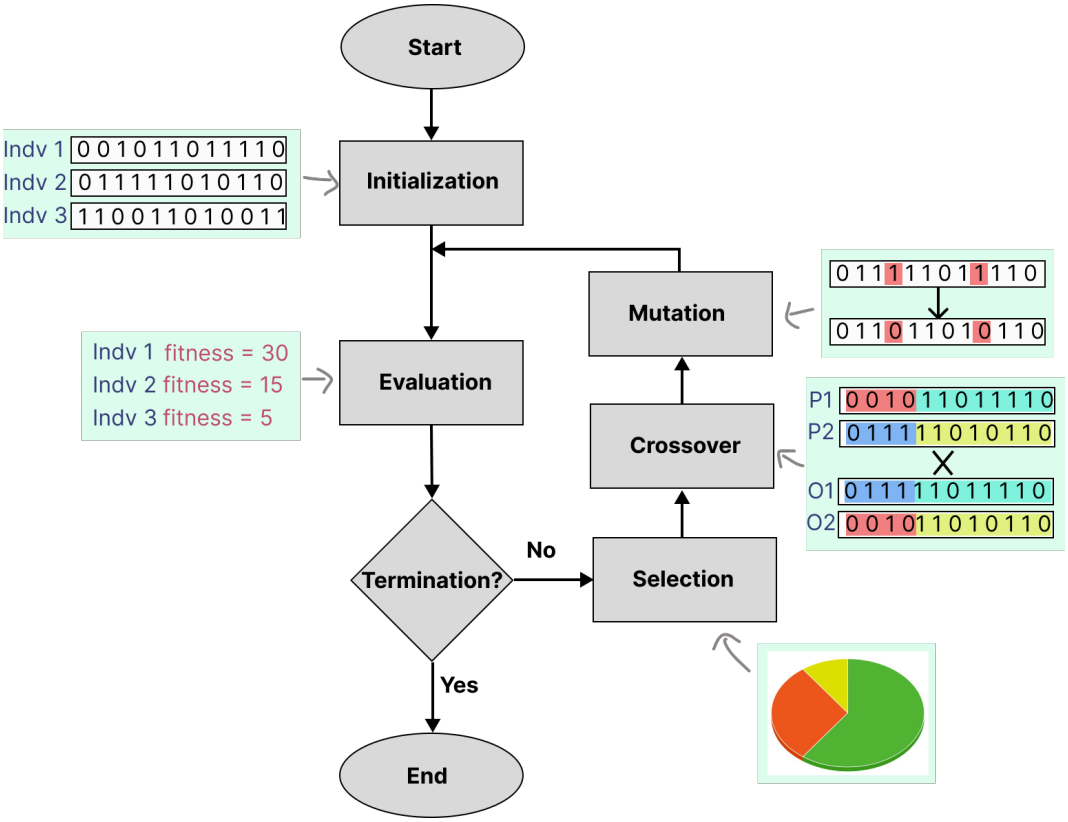


Figure 1.2: A flowchart of the Genetic Algorithm

Evolutionary operators in the context of a GA or generally in EAs are responsible for promoting variation within the population to effectively explore and exploit the search space. *Crossover* [Eiben and Smith, 2015] combines traits from two parent individuals in a generation to create offspring that share characteristics of both parents. Crossover can enhance the performance of an EA [Doerr et al., 2008] and, when combined with other

evolutionary operators, can be utilized to introduce a healthy amount of diversity to the population [Dang et al., 2017, McGinley et al., 2011]. Choosing a proper crossover operator highly depends on the target problem of the EA algorithm and there is no perfect recipe for this operator [Back and Schwefel, 1996]. Figure 1.2 depicts a point crossover algorithm for the bit string representation, in which parts of P1 and P2, which are the two selected parents, recombine to form O1 and O2, representing the two offspring.

Mutations [Eiben and Smith, 2015] are other types of evolutionary operators tasked with introducing diversity to the population of individual models. Similar to crossover, different mutational strategies can be beneficial depending on the specific problem at hand. During mutation, a random change is introduced to the individual's representation. Figure 1.2 shows a point-mutation algorithm which is applied to the bit string representation. Note that the two randomly selected sites, marked by red squares, have their values changed from 1 to 0.

1.4 Genetic Programming

GP [Koza, 1992a] is an extension of GA, in which computational problem-solving models—usually in the form of computer programs—are evolved and optimized over a repeated generational cycle. GP can be considered an evolutionary machine learning technique [Banzhaf et al., 1998]. There are crucial differences between a GA and GP. While a GA uses a fixed-size representation, the representation size in GP may vary. There are also differences in the forms of model representations and genetic operators between the two evolutionary techniques [Woodward, 2003]. Although all GP algorithms follow a core process inspired by natural evolution, they come in various forms and representations. Typically, the GP process also adopts the core principles of EA by starting with a population of random individuals representing unknown solutions to a given problem. A predefined fitness function evaluates these individuals to measure how well they solve the problem. A selection mechanism is then used to choose parent individuals that will undergo evolutionary operators (crossover and mutation) to form the next generation of the population. Sometimes, an additional

survival selection step called *Elitism* is implemented to ensure the best offspring remain unchanged for inclusion in the population. The cyclical process continues until a termination condition, such as finding a model that satisfies user requirements, is met. GP models are interpretable, allowing industry domain experts to understand and evaluate them. The following six preparatory steps should be determined to utilize GP for solving a computational problem [Koza, 1994]:

1. **Representation:** The encoding of individuals in the GP system is referred to as the GP representation.
2. **Terminal Set:** Terminals are the inputs or constant values in a GP system. They can serve as operands for functions or operators.
3. **Function Set:** The function set includes functions or operators that are defined based on the target problem. Examples include mathematical operators, which can be utilized to solve problems within a mathematical framework, such as symbolic regression.
4. **Evaluation Function:** An evaluation function, also known as the fitness function, provides a quantified measure of an individual's fitness. Biologically, higher fitness values indicate fitter individuals. However, in problems aiming to minimize an error value, the error measurement can serve as the fitness indicator, where lower values correspond to fitter individuals.
5. **Configuration Parameters:** These parameters are used to configure the evolutionary process. Examples include the size of the starting population and the mutation rates.
6. **Termination Condition and Result Designation:** Since GP is a cyclic algorithm, conditions must be established to break the cycle. A common practice is to set a threshold for the number of generations before terminating the process. Additionally, a mechanism should be designated to preserve the best results of the evolution.

1.5 GP Representations

Various works in the literature focus on evolving graphs capable of representing solutions to computational problems. Tree GP (TGP) [Koza, 1992a] is the most common type of GP where tree data structures represent models. A *tree* is an acyclic undirected graph where only one edge can connect any two vertices [Bender and Williamson, 2010], making it an excellent candidate for evolving mathematical equations and conditional pathways for decision-making problems. Figure 1.3 illustrates a sample tree representation for a GP individual, denoting a mathematical equation.

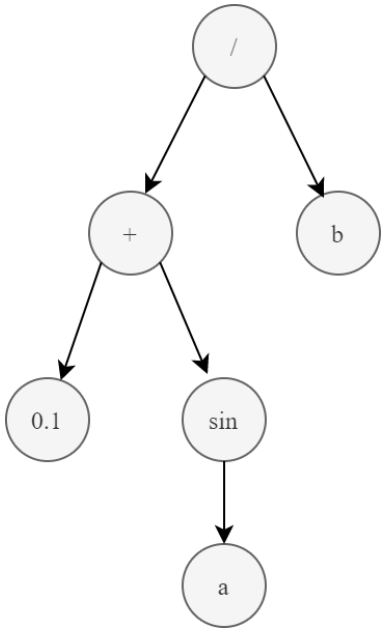


Figure 1.3: An example TGP. 0.1, a and b are among the terminal set, and / and + are among the function set of this tree

TGP has been used for various applications such as transportation [Yao and Hsu, 2009], symbolic regression [Augusto and Barbosa, 2000, Amir Haeri et al., 2017], image processing [Tran et al., 2016], classification [Aoki and Nagao, 1999], and others. Although the tree data structure is straightforward for understanding and evolving solutions, traversing these structures is not computationally trivial and often leads to bloat problems.

Cartesian Genetic Programming (CGP) [Miller, 1999] is another mainstream graph-

evolving GP paradigm that has shown good performance in solving computational problems. CGP uses integer values as genes to represent nodes in a graph, their functions, links between the nodes, and the connections of inputs and outputs to these nodes. Compared to TGP, CGP is computationally less demanding, resulting in faster evaluation times and a reduced tendency for bloat [Miller, 2020]. An interesting feature of CGP is its ability to encode and control computational systems similar to Artificial Neural Networks [Turner and Miller, 2013, Khan et al., 2013]. CGP has also been applied in agent control [Harding and Miller, 2005], image processing [Harding et al., 2013], and circuit design [Hodan et al., 2021].

It is possible to evolve GP models that do not rely directly on graph representations. Linear Genetic Programming (LGP) is among these types of GP, represented as a series of instructions, typically in the form of an imperative programming language or machine language, that execute sequentially. LGP supports branching operators, which allow the execution pointer to jump between instructions. A particular challenge of LGP is determining the correct number of internal registers; an incorrect choice can drastically affect the performance of the solutions [Oltean and Grosan, 2003]. However, LGP programs can be quite fast because they can be designed to run directly on the processor. Another GP variant that does not use graphs for representation is Stack-based Genetic Programming [Perkis, 1994]. In this paradigm, stack-based operations are used to manage operands for the program operators from a data stack and push the results back onto the stack. Depending on the rules, multiple data stacks for different data types might exist. Generally, Stack-based Genetic Programming models are faster than tree structures, and it is possible to create bloat-free mutations and crossover mechanisms. Push GP [Spector, 2001] is one of the most notable stack-based systems and has been used for various applications such as automatic code simplification [Helmuth et al., 2017] and Python code synthesis [Pantridge and Spector, 2020].

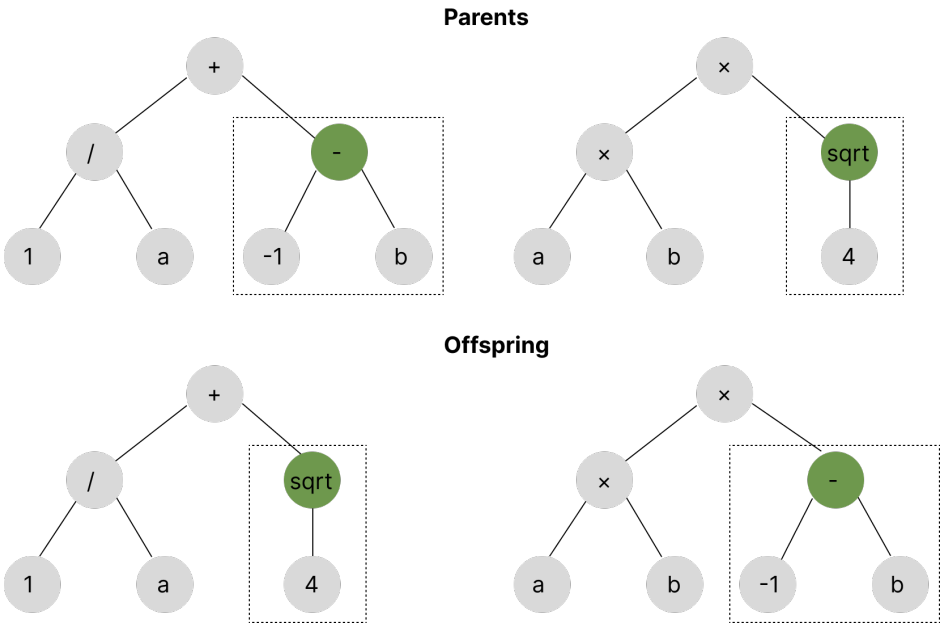


Figure 1.4: Point crossover in a TGP

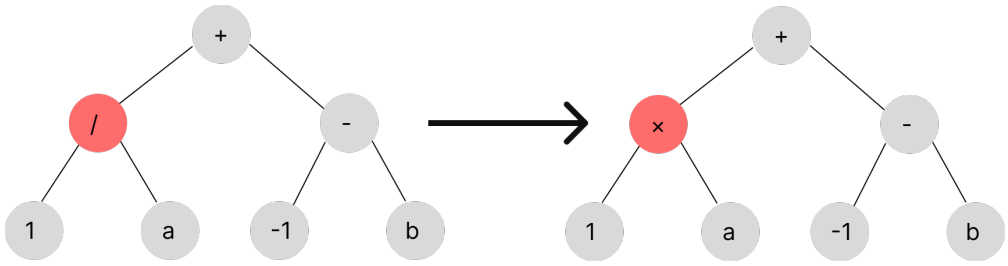


Figure 1.5: Point mutation in a TGP

1.5.1 Crossover and Mutation in TGP

Mutation and crossover in GP are highly similar in principle to those in other EAs, with the primary distinction being the different representations they target.

Figure 1.4 depicts an example of a crossover approach known as point-crossover [Poli and Langdon, 1998] within a tree representation. A common node (represented in green in the figure) is chosen from two parent trees. The subtrees connected to this node are then exchanged to form two new offspring. While most characteristics of the offspring are similar to those of their parents, the exchanged subtrees introduce distinct differences. Numerous studies have examined crossover algorithms to enhance the evolution of models in GP systems [Clegg et al., 2007, Beadle and Johnson, 2008, Poli et al., 1998].

Figure 1.5 illustrates a point-mutation [Poli and Langdon, 1998] occurring in a tree structure. The node selected for mutation (highlighted in red) is changed from a division operator to a multiplication operator, while the terminal nodes connected to it remain unchanged. If the nature of the newly selected operator is incompatible with the old terminal nodes, then these terminals may also be subject to random changes. The exact approaches used for such modifications depend on the design decisions of the underlying mutation algorithm. The study of mutational algorithms is a significant aspect of the literature [Piszcz and Soule, 2006, Langdon et al., 2010, Beadle and Johnson, 2009].

1.5.2 Crossover and Mutation in LGP

In the context of this dissertation, the LGP paradigm is extensively incorporated. Therefore, it is particularly important to discuss the common approaches taken to add variety to LGP populations. Figure 1.6 illustrates a two-point crossover algorithm. Two random points are chosen for each individual, as indicated by the lines in the figure, to describe a randomly selected region (marked with red and blue rectangles in the figure). During the crossover, the instructions located in the randomly selected region of parent A are transferred directly to offspring B, while the remaining instructions for offspring B are taken from parent B.

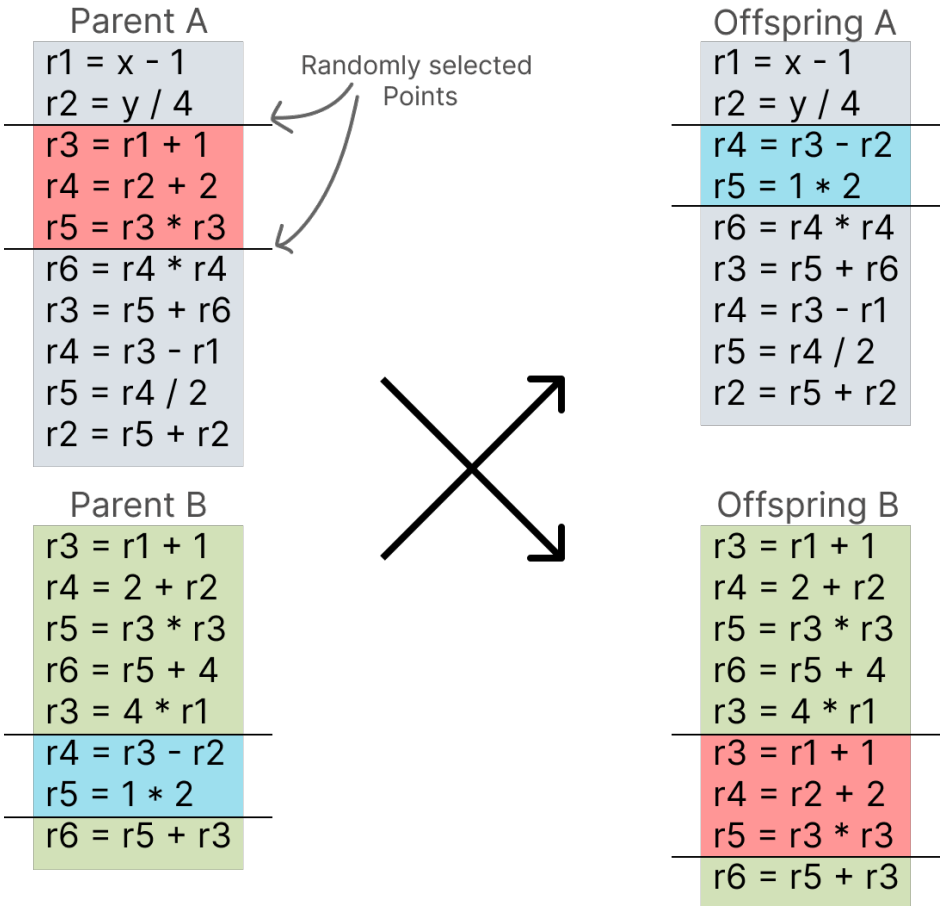


Figure 1.6: A two-point crossover algorithm in a LGP system

Conversely, offspring A consists of the randomly selected region from parent B, and the rest of the instructions are taken from parent A.

A variety of mutation schemes can be utilized for LGP programs. Figure 1.7 depicts two of these approaches. In the first mutation scheme, an operand in a randomly selected instruction of an individual is subjected to a random change, wherein register $r5$ is replaced with another register, $r4$. The second mutation scheme adds a randomly generated instruction to a random position within the individual, which is marked by the red rectangle. Other types of mutations in LGP aim to manipulate the program instructions to add diversity to the population. For example, there are mutation schemes that add instructions to an individual program or change the operator in an instruction rather than altering an operand.

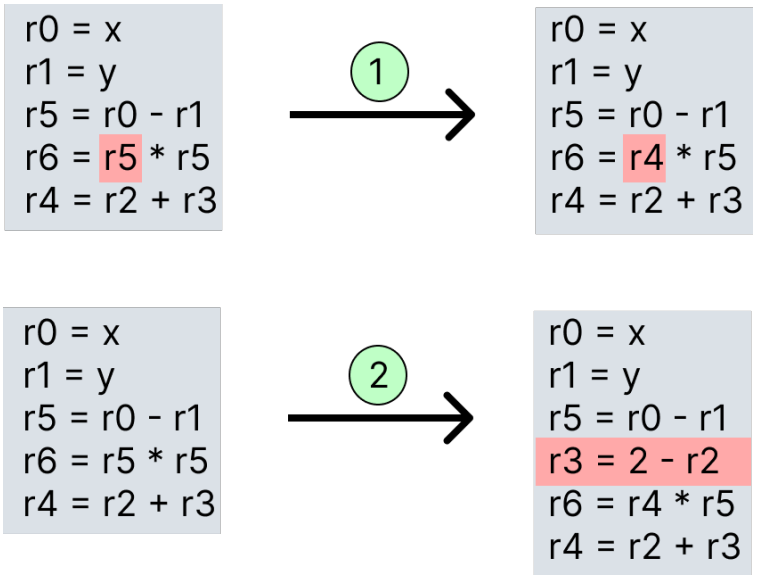


Figure 1.7: Two instances of different mutations occurring in a LGP system

1.6 Selection Mechanisms and Tournament Selection

Selection mechanisms determine which individuals from the current generation will survive and be chosen to undergo crossover and mutation, thereby forming the next generation. Among the most common selection approaches are Tournament [Blickle, 2000], Roulette Wheel, and Lexicase selection [Helmuth et al., 2014], with Tournament selection being the method used in the experiments conducted for this dissertation. As illustrated in Figure 1.8, this selection type involves randomly choosing a group of individuals from the population and ranking them based on their fitness values. In a multi-winner tournament, the individuals with the highest fitness values are then selected as parents. These parents undergo crossover and mutation to produce offspring that will become part of the next generation in the EA. Tournament selection is effective, straightforward to implement, robust, allows for parallelization, and can adjust the selection pressure by configuring the tournament size [Miller et al., 1995].

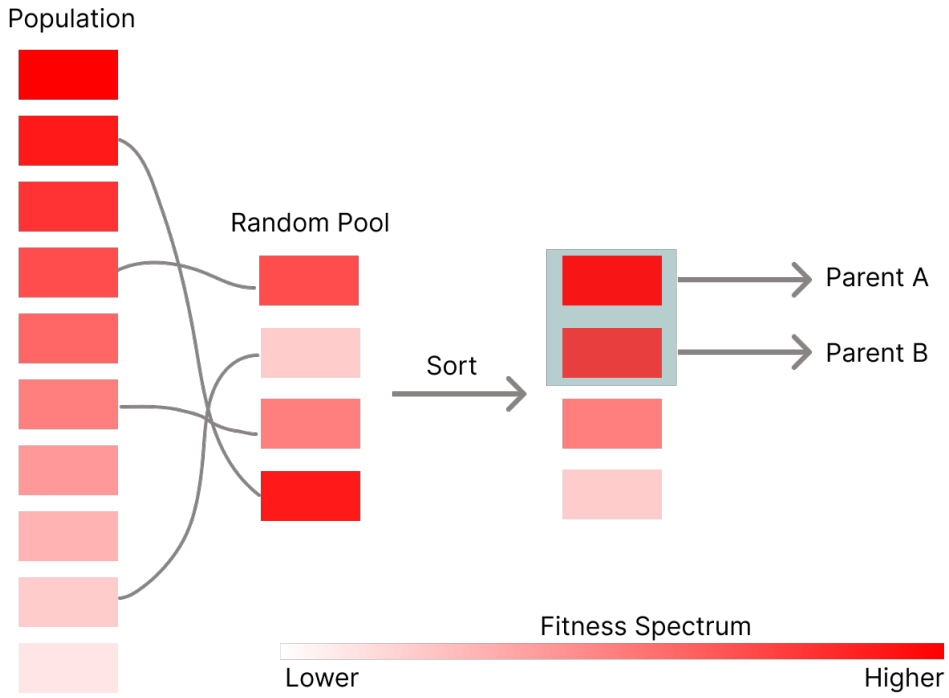


Figure 1.8: An overview of a multi-winner tournament selection

1.7 Space-Oriented Computing

While there is no universally accepted definition for Space-Oriented Computing in the literature, it is generally regarded as a field where computational models integrate space as a primary factor for optimization. The principles of space-oriented computing can be applied in GP when designing spatial GP algorithms. This field points out the importance of space in computational processes, highlighting three key reasons [DeHon et al., 2007]:

1. **Efficient Management:** In large computational systems with numerous components, space provides an intuitive means to manage, categorize, classify, and regulate interactions among these elements.

2. **Computational Power:** Spatial properties are crucial for the computational capabilities of certain systems. For example, the behavior of ant colonies or fish swarms cannot be accurately modeled without considering their spatial dynamics.

3. **Real-world Integration:** The integration of computational systems with the phys-

ical world necessitates a comprehensive understanding of spatial characteristics. For instance, architectural algorithms must consider the spatial properties of building components to ensure the validity of relevant computations.

These considerations are particularly relevant in the context of GP, where the spatial arrangement of individuals can significantly influence the evolutionary process and the resulting computational models.

Furthermore, akin to natural systems, space offers an inherent mechanism for enabling parallelization within computational frameworks [Gear, 1993, Narlikar and Blelloch, 1997]. Also, the ability to visualize spatial elements can simplify the analysis of these systems.

1.8 Research Objectives and Contributions

In my research, I aim to explore the often-overlooked dimension of *space* in bio-inspired systems and its applications for computational frameworks, and specially GP. Recognizing that many computational models traditionally neglect the dimension of *space*, my study is motivated by the following hypotheses:

1. Spatially represented individuals within GP systems can be constructed to achieve results comparable to those of common GP representations.
2. The incorporation of space as a primary dimension in GP evolution could enhance key aspects of the evolutionary algorithm, such as diversity, thereby improving its problem-solving capabilities.
3. In spatial GP, similar to natural processes, evolution prompts computer programs to localize and form clusters that fulfill specific functions.
4. Proximity in space can serve as a straightforward method for evolving iterative structures.

Specifically, my research explores the integration of the dimension of *space* in GP, examining its impact on forming iterative structures, diversity in the population, localization of

computer programs, and the construction of conditional pathways. I propose SGP, a GP paradigm where individuals are program nodes distributed across a 2D space, executing in an order determined by their spatial positioning. This dissertation presents SGP as a novel tool for the study of spatial GP dynamics.

In exploring the application of SGP to various problem types, I utilize SGP—primarily in its spatial mode—to evolve problem solvers, targeting a testbed of symbolic regression problems. I demonstrate that a spatially represented GP can produce results comparable to those of LGP [Brameier et al., 2007] and TGP [Koza, 1992b]. By introducing the programmatic mode of SGP, I identify an approach that combines the spatial properties of programs with logical statements to create conditional pathways for program execution within an SGP individual. I apply SGP mainly in its programmatic mode to a series of decision-making and classic control problems and compare its performance with that of LGP and TGP.

A significant portion of my research focuses on how spatial proximity within SGP affects the evolution of iterative loops and the precision in determining the number of iterations in evolved models. I present evidence that SGP, with its inherent spatial characteristics, exhibits a unique capability in evolving loop structures for solving problems that necessitate iteration. Through spatial analysis, I show the feasibility of evolving such complex structures by leveraging the spatial elements of SGP.

Furthermore, my research examines the impact of different spatial topologies on the performance of the SGP system. I conduct experiments with various constraints on 2D topologies, such as lattice and ring structures, and extend my exploration to 3D spatial configurations. These studies provide insights into how spatial structuring and topological constraints can direct and shape the evolutionary process within the SGP framework.

Another contribution of my study is the utilization of spatial evolutionary operators and their impact on SGP's performance. I explore scenarios in which different 2D regions exhibit varying mutation rates. The goal of this setup is to understand how these variations in spatial conditions affect the evolution of SGP models. This aspect of my research seeks to mimic

natural phenomena where environmental or spatial factors influence evolutionary processes.

Through this study, I hope to contribute to a broader understanding of spatial dimensions in computational systems, emphasizing how spatial characteristics can alter the behavior and performance of GP systems. I hope that my findings open new pathways for incorporating spatial elements into various computational paradigms, potentially leading to novel approaches in problem-solving and evolutionary computing.

My findings can also initiate discussions on potential future research topics. I hope that this work serves a foundation for numerous studies on the impact of spatial dimensions on the evolution of nature-inspired models, raising various research questions. For instance:

- What other aspects of the evolutionary process are altered through the introduction of the dimension of space?
- How can spatial elements provide a handle for controlling localization, diversity, or bloat?
- Are there other dimensions or fundamental concepts that should be considered when designing AI or evolutionary algorithms?
- Does constraining the spatial dimension through the employment of spatial topologies affect the evolutionary process for solving various classes of problems? If so, how can we benefit from it?
- How can the incorporation of space make a GP system more interpretable or understandable through spatial analysis?
- Is it possible that the localization emerging through spatial GP leads to a better or easier understanding of the characteristics of individuals in a population?

These questions focus on the potential for spatial considerations to influence the future of genetic programming approaches.

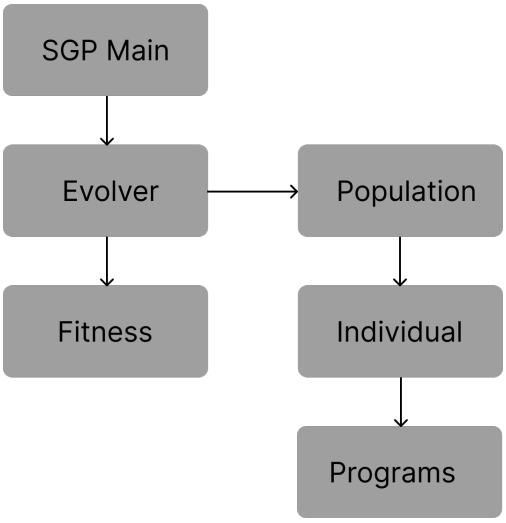


Figure 1.9: A high-level architecture of SGP

1.9 Research Methods

I conduct most of my experiments using the SGP framework that I developed over the course of my research. The modularity and flexibility of SGP are achieved through a configurable setup that positions it as a potential framework for investigating individual models in spatial GP. The high-level architecture of the system is depicted in Figure 1.9. The main method of SGP orchestrates the system's initialization by linking various essential modules, each governing a distinct aspect of the algorithm. Implementations of modules that follow the requirements of SGP can replace the existing ones.

The core loop of the GP system resides within the Evolver module, which essentially acts as the evolutionary algorithm responsible for evolving GP models. The Evolver oversees the Population and Fitness modules. The Population module is tasked with creating and maintaining a population of individuals, while the Fitness module evaluates each individual's performance within the Evolver. The Evolver then selects the fittest individuals, executes evolutionary procedures like crossover and mutation, saves the top-performing elite individuals, and provides updates on the status of each generation.

SGP is equipped with a suite of tools for analyzing SGP populations. The *SGP Analysis Tool* performs multiple analyses on a given population of SGP individuals. For example, Fig-

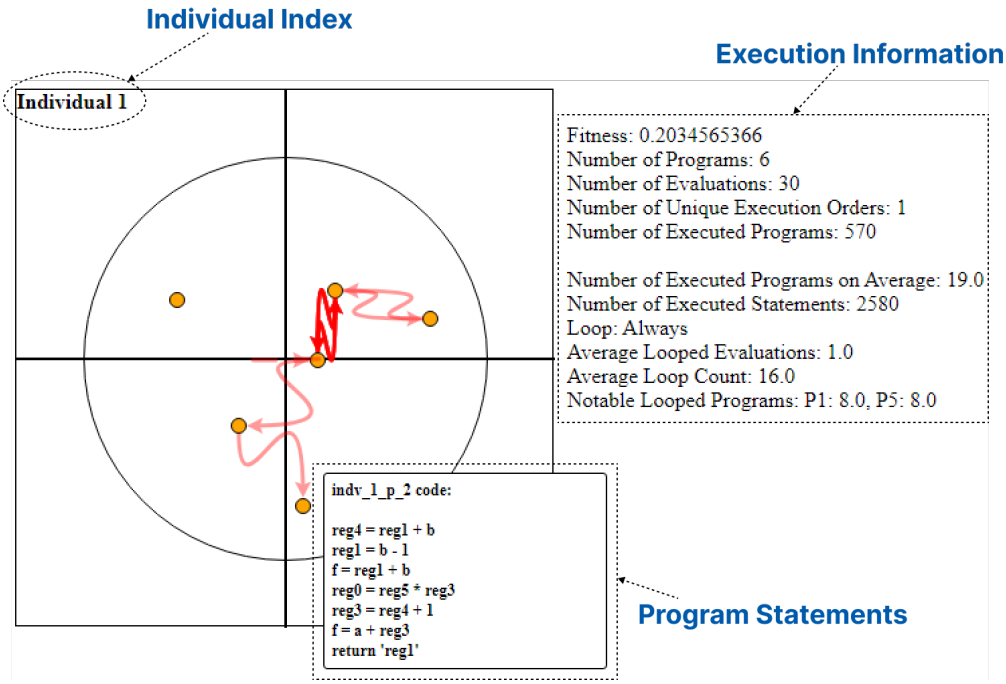


Figure 1.10: SGP analysis tool

ure 1.10 provides information about an individual's index, execution details, each program's statements, the spatial position of each program, and the order of execution for the individual (indicated by red arrows).

This tool generates graphs that depict the evolutionary progression of the best individuals and the average of the population. Additionally, the analysis tool creates spatial heatmaps of the population, which illuminate common traits among individuals through the localization of programs (see Figure 1.11). These graphs and heatmaps are produced using the public Python libraries *matplotlib* [Hunter, 2007] and *seaborn* [Waskom, 2021]. For data analysis, the *pandas* [development team, 2020] and *numpy* [Harris et al., 2020] libraries are utilized.

For the experiments involving the implementation of the TGP algorithm, I utilized the *DEAP* library [Fortin et al., 2012]. This library offers a comprehensive framework and guidelines for employing various evolutionary algorithms and provides statistical tools for their analysis. For experiments addressing classic control problems, I used OpenAI's Gymnasium library. This resource encompasses a suite of benchmarks designed for solving a diverse array of problem classes, including classic control challenges.

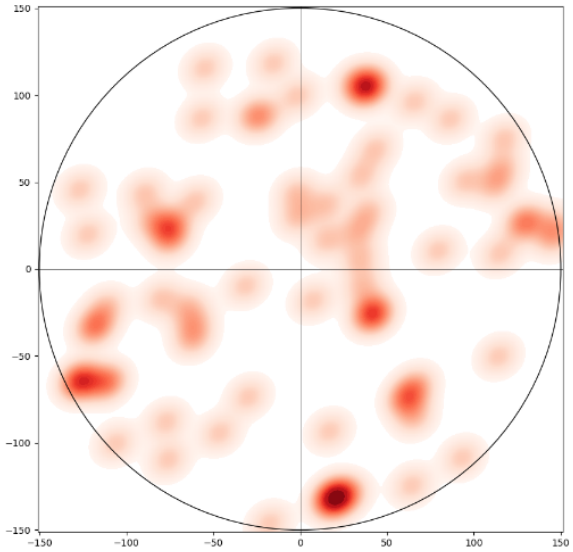


Figure 1.11: An example heatmap generated by the SGP analysis tool. Hotter points represented by increased hue of red represent localized clusters of programs

The majority of the experiments were conducted with the support of computational resources from the Institute for Cyber-Enabled Research at Michigan State University [MSU, 2023]. ChatGPT-4 [OpenAI, 2023] was employed to review parts of this manuscript for grammatical accuracy. Finally, the iThenticate [iParadigms, 2023] tool, provided through Michigan State University, was used to make sure the work complies with the guidelines of responsible research conduct.

1.10 Outline of Dissertation

The chapters of this thesis explore the utilization of SGP for solving various types of problems and examine the role of space in the evolution of its individual models.

Chapter 2 introduces SGP as a tool for studying space within GP, explaining the SGP algorithm and its spatial representation of individuals. This chapter also details the execution of SGP models, the conditions under which they evolve, the operators used, and the process for adding new operators. It discusses how the system can be configured for different problem scenarios and describes the problem benchmarks included in the SGP system.

Chapter 3 conducts a spatial analysis of the system, demonstrating how iterative behaviors

and loop structures can emerge in SGP. It investigates the localization of programs within the system, examines the impact of different spatial topologies, explores various evolutionary operators, and measures the system's diversity.

In Chapter 4, SGP is applied to symbolic regression problems as a proof of concept, with an analysis focused on system performance.

Chapter 5 applies SGP to different classes of classic control and decision-making problems in a distinct mode. It demonstrates how incorporating logical elements to determine the execution order of SGP programs can introduce impactful conditional pathways.

Finally, Chapter 6 concludes the results presented in the previous chapters of this dissertation and discusses potential future research opportunities that arise from this work.

1.11 Summary

When EAs are applied properly, they serve as potent methods for addressing a wide array of computational challenges. Genetic Programming (GP), in particular, leverages the distinctive processes of biological evolution for problem-solving and machine learning applications by evolving computer programs. Examining the differences between bio-inspired models like GP and their natural counter-parts can result in insights into the necessary level of abstraction and the potential trade-offs involved.

In this dissertation, I introduce and explore a novel variant of GP known as Spatial Genetic Programming (SGP), where individuals are defined within a spatial context. I demonstrate that SGP can achieve results comparable with conventional GP methods while outperforming them in some cases. As a foundational step, I conduct a spatial analysis of SGP individuals, revealing how spatial considerations affect diversity and program localization. Additionally, I investigate the utility of spatial proximity in the evolution of complex loop structures.

SGP is applied to a spectrum of problem types, and its performance is benchmarked against two prevalent GP methodologies. This comparative analysis not only underscores the viability of SGP but also opens research pathways for further studying the integration

of spatial dynamics within EAs.

Chapter 2

A Framework for Studying the Impact of Space in GP

Parts of this chapter are reproduced with permission from Springer Nature from [Miralavy and Banzhaf, 2023a].

2.1 Introduction

Morphology, shape, mass, and volume of an organism are all intrinsically linked to the three-dimensional space they occupy, making spatial considerations a cornerstone of natural evolution. The proximity of spatial entities shapes the habitats of organisms, highly influencing their evolutionary paths. For instance, sloths have adapted to conserve energy through slow metabolism to thrive in nutrient-scarce environments [Cliffe et al., 2018], while snow leopards have developed long claws and thick fur to navigate and survive in harsh, icy terrains [Fox et al., 2024]. Drawing inspiration from these natural phenomena, computational problem solvers often overlook the dimension of space in evolutionary algorithms. This abstraction simplifies the algorithms, potentially easing implementation and reducing computational demands. However, such simplification comes at the cost of removing the advantages that spatial considerations can offer. Historically, limited computational power necessitated

abstractions to develop complex problem-solving algorithms. Yet, with technological advancements, this constraint is becoming increasingly obsolete, allowing us to reconsider previously unexplored domains of bio-inspired evolutionary algorithms without traditional limitations. Moreover, space as a dimension can facilitate parallelization, potentially accelerating the computational processes within evolutionary systems. Complexity in algorithm design often leads to models that are challenging to interpret. However, one of the defining features of spatially-oriented computational problem solvers is their inherent capacity for organization, management, and visualization of entities. Spatial analysis, a technique widely employed across numerous scientific disciplines through imaging, graphs, maps, and heatmaps, offers a natural and intuitive means to understand and interpret complex data [Fotheringham and Brunsdon, 1999, Dell'Ovo et al., 2018, Dale and Fortin, 2014].

The impact of space in natural evolution is not yet fully understood. Some researchers focus on analyzing the effect of spatial elements in natural processes. For example, Allen et al. [Allen et al., 2015] argue that spatial properties impact the molecular clock. Using mathematical models they show that asymmetries in space can lead to increase or decrease of the number of mutations. [Hickinbotham et al., 2021] perform a series of comprehensive experiments to show how spatial patterning can prevent the extinction of parasitism. Hancock et al. [Hancock et al., 2022] argue that the common methods used for phylogenetic inference such as multispecies coalescent are spatially-independent. In their model, they include a z-axis which proves useful in identifying modes of speciation and characterization of demographic factors that impact the phylogenetic trees.

Space is a basic principle in some areas of evolutionary computation, particularly in fields such as Artificial Life, where digital organisms heavily rely on spatial properties [Gershenson et al., 2020, Chan, 2018, Kondo and Miura, 2010]. A notable example is Avida, introduced by Ofria et al. [Ofria and Wilke, 2004], which serves as a framework for studying digital evolution through self-replicating digital organisms. These organisms undergo mutation, reproduction, and natural selection, enabling researchers to analyze and observe their evolutionary processes

using the Avida framework. Notably, this system allows digital organisms to possess spatial properties, which are instrumental in determining interactions among them. Spatial Evolutionary Game Theory [Killingback and Doebeli, 1998, Roca et al., 2009, Cui et al., 2015] has also garnered significant interest among researchers. Killingback et al. explored this domain in their work [Killingback and Doebeli, 1996], comparing spatial models with traditional models of evolutionary game theory. They spatially represented the Hawk-Dove game, where different sites evolve distinct strategies, and each site's strategy changes if a neighboring site adopts a more favorable approach. Their findings unveiled differences between the two systems, including statistical variations in the number of hawks compared to the traditional model. Furthermore, they observed that spatial properties facilitate the evolution of specific types of strategies.

A common approach for modeling systems that require spatial elements involves the use of Cellular Automata (CA) [Chopard and Droz, 2005]. CAs are simple discrete models consisting of a grid of cells, each capable of having multiple states. A set of rules governs changes to the cell states at each time step of the system influenced by a properly defined neighborhood of cells. For instance, Vayadande et al. [Vayadande et al., 2022] utilized a CA to simulate Conway's Game of Life, observing the emergence of natural groups. Such simulations provide valuable insights for researchers aiming to understand and predict phenomena in their natural counterparts. Giabbanelli et al. [Giabbanelli et al., 2019] proposed several approaches for modeling the cell-to-cell and cell-free spread of the HIV-1 virus within the human body using CA. Their goal was to strike a balance between realism and model validity. In a more computational context, Enescu et al. [Enescu et al., 2019] asserted that CAs are spatially-aware systems that can be evolved to account for spatial elements in image processing tasks. They employed an evolutionary algorithm and defined evolutionary operators to evolve CAs capable of detecting edges in images.

Spatially structured [Tomassini, 2005] and cellular [Tomassini, 2010] evolutionary algorithms are examples where interactions among individuals in a population are influenced

by their spatial elements. Dittrich et al. [Dittrich and Elmenreich, 2015] conducted a study comparing the evolution of a panmictic population of artificial neural network controllers with a spatially-structured population to solve an XOR task and a complex agent self-organization task. Their results indicated that the spatially structured population outperformed the panmictic one. Fernandes et al. [Fernandes et al., 2018] argue that structured evolutionary algorithms can't fully reach their potential unless there is flexibility in population size. They demonstrated that, for a fixed-size population, structures represented by regular graphs with lower degrees consistently converge to global optima. However, when the population size is appropriately set, convergence occurs faster and with a higher probability for graphs with higher degrees. Kuo et al. [Kuo et al., 2021] introduced the idea that populations represented by more complex graphs may be more effective. They demonstrated that the distribution of network degrees and the organization of nodes play a crucial role in shaping the dynamics of new mutations within the system. [Dick and Whigham, 2013] employs a spatially structured population to control bloat in a GP system. This approach, known as spatial structure with lexicographic parsimonious elitism (SS+LPE), integrates a lexicographic parsimony scheme during replacement to effectively manage program size and improve fitness.

Understanding the structural properties of the target problem is crucial for applying problem-solving techniques to real-world physical challenges. For instance, Richards and Amos [Richards and Amos, 2016] incorporated regulatory representations for architectural building design. To tackle this problem effectively, the regulatory representation must account for spatial constraints essential for evolving feasible structures. One intriguing aspect of combining evolutionary computation with spatial considerations is the emergence of novel spatial patterns, which adds to its appeal. Kicinger et al. [Kicinger et al., 2005] conducted an extensive review of state-of-the-art algorithms that integrate evolutionary computation into structural design. Circuit design is another domain where evolutionary computation has found application [Vasicek and Sekanina, 2014, Miller et al., 2000]. In such cases, a significant challenge lies in minimizing space usage while adhering to constraints that necessitate specific

circuit elements to be located closer to one another. Lastly, most of the swarm algorithms such as ant colony [Dorigo et al., 2006] or particle swarm optimization [Kennedy and Eberhart, 1995] are inspirations from natural systems that often rely on spatial organization of agents. Importance of these spatial organizations becomes more announced in swarm robotics [Schranz et al., 2020].

The rest of this chapter introduces SGP as a tool for studying space in GP and discusses in details how SGP algorithm performs.

2.2 Spatial Genetic Programming

We begin our study on the impact of space as a first-order effect in evolving GP models by introducing the Spatial Genetic Programming (SGP) computational framework. SGP, is a GP system in which models are represented by a number of LGP program nodes spread in a 2D space that execute in an order determined by their spatial position. SGP operates in two modes: the programmatical mode, which is useful in solving decision-making problems, and the spatial mode. In the current chapter, we focus on the spatial mode of the system. The programmatical mode will be explained in detail in chapter 5.

Figure 2.1 shows the schematic of an example SGP individual. This individual consists of eight LGP programs which include statements capable of tweaking the internal system registers of the model. Programs represented by circle nodes, are non-terminal nodes and the program represented by a square, is a terminal node (P7). Only four of these eight programs execute while the rest of the programs, shown with faded orange color, do not execute and therefore have no impact on the output of the model. The order of execution of these programs, depicted with green lines, is based on attempts to minimize the traverse cost. In the spatial mode of the system, this cost only considers spatial proximity. In the rest of this section, first an overview of the LGP paradigm incorporated in our study is given and then algorithms for determining the order of the execution of the LGP models are explained.

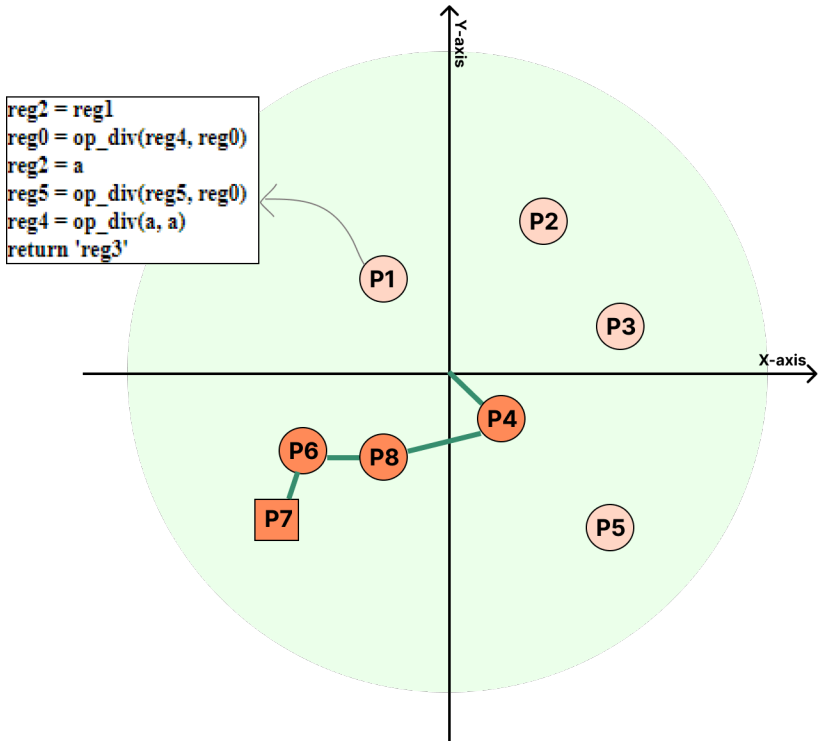


Figure 2.1: Schematic of a SGP model and its order of execution. The execution starts from the (0, 0) coordinate and stops with the execution of P7 as its terminal node

2.3 LGP programs

SGP nodes can be Neural Networks, trees of TGP or any other type of problem solver that follows an input/output approach. In this study, programs within SGP comply with the guidelines of a basic LGP system, which encompasses fundamental mathematical operations, rudimentary branching options, and function calls. These LGP programs are devoid of loops, allowing only basic *if* statements, each carrying a singular instruction. Every program statement is essentially a call to an operator function that might require zero or more operands for execution. These operators are picked from the GP function set during model evolution.

The primary role of program statements is to adjust the internal state of SGP, principally represented by shared memory registers. A number of internal registers can be marked to act as output registers. Nevertheless, statements might also include calls to atomic functions or interactions with an external environment. For example, basic mathematical operations

like -, +, /, and × demand two numerical operands, which could be constants, inputs, or values residing in internal registers. The output of these operators can only be stored in an internal or an output register, subsequently modifying the system's inherent state. In contrast, invoking a function like 'print' does not alter the system's inner state as it remains agnostic to register value adjustments.

There are a few additional details about the LGP programs to note. Every LGP program concludes with a *return* statement responsible for returning a single numerical value, R, to the SGP system. This value could be a register, a constant, or an input value and may be used later as a factor in shaping the cost function. When R is utilized in the cost function, the order of executed programs in an individual becomes dependent on the problem's input values. Further examples on cost functions that incorporate R as a factor can be found in chapter 5. Additionally, output registers are exempt from being used as operands for operators, and their values can only be modified by the return outcomes of other operators. Finally, input registers, which constitute the problem inputs, remain immutable to any statement.

2.4 Spatial Representation and Execution of Individuals

SGP individuals are represented by LGP programs that are distributed in a continuous 2D space and execute sequentially in an order determined by their spatial positions. The execution of a LGP program can modify the internal state of the system by changing the values of internal registers. During the process of evolution, not only do the instructions within these LGP programs evolve, but their spatial positions also can change in order to discover an optimal execution order for these programs. This flexibility allows SGP individuals to possess unrestricted topologies and to form intricate control structures.

In an imperative SGP model, program nodes consist of non-terminal nodes and a single terminal node which, when executed, terminates the execution of an individual model. The order of the execution of programs in SGP follows a simple distance-based rule: Starting

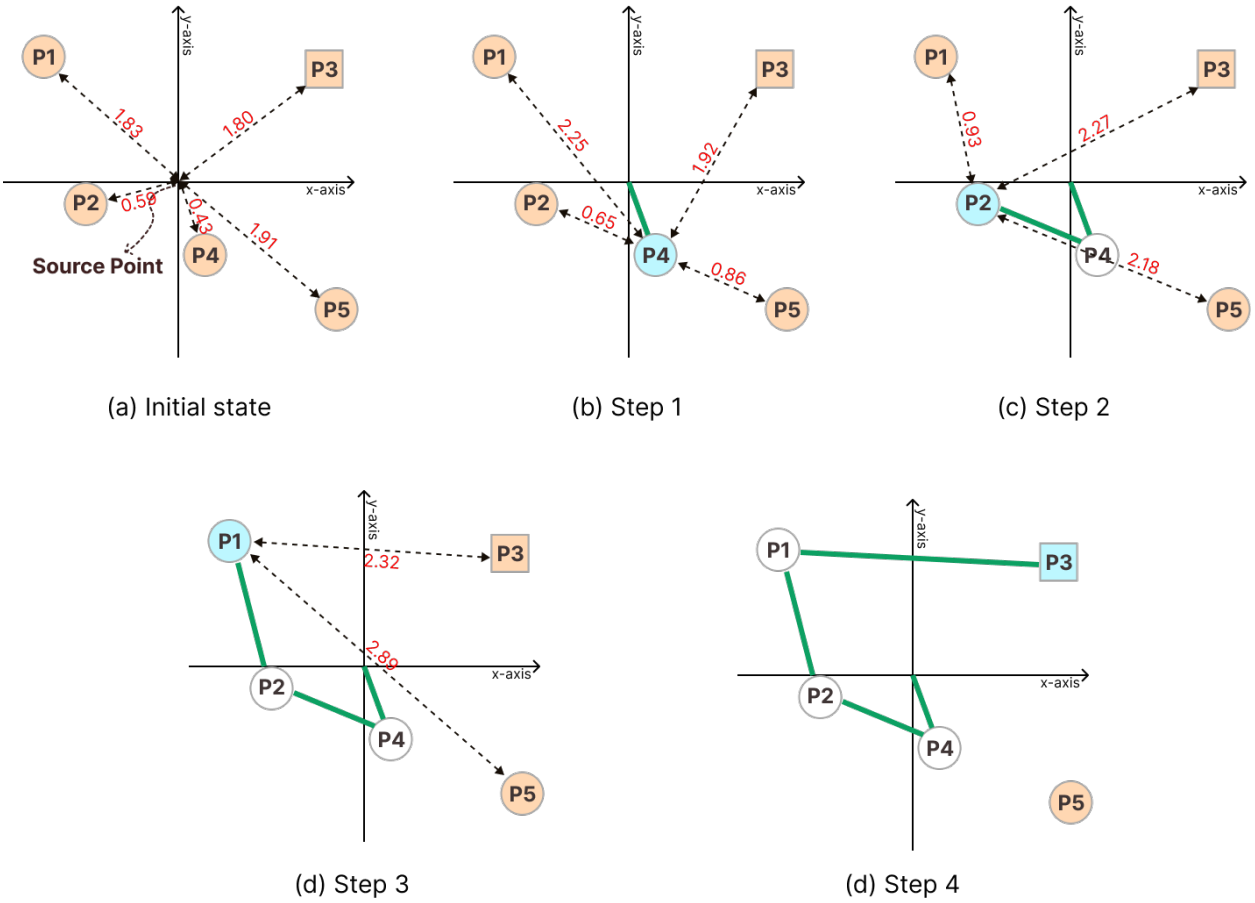


Figure 2.2: In the spatial mode of SGP, the order of execution for LGP programs is determined based on the provided diagram. In the diagram, circular nodes represent non-terminal nodes, while square nodes represent terminal nodes. Each node corresponds to a LGP program. There are a total of five programs, consisting of four non-terminal programs and one terminal program node (P3). The green lines in the diagram indicate the sequence of executed programs. The light blue node in the diagram represents the program currently selected for execution, while white nodes depict programs that have been previously executed

from (x, y) = (0, 0) as the source point, the program which has the cheapest traverse cost is selected for execution and the source point updates to be the position of currently executed program. In case the chosen cost function is only compromised of a distance[1] method, the cheapest traverse cost program node is the same as the closest program to the source point. This process continues until there are no more program node or until a terminal program node is executed.

Figure 2.2 shows a SGP individual where loops are not allowed (a node only executes once)

[1]Distance method, returns the Euclidean distance between a source and a target point.

Algorithm 1: Algorithm of how SGP handles selecting the next program for individual execution

Data: $P_{current}$, $Programs$
Result: P_{next}
$P_{next} \leftarrow None$;
for P_i *in* $Programs$ **do**
    // Loop through all programs
    $cost_i \leftarrow D(loc_{source}, loc_{P_i})$;
    if *loops are not permitted and* P_i *is already visited* **then**
        continue;
        // Don't allow revisit if loops are set to off
    end
    if $P_{next} == None$ **then**
        $P_{next} \leftarrow P_i$;
        // Select the first program, if there is no candidate $cost_{next} \leftarrow cost_i$;
    end
    if $cost_i < cost_{next}$ **then**
        $P_{next} \leftarrow P_i$;
        // Replace the candidate $cost_{next} \leftarrow cost_i$;
    end
end
return P_{next};

and how the order of execution of the LGP programs is determined for that case. Initially, starting from the source point in Figure 2.2a, the Euclidean distance to every other program node is calculated and normalized as the traverse cost. The program with the lowest cost, in this case P4, is then selected to be executed, and the source point is updated to the location of P4. In step 1, the same approach of cost calculation takes place to find the cheapest cost program with regards to P4, and P2 is selected for execution. The same process continues and P1 and P3 execute in order. P3 is a terminal node and therefore concludes the execution of the individual model causing P5 not to be executed. P5 can be considered similar to introns in biological genes which are not expressed.

The cost function for calculating the traverse cost is as follows:

$$cost_i = \ln\left(1 + D(loc_{source}, loc_i)\right)$$

Here, $cost_i$ represents the cost of traversing from the source point to program i. The function $D(source, target)$ calculates the Euclidean distance between two points in a 2D space, with loc_{source} and loc_i representing the 2D coordinates of the source point and program $i's$ location, respectively. The natural logarithm is employed to normalize the distance value. The cost function is configurable and while normalization may not be necessary for a simple spatial cost function like the one described, it proves beneficial when the cost function is combined with additional elements to create more sophisticated equations. For a more detailed understanding of how the next program is selected for execution in SGP, Algorithm 1 provides more details.

Algorithm 2: Algorithm for execution of SGP individual models

Data: $model$, $registers$, $inputs$
Result: $output, memory$
$memory \leftarrow registers + inputs$;
// Initialize the system parameters
$P_{current} \leftarrow None$;
$outputs \leftarrow None$;
reset($model$);
$time_{start} = time_{now}$;
while *True* **do**
  $P_{current} \leftarrow select\_program(P_{current}, model.programs)$;
  if $P_{current} == None$ **then**
    break;
    // No more program candidates left for selection
  end
  $output, memory \leftarrow execute(P_{current}, memory)$;
  if $is\_terminal(P_{current})$ **then**
    break;
    // Terminal program reached
  end
  $time_{current} = time_{now}$;
  if $time_{current} - time_{start} > T$ **then**
    break;
    // Time limit reached
  end
end
return $output, memory$;

If revisiting a node is permitted, the program execution order can potentially lead to the emergence of loops. To prevent infinite loops, a modification has to be made to the cost

function as follows:

$$cost_{final} = cost_i + (cost_i + 1) \times v \times \mu$$

Here, $cost_{final}$ represents the traverse cost to P_i after applying the re-visit penalty. In this equation, v denotes the number of previous visits to P_i, and μ is an arbitrary constant representing the re-visit penalty coefficient. Larger values of μ increase the difficulty for the system to evolve loop structures.

Algorithm 2 outlines the execution process of a SGP individual. The algorithm initiates by initializing all the necessary variables. The function $reset(model)$ is responsible for clearing any information stored in a model from prior executions, such as tracking the number of times each program has been visited. The variable $time_{start}$ serves as a timestamp marking the beginning of the individual's execution. This timestamp plays a role in a mechanism that prevents individuals from stalling execution for more than the specified time threshold, denoted as T. The function $select\_program(P_{current}, model.programs)$ determines the next program to be executed, following the principles described in Algorithm 1. If $P_{current}$ holds a *None* value, it signifies that there are no suitable candidate programs for execution, and consequently, the individual's execution loop is terminated. Subsequently, the selected program is executed, producing return values that include a single numerical value and memory representing the internal state of the system. If $P_{current}$ is a terminal node, this marks the termination of the individual's execution. The final termination condition arises when more than T seconds have elapsed since the start of the individual's execution. This time limit primarily serves to handle situations involving infinite loops.

2.5 Evolution of Models

The evolution of models in SGP commences with the random initialization of a population of individuals. During the initialization phase, a random number of LGP programs with randomly generated instructions are created. SGP employs a panmictic population structure

as opposed to the internal spatial structure of each individual which means that the population lacks spatial organization. In such population, every individual has an equal chance of producing offspring for the next generation through tournament selection. Each individual is then evaluated based on a fitness function tailored for solving a specific computational problem. The fitness function assesses how effectively the model has performed, assigning a fitness value to each individual. In the selection phase, a fixed number of individuals are randomly chosen, and the two best individuals with the highest fitness values are selected as parents to produce two offspring for the next generation. If crossover is permitted, the two parent individuals recombine to create two new offspring. These two offspring undergo mutation, which introduces changes to the LGP statements of the model programs or their spatial positions. In case there is no crossover, the two parent individuals only mutate to produce two new offspring. This process continues until a new population of models, equal in size to the previous generation, is generated. Throughout this evolutionary process, the best individuals of each generation, referred to as elites, are preserved. The termination condition for SGP is determined by a predefined total number of generations set before running an experiment. For most of the experiments conducted for this dissertation, no crossover algorithm is used; however, forms of spatial crossover is investigated and explained in the next chapters.

2.6 Mutation Operators

In SGP, mutations can either influence the spatial attributes of program nodes or happen at the LGP level. Various mutation operators are designated for each mutation category. The application of the mutational operators to each program or statement is elaborated upon in this section.

2.6.1 LGP Mutations

There are five mutation operators that are used to introduce diversity in the LGP programs:

1. **Add statement:** There's a m_{LGP} percent chance for each program of an individual to gain a new statement. Individuals cannot exceed a predefined maximum number of statements.
2. **Remove statement:** There's a m_{LGP} percent chance for each program of an individual to lose a statement. An individual must maintain at least one statement, as every program requires a return statement.
3. **Mutate return value:** There's a m_{LGP} percent chance for each program to have its return value (last statement) altered.
4. **Mutate operand:** Each statement has a m_{LGP} percent chance of having one of its operands modified.
5. **Mutate statement output:** Each statement has a m_{LGP} percent chance for its return value to be stored in a different register.

2.6.2 Spatial Mutations

There are three mutation operators that are used to introduce diversity to the spatial properties of the programs:

1. **Add program:** There's an $m_{spatial}$ percent chance for an individual model to gain a program. The total number of programs cannot exceed a predefined maximum size.
2. **Remove program:** There's an $m_{spatial}$ percent chance for a program to be removed from an individual model. An individual must maintain at least one program.
3. **Position mutation:** Each program has an $m_{spatial}$ percent chance of being assigned a new random spatial position in the entire 2D space.

2.7 Introduction of New Operators

A pivotal aspect of any GP system involves the ability to introduce new operators tailored for solving specific target problems. For instance, mathematical operators prove effective

for solving many regression problems, while addressing image processing tasks necessitates operators capable of manipulating or extracting features from images. In the case of SGP, it comes equipped with a default set of operators, including basic mathematical operations. To facilitate the incorporation of new operators, SGP provides an abstract operator class alongside its default operators, offering guidelines for implementing custom ones. The default SGP operators encompass fundamental mathematical operations such as addition, subtraction, multiplication, protected division, square root, logarithm, exponential, sin, cos, as well as an assignment operator and a basic conditional *if* statement operator. The *if* operator can skip the execution of the next instruction if a specified condition is not met, but it cannot prevent the execution of the return statement. Typically, an SGP operator necessitates manual definition of internal functions that specify the number and types of inputs and outputs required by the operator. It also mandates an evaluation function that processes the inputs and generates the expected outputs. Additionally, the operator provides an annotation indicating how it can be utilized within a programming language. In the context of this work, the Python programming language [Python, 2021] is selected as the target due to its simplicity. This approach allows for the seamless integration of new operators into the SGP framework, enhancing its adaptability and problem-solving capabilities.

The process of introducing new operators in the SGP system requires manual implementation of a dedicated class for each operator. While this approach may appear more involved compared to certain other GP frameworks, it offers two significant advantages:

1. Modular Expansion: By adding operator classes in a modular fashion, users gain the flexibility to introduce more complex operators into the system. This opens the door to incorporating sophisticated operators, including machine learning algorithms. Such flexibility can create a hierarchy of problem-solving capabilities within the SGP framework.

2. Support for New Data Types: The approach also enables the introduction of new data types to the system. For instance, an operator can specify an arbitrary input type,

such as *XYZ*. SGP will subsequently search for input values, registers, or constants with the same data type and align the operators with the operands accordingly. This capability enhances the versatility of SGP by accommodating a broader range of data types and problem-solving scenarios.

2.8 The Translator Component

Another contribution of this work is a translator component that produces Python scripts equivalent to the evolved SGP models. These scripts can be quickly evaluated by simply executing them with the Python compiler. They are written in style easy enough to explore and understand, even though they are generated from complex SGP models. Relying on these scripts also increase the transparency of the SGP models, allowing domain experts to understand and modify the models if necessary. Selecting Python as programming language for the produced scripts was an arbitrary choice due to its simplicity and closeness to human language. The principles used in the design of the SGP translator could be utilized for any other similar programming language. Figure 2.3 illustrates different sections of these Python scripts and parts of corresponding codes taken from an evolved SGP model. When executed, the script starts by importing required packages for running and a debug flag which, if set to true, shows the order of execution of the SGP programs (For example, P0, P1, P2, P1, P2, end). Next, the script attempts to retrieve problem inputs by requesting the user to enter appropriate values. This part of the script can be easily modified to feed the script with arbitrary inputs using other approaches. For example, directly connecting the SGP script to external software. Then, the register values are defined and initially set to zero. All the operators in the SGP operator set are, in fact, human-written code snippets that perform a custom action (see *Introduction of New Operators*). Declaration of these operator methods is added to the generated script for SGP programs to call during the execution process. LGP program classes are blueprints of the evolved LGP programs that are translated into Python codes. These classes have attributes determining properties of the evolved program, such

as length, type, return parameter, coordination, and the number of times this program has previously been executed (an essential attribute for the interpreter to avoid infinite loops). Next are the model attributes, which define the parameters necessary for the SGP interpreter to execute the model correctly. Furthermore, a critical portion of the SGP interpreter is included in the script, which serves as a glue that connects everything and executes the evolved programs in the proper order. Finally, the model output as described is printed on the output screen; however, much like the situation with inputs, this part can be easily modified in other desirable ways.

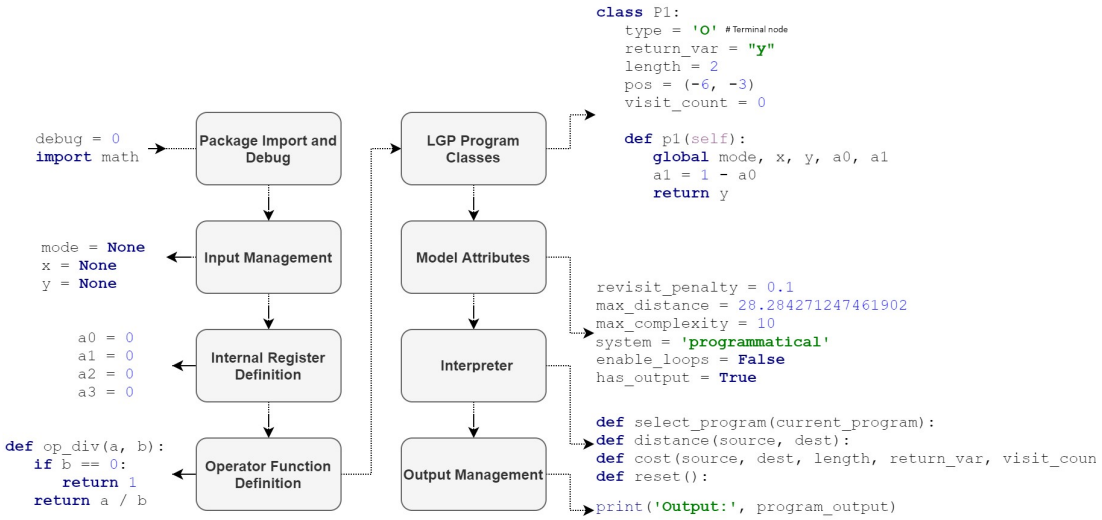


Figure 2.3: An overview of the different sections of a produced Python scripts from a SGP model

2.9 Configuring the SGP system

Setting up the SGP system is achievable through modifying a single configuration file. In this section, every hyper-parameter of the system that is modifiable is explained in details and a brief introduction of the technical implementation of the system is given. Keep in mind that not all of the parameters described here were part of the experiments conducted for this dissertation. However, the SGP tool comes with these capabilities. The title of the following parameters is as appears in the configuration file of SGP to make the following descriptions

more practical:

- **fitness:** Points to a fitness file located in the *Fitness* directory of the SGP environments. This file is in charge of evaluating SGP models based on a specific problem. For example, *fitness = Loop.A5B5* indicates that the SGP is incorporated to solve the *A5B5* problem which resides in the *Fitness/Loop/* directory of the SGP environment.
- **seed:** Determines a random seed for the experiment. This value is extremely helpful for identical replication of the experiment.
- **generations:** Number of evolutionary generations set to be the termination condition of the system.
- **population_size:** Determines the size of the population.
- **tournament_size:** Size of the tournament pool used during selection.
- **structural_mutation_rate:** Chance of performing a structural mutation. This parameter takes a value in the range of $[0, 1]$ representing the percentage chance for mutation. For example, 0.2 indicates a 20% chance of mutation.
- **LGP_mutation_rate:** Chance of performing a LGP mutation. Range: $[0, 1]$
- **crossover_rate:** Chance of performing a crossover between the two parents chosen by the tournament selection. Range: $[0, 1]$
- **elitism:** Number of elite individuals to be taken to the next generation without modification.
- **cost_formula:** The cost function used for determining the order of execution of the programs in each individual. Any combination of the following variables and mathematical operators can be utilized to form the cost equation: *distance, max_distance, length, max_length and return_val*. *distance* is the distance between the source point and the target program. *max_distance* is the maximum distance possible between two programs. *length* is the number of statements in the target LGP program. *max_length*

is the maximum allowed number of statements for a LGP program. *return_val* or the R value is the numerical value that every LGP program returns as their final statement.

- **topology:** Sets specific constraints on the position of the LGP programs. Possible values are: circle, ring, line, lattice and 3D
- **output_ratio:** This parameter can take values of *single* indicating that the SGP models will only be consisted of a single terminal node, *none* indicating that the SGP model will not have a terminal node and every program should execute in order for the model execution to terminate and a numerical value in the range of [0, 1] with indicates a chance that every individual might be a terminal node. Numerical values can cause a SGP system to have multiple terminal nodes.
- **evaluation_count:** Sets the number of times a fitness function should evaluate an individual model to return a fitness value.
- **conditionals:** A list of conditional operators that can be given to the GP system to be used in LGP statements.
- **constants:** A set of constant values given to the GP system to be used in LGP statements
- **operators:** A set of operators that is given to the GP system as its function set.
- **registers:** The number of internal system registers for chosen for the experiment.
- **init_size_min:** Minimum number of programs when randomly initializing the individuals.
- **init_size_max:** Maximum number of programs when randomly initializing the individuals.
- **size_max:** Maximum number of allowed programs in the individuals.
- **enable_loops:** Allows or disallows the formation of loop structures. Can be True or False.

- **self_loop:** Whether or not self-loop of programs is allowed. Can be True or False.
- **revisit_penalty:** This is a coefficient that increases the cost of travelling to an already visited node in SGP. The default value is set to be 0.01. Higher values makes it harder for the system to form iterative structures while lower values works conversely.
- **init_lgp_size_min:** Minimum number of statements when randomly initializing LGP programs.
- **init_lgp_size_max:** Maximum number of statements when randomly initializing LGP programs.
- **lgp_size_max:** Minimum number of statements allowed in programs.
- **init_radius:** Determines the size of the 2D space of the individuals.
- **individual:** Points to the implementation file for the individual module of the system. This or similar modules can be replaced with other implementations this way.
- **population:** Points to the implementation file for the population module of the system.
- **evolver:** Points to the implementation file for the evolver module of the system.
- **program:** Points to the implementation file for the program module of the system. For example, the default is set to be LGP for SGP's programs to be of that type.
- **evo_file:** Path to the evolutionary log of the experiment to be saved.
- **pop_save_path:** Path to where the population file containing information for all of the individuals to be saved.

Once the SGP system is properly configured with respect to a target problem, it can be used to evolve solutions for the target problems.

2.10 SGP Benchmarks

Along with the SGP system comes a series of computational tasks that can ease the evaluation of developing spatial GP algorithms. All of these computational tasks are classified into

different modules, the number and the data type of each input and output is explicitly specified and a suitable fitness function is defined for the optimizers to tackle them. Most of these problems, such as the Symbolic Regression (SR) problems are defined and explained in details throughout this dissertation. However, a list of all of these problems along with a short explanation is provided here:

1. Feynman Symbolic Regression Problems: This is a SR benchmark consisted of 92 trivial to non-trivial problems taken from Feynman Lectures in Physics [Feynman et al., 2011]. Example data points for evaluation of models are provided along with the benchmark.
2. Nicolau Regression Problems: This benchmark taken from [Nicolau et al., 2015] is consisted of 21 non-trivial SR problems.
3. Loop Problems: This is a small collection of simple high-degree SR problems.
4. Classic Control Problems: This is a benchmark of four control problems taken from Open AI's Gym Library [Brockman et al., 2016] and modified to be evaluated in the SGP environment.
5. Toy Problems: This is a collection of three custom decision-making toy problems that are the tackled by SGP in chapter 5.
6. Artificial Ant Variation: This is a variation of the classic Santa Fe Ant problem. The difference between this implementation and the conventional ones is that the problem-solver is in charge of evolving iterative structures to execute an evolved routine.
7. TicTacToe Problem: An implementation of the famous TicTacToe game.

2.11 Summary

In this chapter previous literature on how space can be impactful in computational frameworks was explored and the importance of studying space in genetic programming was briefly discussed. Next, the algorithms for Spatial Genetic Programming were introduced and different parts of the system such as mutational operators, underlying LGP programs, and

the evolutionary process were explained in details. It was discussed how it is possible to add new operators to SGP for solving target problems and how the translator component of SGP is capable for producing executable Python scripts from evolved SGP models to further increase the transparency of the system. Furthermore, different configuration parameters of the SGP system were discussed in details and the benchmark problems included in the SGP tools were named. In the next chapters, experiments using different modes of the SGP system for solving various classes of problems is discussed and the impact of space in SGP individuals is analyzed.

Chapter 3

Spatial Analysis of SGP Individuals

Parts of this chapter is adopted from [Miralavy and Banzhaf, 2023a].

3.1 Introduction

In this chapter, we aim to increase our understanding of the characteristics of SGP caused by integrating space as a fundamental concept within its individuals. We explore how the addition of spatial dimensions can impact traditional frameworks of genetic programming. To this end, a spatial analysis of SGP is conducted, involving various experiments to study diversity, localization, and the structure of SGP individuals. The structural characteristics of the SGP system are examined from two perspectives: how imposing spatial constraints affects system performance, and how SGP individuals inherently self-organize over the course of evolution

Spatial proximity, a fundamental concept in computational models, offers unique features that are particularly beneficial in EAs and GP. This chapter studies the utilization of spatial proximity, defined by Euclidean distance, within the context of SGP's individuals. While the notion of spatial proximity in SGP is novel, somewhat similar concepts have been previously explored in the literature. For instance, in tree-based structures, the traversal distance between nodes can be considered a form of proximity. Similarly, in grammatical

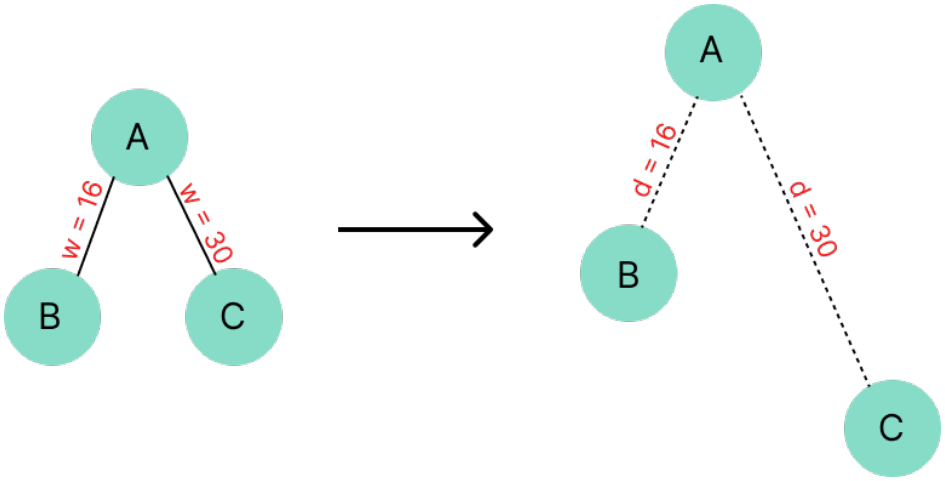


Figure 3.1: A Non-spatial tree structure and it's spatial equivalent

representations or the statements in LGP systems, the sequential arrangement of instructions captures the essence of entities being close in a two-dimensional (2D) space. Instructions that are sequentially adjacent resemble entities that are near each other in 2D space, whereas those that are further apart in the sequence are akin to entities that are more distant in 2D space.

To illustrate, consider the weighted graph shown in Figure 3.1. On the left, a simple graph with three nodes is presented, where the weight of the edge between nodes A and B is less than that of the edge between nodes A and C. On the right, an equivalent representation is depicted, with weights translated into Euclidean distances within a 2D space.

In the previous chapters the importance of exploring the dimension of space in computational algorithms was discussed. Towards this mean, SGP was introduced as a framework which facilitates research of understanding the impact of space in GP. The focus of this chapter is twofold: To use SGP to evolve complex iterative structures and to perform a spatial analysis on the impact of space on the SGP models.

Index	Equation	SGP	LGP
93	$f = a^{20}$	Yes	Yes
94	$f = a^{40}$	Yes	Yes
95	$f = a^5 + b^5$		
96	$f = (a + b)^5$	Yes	Yes

Table 3.1: High-degree equations tackled by SGP and LGP. These equations were not part of the Feynman equation set

3.2 Results: Evolving Abstract Loop Structures

Incorporating spatial elements into the system introduces the capability of using distance as a mechanism to control the number of iterations in a loop between two program nodes in a way that the closer the nodes are, the higher the chance that a loop between the two occurs. This unique characteristic of SGP paves the way for generating abstract models particularly suited for tasks demanding iteration.

Figure 3.2 showcases the evolutionary progression of the best individuals evolved by SGP and LGP, aiming to tackle four elementary high-degree symbolic regression challenges as detailed in Table 3.1. The index column, specifies the a unique number associated with the equations as a naming convention. Configurations of the SGP system is experimentally chosen and is as disclosed in Table 3.2. The data used for evolving models in this experiment, as well as for the subsequent regression problems, are adopted from the example files associated with [Udrescu et al., 2020]. Unless otherwise stated, a simple correlation-based fitness metric is employed to evaluate the performance of models in the symbolic regression experiments. This correlation fitness metric is explained in detail in Section 4.4, with the objective being to minimize the fitness equation.

A spatial mutation rate of 40% was adopted for these experiments, and the function set for both algorithms was limited to the four fundamental mathematical operators. To make

Parameter	Description	Value
g	Number of evolutionary generations	100
$Size_{pop}$	Population size of the experiment	300
$Size_{tournament}$	Size of the tournament selection pool	10
e	Number of elites that are directly selected to be a part of the next generation	3
$m_{spatial}$	Spatial mutation rate	20%
m_{LGP}	LGP mutation rate	20%
C	Crossover rate	0%
$cost_i$	Cost function	$\ln(1 + distance)$
$Topology$	Describes any topological limitation for placing nodes on the 2D space	circle
$Type_{output}$	Describes the output type of the system	single
$Count_{evaluation}$	Number of points used for evaluation of an individual model	30
$Count_{registers}$	Number of internal system registers	input count + 2
$initSize_{max}$	Minimum number of programs per individual during initialization	10
$Size_{max}$	Maximum number of programs per individual	15
$lgpInitSize_{max}$	Maximum number of statements per program during initialization	10
$lgpSize_{max}$	Maximum number of statements per program	15
T	Maximum individual evaluation time	100ms
$radius$	Size of the topology representing the 2D space	150
Set_{OP}	Operator or function set	$+, -, \times, /$
$Constants$	Set of constant values	-1, 1, 2, 3, 5
$reps$	Number of replicates for each experiment	50

Table 3.2: Parameter setup of SGP. Relevant parameters for LGP unless specified are chosen similarly

the comparison fair, for the LGP system maximum number of statements is set to be 225 to compensate with the maximum of 15 programs with 15 instructions in SGP. Both LGP and SGP demonstrate proficiency in solving $f = a^{20}$, $f = a^{40}$ and $f = (a+b)^5$. However, they fail to solve the equation $f = a^5 + b^5$ within 100 generations. The two solved problems by SGP seem to be trivial for this system, as it consistently derives solutions in under 5 generations. In contrast, while LGP can also decipher these equations, a recognizable pattern emerges: as the number of iterations grows, LGP demands an extended period to identify an optimal solution for such problems. $f = (a+b)^5$ and $f = a^5+b^5$ are two equations of the same degree.

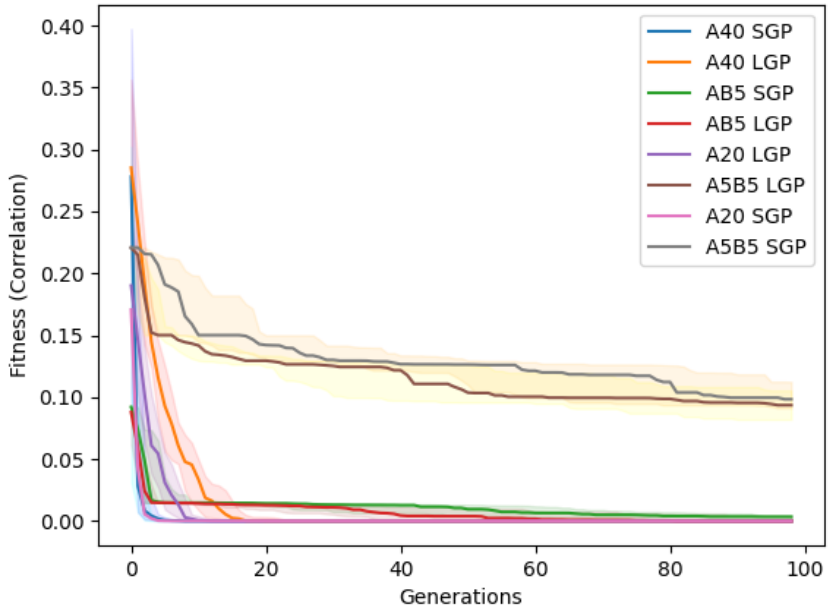


Figure 3.2: Evolution of programs for solving high-degree equations. Lines are median values while the shaded areas represent 75th percentiles. AB5 denotes the $f = (a + b)^5$ equation while A5B5 denotes the $f = a^5 + b^5$ equation

Even though the expanded form of equation 96 is more complicated than equation 95, it can be simplified into a single high-degree term. This comparison leads to the conclusion that both LGP and SGP, in their current configurations, are not proficient at solving symbolic regression equations comprised of two or more high-degree terms that cannot be simplified into a single high-degree term. However, this presents a research opportunity to investigate evaluation metrics that encourage these systems to solve such problems.

Figure 3.3 illustrates an optimal solution that SGP develops to address the equation a^{20}. Arrows indicate the order in which SGP orchestrates its model programs, with a deeper hue on the arrows with multiple traversals of the same execution path. In the solution exemplified, program P4 is the initial program executed. Upon its termination, the model pivots to P1, drawn by its spatial closeness to P4. Subsequent to P4, even factoring in the revisit penalty of μ, P1 retains the most minimal traversal cost. These two programs, P1 and P4, create a loop that cycles 21 times (both P1 and P4 are executed 21 times each), halting only when

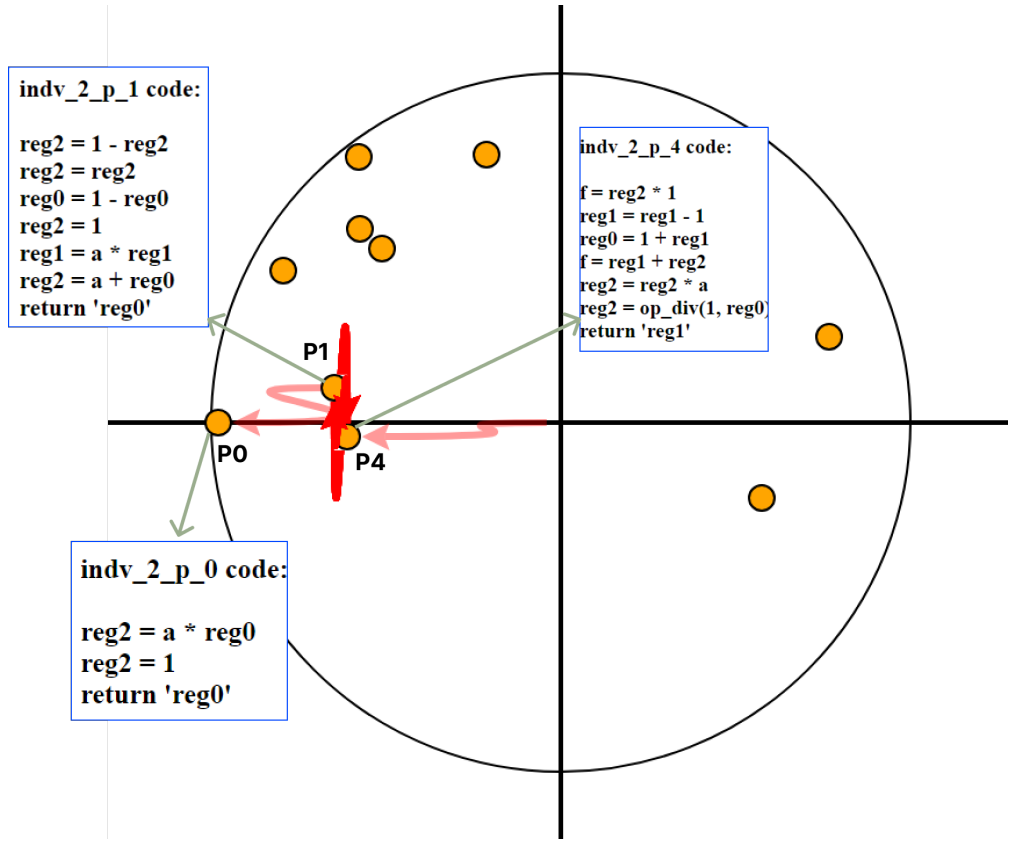


Figure 3.3: An optimal solution comprising 9 programs, of which only 3 are executed. Other than P0, P1 and P4, the rest of the orange circles represent programs that are never executed and serve as introns for the individual. This visualization is generated automatically by the SGP's analysis tool. Each executed node is associated with its corresponding LGP program. The title of each program designates the individual under analysis, accompanied by a program index for easier identification

transitioning to P0 (a terminal node) becomes a more economical choice than preserving the loop between P1 and P4 by having a lower traverse cost.

This particular solution encompasses 17 statements dispersed across the three programs. A manual simplification of the programs, which involves removing only obvious redundant statements, reveals that this model effectively operates on 11 statements. In contrast, an equivalent solution generated by LGP entails over 20 effective statements. Notably, as the required iteration count for a problem rises (such as with higher-degree equations), LGP demands an increasing number of statements, and consequently larger models. However, SGP can achieve similar results with a consistent statement count, but with altered spatial

placements. To illustrate, adapting the solution for a^{40} instead of a^{20} only requires a minor spatial adjustment: the two programs, P1 and P4, must be positioned slightly closer, enabling them to iterate 41 times rather than 21.

On the other hand, both SGP and LGP fail to solve $c = a^5 + b^5$, an equation which theoretically requires two separate iterative components to add together to result in a perfect solution. This could potentially be due to not choosing a proper fitness function or data points for solving the problem, however, it also indicates that not every loop problem can be solved by SGP as simple as the ones that are solved here even though that SGP seems to be well-equipped for solving iterative problems.

3.3 Results: Localization of Programs

Localization manifests itself at multiple levels in SGP. Efforts have been made to underscore instances of localization throughout this chapter. This phenomenon surfaces at the individual, population, and program levels. At the LGP level, experimental outcomes indicate that each program tends to cluster instructions that cater to a particular function, and in certain scenarios, this operates as though a program is a specialized module designed to address a specific task. This behavioral pattern becomes more pronounced for tasks that are more interpretable than mere mathematical equations. Illustrating such behavior, is presented later in this contribution while tackling a variant of the Santa Fe ant problem.

At the population level, localized clusters emerge, suggesting that members of the population possess traits beneficial for addressing the task at hand. A spatial analysis of an individual pinpoints areas within the 2D space where these clusters form. The subsequent step involves identifying individuals possessing these significant spatial traits. Such localization, apparent upon spatial analysis, can serve as a straightforward method for examining and pinpointing similar traits within the population. The presence of these population clusters becomes clear when observing heatmaps that depict the spatial positions of a population.

Lastly, localization can also manifest itself at the individual level. Here, programs that

have synergistic functions in addressing a portion of a problem tend to gravitate spatially closer, ensuring they are executed consecutively. An illustration of this type of localization is evident in Figure 3.3, where the iterative nature of the target problem encourages P1 and P4 to be in close proximity, fostering the formation of loops between them.

To investigate the phenomenon of localization in the evolution of SGP programs in more detail, an additional series of experiments was undertaken. The goal was to ascertain how frequently programs gravitate towards one another to establish a cluster, which occupies less than 4% of the total permissible 2D space of an individual. For this analysis, the DBSCAN [Schubert et al., 2017] (Explained in details in the *Methods* section) algorithm was employed, setting the epsilon value at 30 and a minimum sample of 2, given that the complete allowed 2D space for program placement is a circle with a radius of 150. Three loop problems from Table 3.1 and equation number 18 from Table 4.2 were selected for this study ($P = \frac{q^2 a^2}{6\pi\epsilon c^3}$). Notably, equation 18 can be resolved without iteration, making it intriguing to measure if, in instances where program proximity doesn't induce iterative behavior, there still exists evidence of localization. The same experimental settings as outlined in table 3.2 were employed for resolving all problems, with the exception that no loops were permitted for the solution of equation 18.

Figure 3.4 depicts the average number of clusters formed during 50 replicates of SGP while evolving models to address the four designated problems. A clear pattern emerges: as the programs evolve, they exhibit an increased inclination for localization. This trend can be attributed to the fact that spatially proximate programs are more likely to be executed consecutively. Interestingly, for equation 18, which does not necessitate iteration, clustering still occurs and is not significantly diminished compared to the other scenarios.

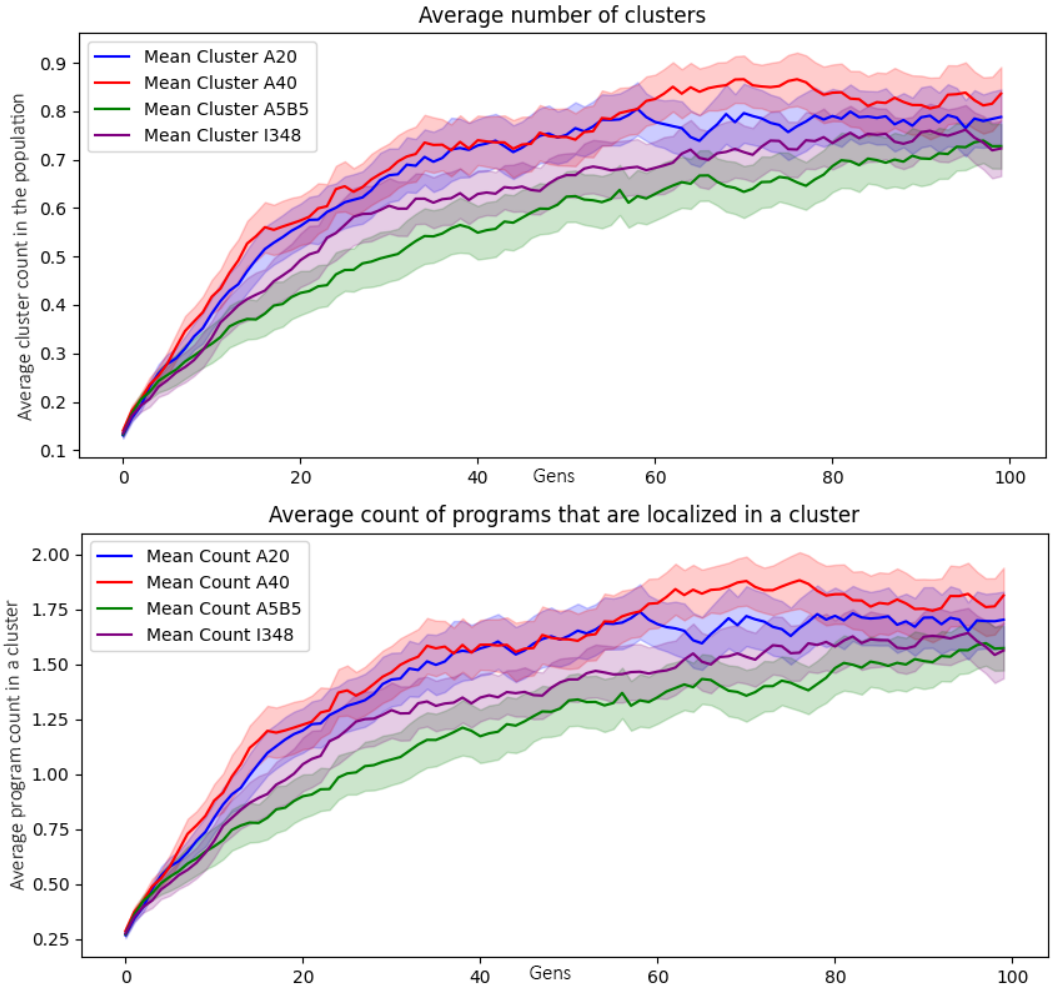


Figure 3.4: Localization of SGP programs. Lines are mean values and the shaded areas are 95% confidence intervals. Top figure shows the number of clusters formed on average in 50 replicates of each experiment over generation. Bottom figure shows the number of programs that are inside a cluster for each experiment over generations

3.4 Results: Impact of the Spatial Elements on Diversity

We examine the influence of integrating spatial components on the diversity of population members. Computationally, quantifying diversity within SGP individuals and contrasting different system configurations—or even comparing it with LGP—poses a challenge. As a result, we approach diversity assessment indirectly, focusing on the fitness diversity of individuals across generations and analyzing their structural variations.

3.4.1 Fitness Diversity

To measure the fitness diversity within a population, we calculated the distinct fitness values for each generation. Specifically, four configurations were used. A LGP configuration and three SGP configurations with spatial mutation rates of 20, 40 and 60. Figure 3.5a depicts the evolutionary progress of the best models over 20 generations. A shorter generation span was chosen to analyze the system phase where the majority of individuals haven't yet converged to an optimal solution. The evolutionary paths, upon comparison, show little differentiation, though LGP, on average, exhibits superior performance. Figure 3.5b visualizes the fitness diversity throughout generations for the four analyzed configurations. This experiment's setup mirrors that described in table 3.2, but with these variations: the population size is fixed at 100, the tournament size at 5, and only a single elite individual is carried over to the succeeding generation unaltered. Instead of relying on a correlation-based metric, we adopted an RMSE technique for fitness evaluation. This ensures that the recorded fitness values aren't influenced by subsequent calculations, as the metric is error-based. Every data point symbolizes the mean standard deviation value across 50 experimental replications for each setup. The error bars display the confidence interval for the standard deviations over the 50 repeated tests. This chart reveals that, on average, SGP configurations yield a notably reduced count of unique fitness values within a population over 20 evolutionary generations. A crucial point is that even though helpful, solely examining the diversity based on distinct

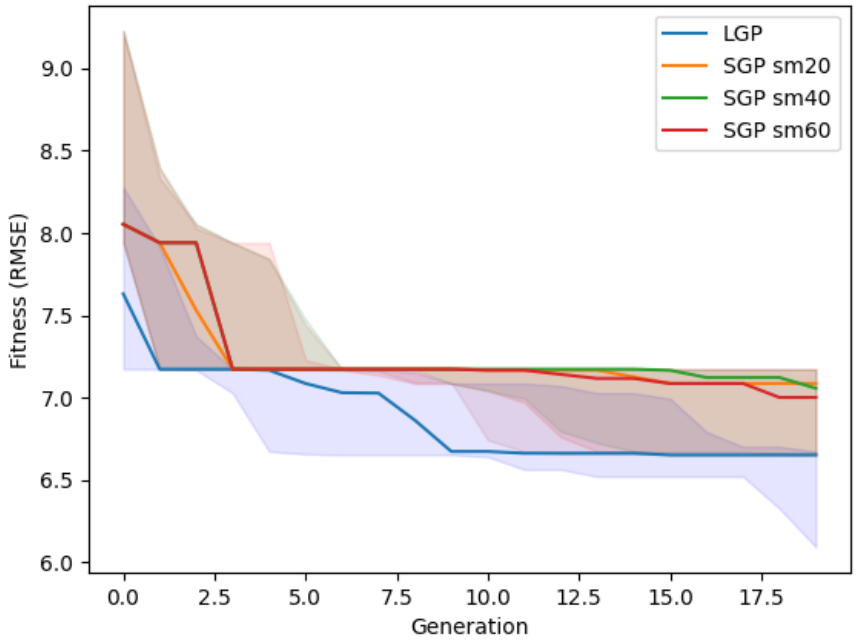
fitness values provides a limited perspective and doesn't inherently signal overall diversity. For instance, consider two individuals: Individual A, with a lone program and statement being $2 + 2$, always outputs 4. On the other hand, Individual B's sole instruction is $2 * 2 = 4$. While both models possess identical fitness values upon evaluation, Individual A investigates the application of addition, whereas Individual B delves into multiplication in their quest for the optimal solution. Despite their inherent diversity, their equivalent fitness values are not able to capture this distinction.

3.5 Structural Diversity

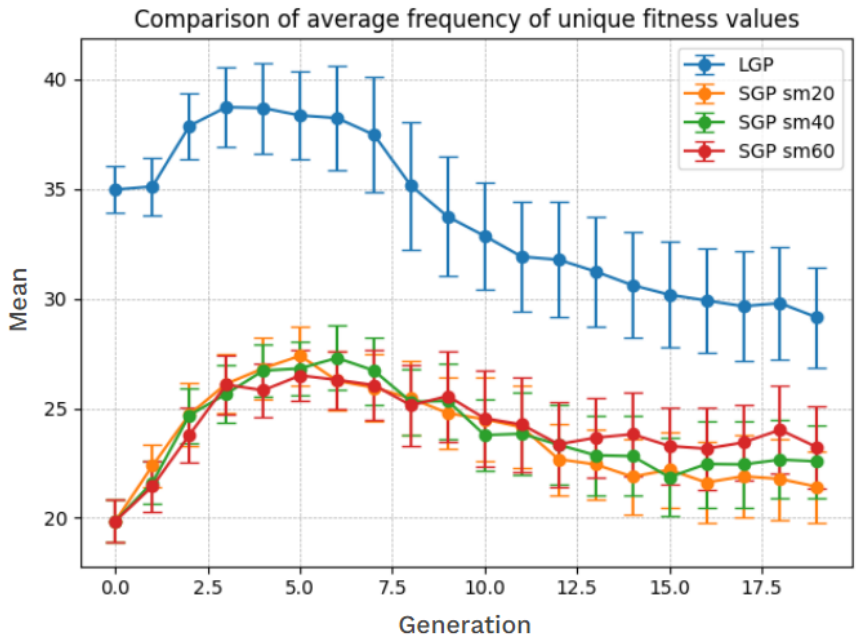
It is evident that SGP programs exhibit greater structural diversity compared to LGP, due to spatial operators that can dramatically alter the state of an individual model by adding or removing LGP programs. Quantifying this claim is challenging given the structural differences between LGP and SGP. Yet, another insightful diversity metric, which is somewhat structural and permits a comparison between the two systems, pertains to the number of statements executed by each individual model. Figure 3.6 presents the outcome of this comparison.

Analyzing the average number of executed statements over generations for each experiment's replicates reveals that, despite employing identical values for both the maximum and minimum statement counts for model initialization and evolution in LGP and SGP, the executed statement count in LGP surpasses that of the three SGP configurations by a significant margin. As a general principle, a greater number of executed statements likely results in more diverse system state alterations, thus yielding varying fitness values. This partially elucidates why LGP demonstrates a heightened fitness diversity compared to SGP configurations.

Conversely, Figure 3.6b, which displays the standard deviation for the number of executed programs across 50 replicates of each experiment throughout the generations, indicates that while LGP offers more diversity during the initial generations, this diversity diminishes rapidly. This suggests that in subsequent generations, LGP primarily probes a specific portion of



(a) Evolution of models



(b) Fitness diversity

Figure 3.5: Fitness diversity among individuals for different configurations. *sm* indicates spatial mutation. a) Shows how the best models from each replicate evolve during 100 generations. Lines indicate median values while the shaded areas indicate 75th percentiles b) Shows the fitness diversity observed among the individuals of a population. Lines are mean values and error bars indicate 95% confidence intervals

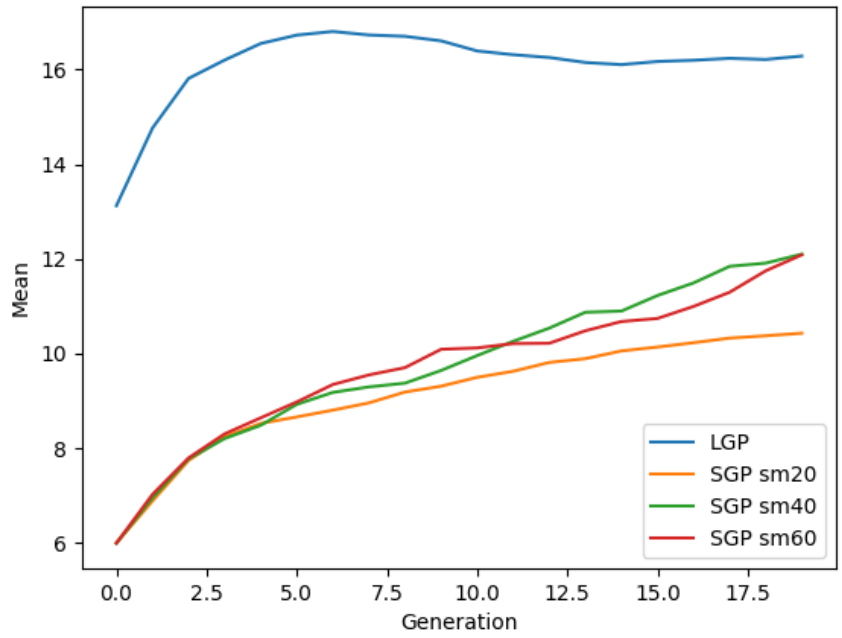
the search space, consistently engaging with a similar number of instructions (approximately 16 on average). In contrast, the diversity metric for SGP notably increases post the initial generations. This implies that SGP individuals maintain a diverse range of both shorter and longer programs.

3.6 Results: Structured Spatial Topology for Models

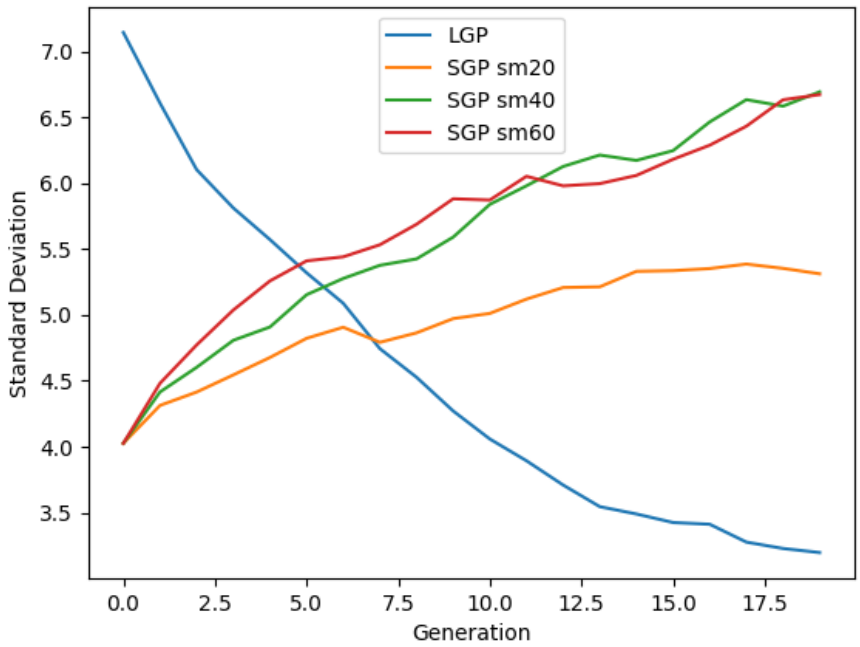
The incorporation of spatial elements in SGP systems offers a unique dynamism by enabling computer programs to maneuver within a 2D circle in an individual. The topology within which these programs operate can influence the performance and output, contingent on the nature of the target problem and the output extraction methodology. While different topologies may favor specific problems, the overarching aim of this work is not solely to test an array of spatial topologies and benchmark them against diverse problem types. Instead, it seeks to demonstrate the potential of SGP for investigating different topologies within the framework.

To illustrate this, several topologies were selected for the symbolic regression problem equation number 18 as a proof of concept. These encompass:

1. **Circle Topology**: Serving as the default configuration, the circle topology grants every program the liberty to position itself anywhere within a circle of radius *radius*.
2. **Lattice Topology**: This topology predetermines points on a lattice, contingent on the maximum permissible program size in SGP. Within each individual, programs sequentially occupy lattice positions starting from the zero index. Should an individual contain fewer programs than the maximum size, the residual lattice slots remain unoccupied. Specifically, for this topology, since changing position to a random point in the 2D space was not feasible, a special spatial mutation approach was used in which two programs have a chance to have their spatial position substituted.
3. **Line Topology**: In this configuration, programs can only migrate along a 2D line aligned with the x-axis.



(a) Average count of executed statements



(b) Statement Execution Diversity

Figure 3.6: Comparing structural diversity. a) Compares the mean count of executed statements between LGP and 3 SGP configurations b) Shows the standard deviation of average count of executed statements across the replicates for each generation

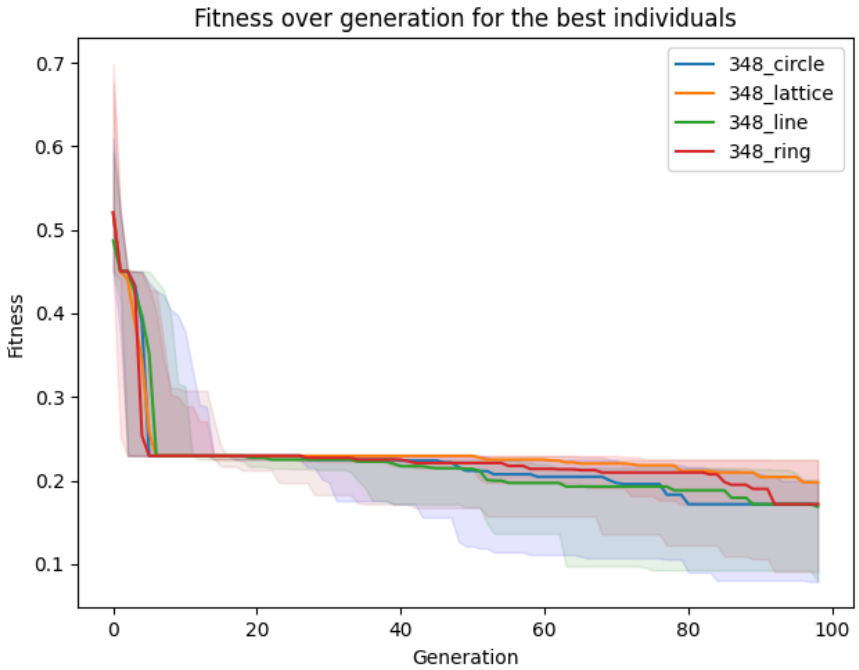


Figure 3.7: Evolution of models with different topologies for solving the regression problem number 18 from Table 4.2. The number 348 represents the ID of the equation. Lines represent median values while shaded areas are 75th percentiles

4. **Ring Topology**: Here, programs can move exclusively on the circle's perimeter, with the interior being off-limits.

Through this exploration, it becomes evident that SGP can adapt to a range of spatial topologies, with each potentially influencing the evolutionary trajectory and solution quality. Determining the optimal topology necessitates an understanding of the problem's nuances, combined with empirical testing to identify the most suitable spatial configuration.

Here, the experimental setup is the same as in table 3.2, but with the following alterations: the population size is set to 100, the tournament size to 5, and only a single elite is carried over to the succeeding generation unaltered. Figure 3.7 shows the results of this comparison. When observing the evolutionary trajectory across all topologies, no significant difference is evident among the individuals. The shaded areas, representing the 75th percentiles, suggest that individuals with the circle topology tend to be more diverse in terms of fitness and

Topology	Circle	Lattice	Line	Ring
Solved	18%	4%	12%	20%

Table 3.3: The percentage of the individuals that solve the problem

have more models with fitness values close to zero (perfect fitness). Table 3.3 displays the percentage of the programs that completely solve the given problem. For a basic symbolic regression problem, the increased spatial freedom of the ring and circle topologies likely enhances the performance of the individuals, especially when compared to the restrictive lattice topology and the somewhat constrained 1D line topology in which only 4% of the replicates solve the target problem.

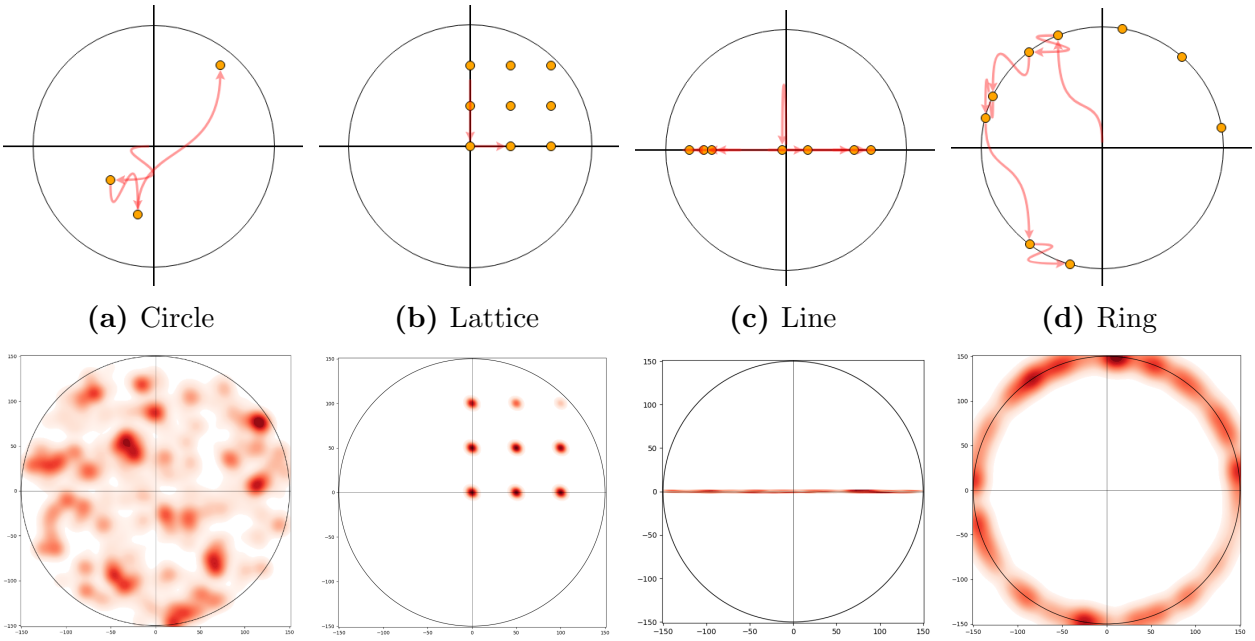
Figure 3.8 displays the spatial positions of a solution individual from each topology, along with their heatmap for four models that evolved to solve the problem. These figures are generated using the SGP analysis tool. Higher density points in the heatmap of the population indicate that a majority of the population's individuals have the same spatial position for programs at that particular point. This can be especially useful for identifying programs that are likely beneficial in solving the target problems.

3.7 Results: Spatial Evolutionary Operators

In this section, the impact of spatial evolutionary operators on the evolution of programs and their spatial positions is examined. A spatial crossover algorithm is introduced, and spatially-aware mutation rates are applied to the system.

3.7.1 Spatial Crossover

Figure 3.9 illustrates the high-level algorithm of the spatial crossover used in this experiment. Firstly, two fitter parents are selected through tournament selection. Let's refer to the top right quadrant of each individual model as Q1. Then, every program of Parent A located inside Q1 of that model will be transferred to Q1 of Offspring B, maintaining its position. The remaining programs of Parent A are transferred to Offspring A. The same process is applied



(a) Circle **(b)** Lattice **(c)** Line **(d)** Ring

(e) Circle Heatmap **(f)** Lattice Heatmap **(g)** Line Heatmap **(h)** Ring Heatmap

Figure 3.8: Schematic of the individuals and their respective population heatmap that solve the equation 18 problem. Four topologies of circle, lattice, line and ring were used

to Parent B, with the only difference being that the Q1 programs of Parent B are directed to Offspring A, while the rest of the programs of Parent B are transferred to Offspring B. If an offspring exceeds the total allowed number of programs, the extra programs (excluding the output program) are randomly selected for removal from the offspring to ensure size limit requirements are met.

3.7.2 Spatial Regions with Different Mutation Rates

An fundamental observation in biology is that the molecular clock of genes can vary, sometimes depending on the environment or the spatial location or orientation of the molecules. SGP enables us to identify zones or regions with distinct rules for mutation or crossover. While it's beyond the scope of this work, this approach could introduce a new form of diversity to the system. In this particular experiment, we designated a region within individuals where programs located therein experience an increased LGP mutation rate. Figure 3.10 depicts a general overview of the region in which LGP mutation rate is increased for the containing

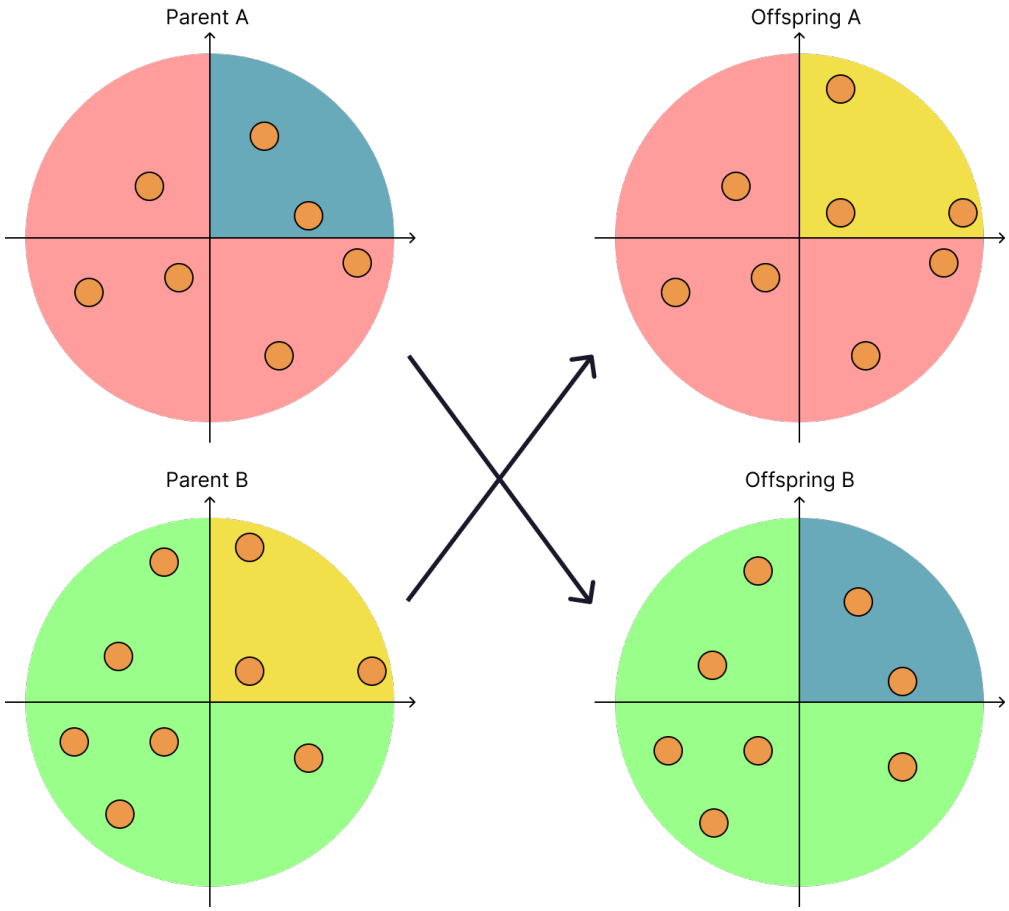


Figure 3.9: Spatial Crossover algorithm. The two programs of parent A residing in its Q1 are moved to the Q1 region of offspring B while maintaining their position. The three programs of parent B residing in its Q1 are moved to the Q2 region of offspring while maintaining their position. The rest of the programs of parent A are moved to offspring A and the rest of the programs of parent B are moved to offspring B

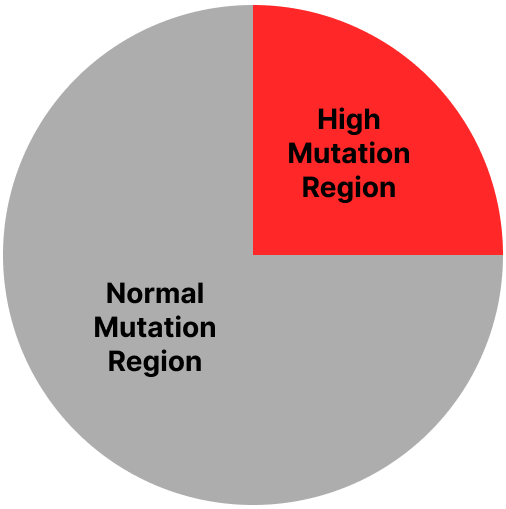


Figure 3.10: Different regions of the 2D space can have different mutation rates in SGP

programs. The rest of the 2D space represented by the grey area of the circle, follow the normal mutation rates.

Table 3.4 presents the results of incorporating the high mutation region and crossover methods described in this section. Once again, equation number 18 from Table 4.2 is chosen to be the target problem and the same experimental setup as table 3.2 is utilized. No significant difference is observed when examining the portion of problems that reside in each quadrant compared to the Base configuration where the crossover rate is set to zero. A similar study is performed in chapter 5, where the programmatic mode of SGP was used to tackle a set of control and decision-making problems. This suggested that spatial crossover causes individual programs to move away from the crossover zone. The results in this section for the fully spatial mode contradict the findings from the decision-making problems and the programmatic mode. The spatial mode of SGP only relies on the spatial properties of the programs of an individual to determine their execution order. On the other hand, in the programmatical mode, logical elements such as problem inputs also impact this order. The spatial crossover introduced in this section, does not account for such logical elements causing it to be more destructive in the programmatical mode, and pressuring the programs to drift away from Q1. It's noteworthy that the total count of the programs drops in cases

where crossover exists. This occurs because when two parents recombine to produce two offspring, an offspring might exceed the maximum allowed size of programs. In such cases, additional programs are removed from the offspring, leading to a decrease in the total number of programs in the population compared to scenarios without crossover.

In Table 3.4, results for having regions with high mutation rates are indicated by different mutation rates of 20%, 40%, and 60%, which represent the additional rate applied to the programs located in Q1. The base LGP mutation rate for all individuals is set at 20%. In this context, the cumulative LGP mutation rate for Q1 programs becomes 40%, 60%, and 80%. There is no decrease in the total count of the programs, although the number slightly rises on average (except for the case with a mutation rate of 40%) when mutation regions are present. Conversely, the density of programs in Q1 diminishes by a little over 1%.

Figure 3.11 illustrates the evolution of SGP with different configurations over 100 generations. The base setup does not include any additional spatial crossover or high mutation regions. Although no benefits from the spatial evolutionary operators are evident in this figure, it is clear that changes in the spatial properties of the 2D space, brought about by spatial evolutionary operators, influence the evolution of individual models. Figure 3.12 displays the heatmap of the entire population from the experiments conducted in this section. Aside from the observation that the Q1 region in Figure 3.12f appears less dense than in other figures, these heatmaps do not provide clear additional insights about the spatial operators. However, it's noteworthy that even at the population level, dense clusters of programs form. This localization is less pronounced in cases where the crossover rate is set to 100% or when the regional mutation rate is set to 20%, with programs appearing more uniformly distributed.

3.8 Results: Moving to a 3D space

In this section, we explore the concept of introducing additional dimensions to the system and simulating a 3D structure for individual models. There were two motivations behind this experiment : 1) to determine if adding more dimensions would correlate with increased

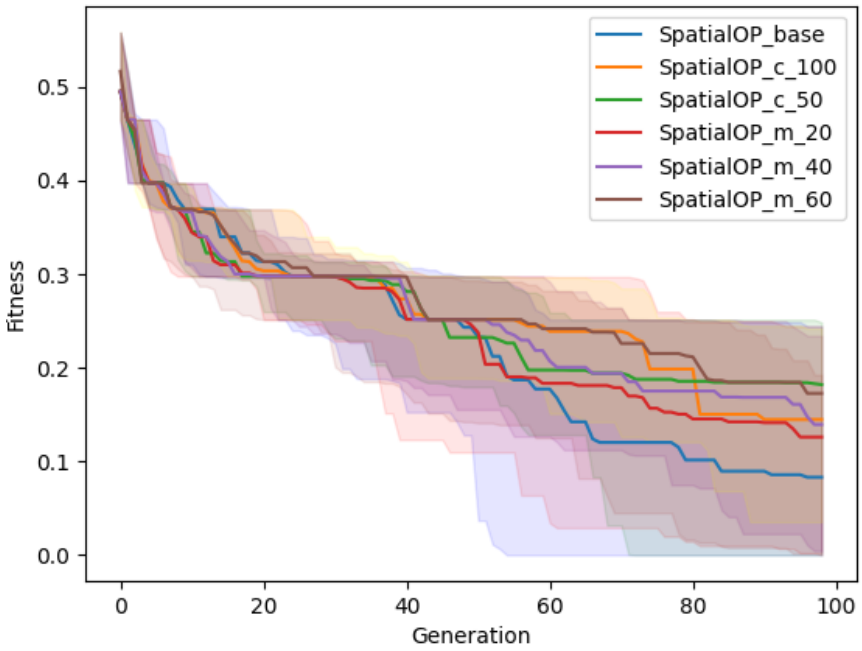
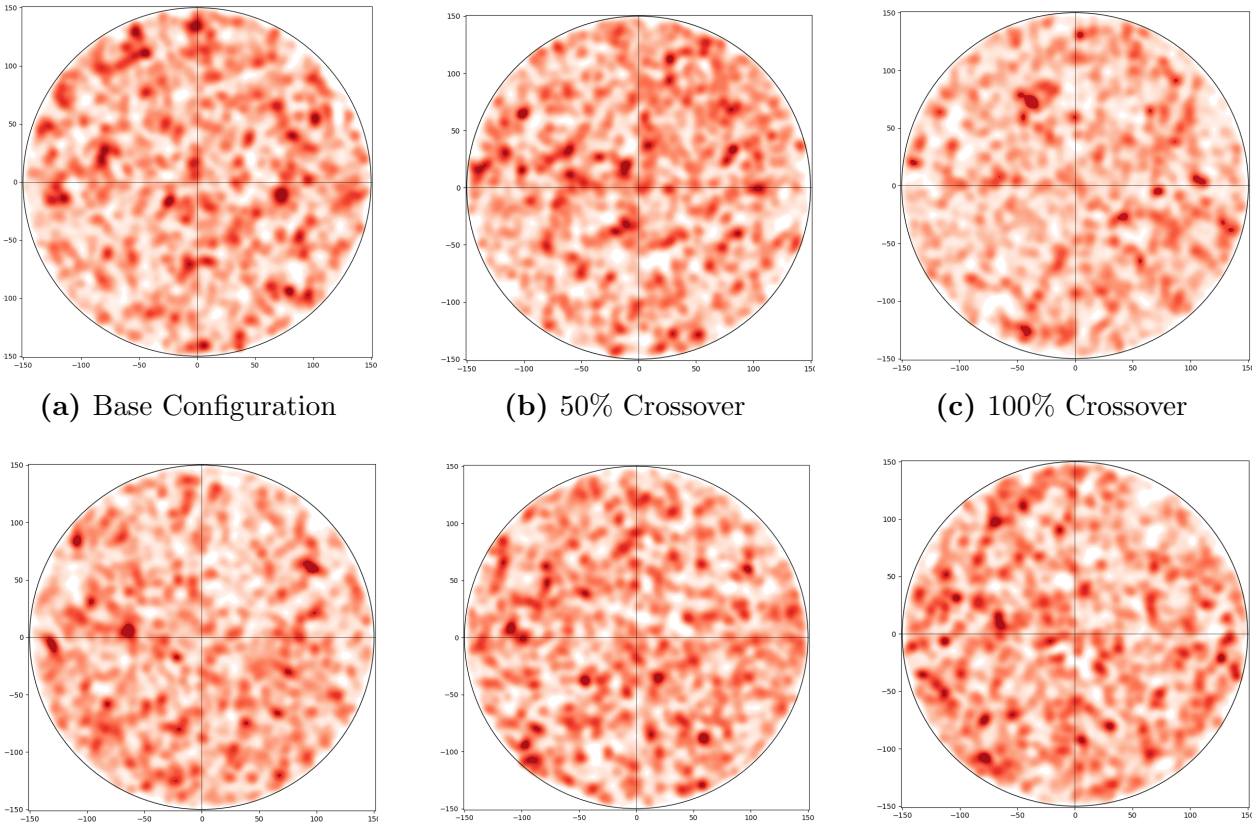


Figure 3.11: Evolution of different configurations of SGP with spatial crossover and high mutation regions

Setup	Q1	Q2	Q3	Q4	Total Count
Base	25.06%	25.07%	24.36%	25.52%	35959
Crossover rate = 50	26.18%	24.18%	24.04%	25.56%	32879
Crossover rate = 100	25.20%	25.70%	24.16%	24.95%	31660
Mutation rate = 20	23.33%	25.78%	25.75%	25.13%	36475
Mutation rate = 40	23.83%	25.55%	24.97%	25.62%	35232
Mutation rate = 60	23.37%	25.31%	25.70	25.61	37006

Table 3.4: Impact of different spatial evolutionary operators on the arrangement of the programs in the 2D space



(a) Base Configuration **(b)** 50% Crossover **(c)** 100% Crossover

(d) 20% Spatial Mutation Rate Increase **(e)** 40% Spatial Mutation Rate Increase **(f)** 60% Spatial Mutation Rate Increase

Figure 3.12: Comparison of the population heatmap produced after applying different spatial operators

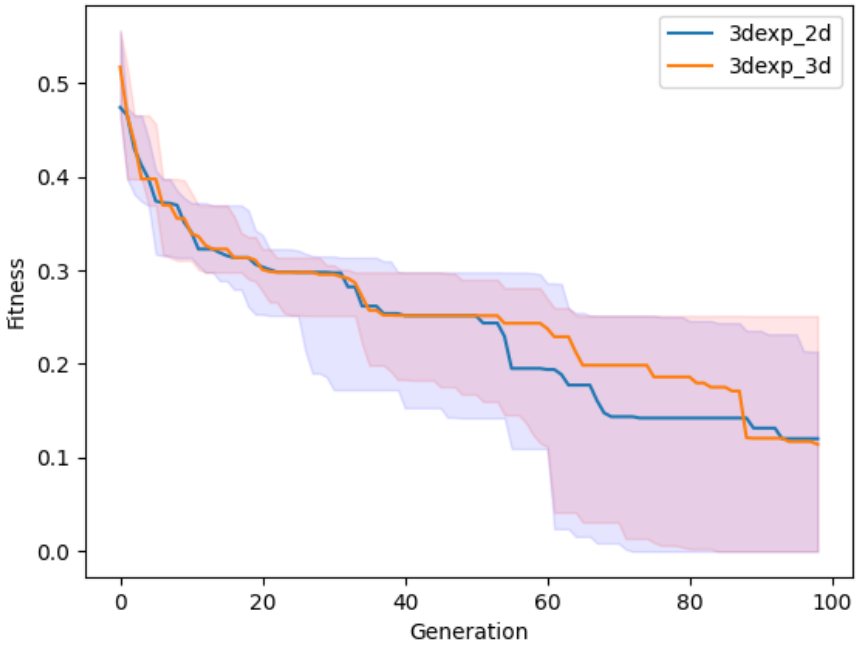


Figure 3.13: Evolutionary performance of SGP in a 2D and a 3D spatial setup

control over the spatial impact of the system, and 2) to investigate a system with fundamental dimensions more closely aligned with real-world biological systems.

Figure 3.13 displays the evolutionary dynamics produced from both 2D and 3D SGP systems that solve the same symbolic regression equation (equation number 18 from Table 4.2) across 50 replicates. No significant difference was observed when comparing the two algorithms. Indeed, for most generations, the two systems operate almost identically, with the shaded areas representing the 75th percentile not showing much variation between the two systems. This suggests that merely adding more spatial dimensions to the system, without taking into account problem-domain specifics does not necessarily lead to a change in system performance.

3.9 Methods

The methods used in this research are explained in more details in this section:

3.9.1 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [Schubert et al., 2017] is a clustering algorithm that groups together points that are closely packed together, marking as outliers points that lie alone in low-density regions. This algorithm is particularly effective for identifying clusters of arbitrary shape in a data set, as opposed to algorithms like k-means, which assume spherical clusters.

DBSCAN primarily requires two parameters:

Epsilon ϵ: This defines the radius of the neighborhood around a point. It determines how close points should be to be considered part of a cluster.

MinPts (Minimum Points): The minimum number of points required to form a dense region. A point is considered a core point of a cluster if it has at least MinPts within its ϵ neighborhood.

These parameters are crucial as they control the scale and density of the clusters identified by the algorithm.

3.10 Conclusion

In this chapter, we explored the dynamics and implications of SGP in evolving computational models, focusing on the impact of spatial elements on program structure, diversity, and performance. Our investigation revealed several key insights:

Program Localization and Clustering: We observed a distinct pattern of increasing localization and clustering in SGP models as they evolved. This trend was consistent across different problem scenarios, indicating a natural inclination of SGP systems towards spatial organization through localization.

Diversity in Fitness and Structure: The integration of spatial components was found to influence the diversity within the SGP population. While LGP populations had significantly higher fitness diversity, the same was not the case when comparing the structural diversity of the two algorithms. SGP primarily due to its spatial operators caused more

dramatic alterations in the models.

Impact of Spatial Topologies: The study demonstrated that SGP could adapt to various spatial topologies (Circle, Lattice, Line, Ring), each influencing the evolutionary trajectory and solution quality differently. This adaptability highlights the potential of SGP in exploring diverse solution spaces through topologies and indicated that some topologies outperform the others with regards to the target problem.

Spatial Evolutionary Operators: The introduction of spatial crossover and region-specific mutation rates introduced new dynamics to the evolutionary process. While these operators did not show a significant advantage in the context of the problems and configurations tested, they underscore the potential for more nuanced and biologically-close control in the evolutionary process.

Extension to 3D Space: Expanding the SGP model to a 3D space aimed to align more closely with real-world biological systems and provide increased control. However, this extension did not result in notable performance differences compared to the 2D setup for the problems considered.

Overall, this chapter contributes to the understanding of SGP by highlighting how spatial elements and topologies can influence the evolution and capabilities of genetic programming models. The findings suggest that while spatial structures and operators introduce complexity, they also offer new avenues for enhancing diversity and problem-solving efficacy in genetic programming. This work lays the groundwork for further exploration into more sophisticated spatial models and their applications in various problem domains.

Chapter 4

SGP for Solving Regression Problems

Parts of this chapter are adopted from [Miralavy and Banzhaf, 2023a].

4.1 Introduction

This chapter focuses on a more practical viewpoint of this research by applying SGP on a series of Symbolic Regression problems. This problem class is specifically chosen, because of how common it is to apply GP algorithms for solving them in the previous literature and can give us a good understanding of the capability of the proposed algorithm when compared against the common GP systems as a proof of concept that the SGP system works.

4.2 Symbolic Regression Problems

Regression is a type of problems where the goal is to find a function that best fits a series of observations. In some cases, the form of the underlying function is known and the goal is to find the correct coefficients to fit the function as much as possible to the observations. However, in cases that the form of the function is also unknown, the problem is referred to as a SR problem [Augusto and Barbosa, 2000]. This is a type of problem that requires the problem-solving model to be transparent, making GP a suitable candidate for solving them. For example, GPTIPS [Searson et al., 2010] is an open source, easy to access and easy to

implement tool for solving SR problems in MATLAB. This tool evolves both incorporates a multi-gene representation and evolves both the structure and the hyper-parameters of the regression models. [Zhong et al., 2018] employed the power of a multi-factorial GP algorithm to solve multiple SR problems in a single run showing the efficacy of such models. [Haut et al., 2022] investigated on the impact of incorporating active learning on the performance of StackGP algorithm for solving SR problems. Numerous other researches have been conducted before in the literature showing how GP can be utilized and customized to solve SR problems [Astarabadi and Ebadzadeh, 2019]. Some focus on the evolutionary operators to improve the performance of GP in solving SR problems [Uy et al., 2011, Uy et al., 2009], some propose techniques which combines GP with machine learning algorithms [Mundhenk et al., 2021,Icke and Bongard, 2013] or some focus on the selection schemes that can improve GP for this purpose [Chen et al., 2017, Martínez et al., 2014].

4.3 Feynman Symbolic Regression Problems

The spatial mode of SGP is designed to be adept in addressing continuous value problems while the programmatical mode is more equipped to solve decision-making tasks. Seeking to delve deeper into this observation, we designed an experiment drawing upon 92 equations from the Feynman lectures on Physics [Feynman et al., 2011]. These equations served as our testing ground to evaluate the performance of SGP. The results generated from this were compared against those obtained from utilizing TGP and LGP. Importantly, for a fair comparison, the same LGP system embedded within SGP was employed. To expedite the LGP implementation, we used the capabilities of the DEAP framework [Fortin et al., 2012] which is explained in details further in the chapter. Configurations of each system can be found in table 4.1.

Parameter	Description	Value
g	Number of evolutionary generations	100
$Size_{pop}$	Population size of the experiment	300
$Size_{tournament}$	Size of the tournament selection pool	10
e	Number of elites that are directly selected to be a part of the next generation	3
$m_{spatial}$	Spatial mutation rate	40%
m_{LGP}	LGP mutation rate	20%
C	Crossover rate	0%
$cost_i$	Cost function	$\ln(1 + distance)$
$Topology$	Describes any topological limitation for placing nodes on the 2D space	circle
$Type_{output}$	Describes the output type of the system	single
$Count_{evaluation}$	Number of points used for evaluation of an individual model	30
$Count_{registers}$	Number of internal system registers	inputs + 2
$initSize_{max}$	Maximum number of programs per individual during initialization	10
$Size_{max}$	Maximum number of programs per individual	15
$lgpInitSize_{max}$	Maximum number of statements per program during initialization	10
$lgpSize_{max}$	Maximum number of statements per program	15
T	Maximum individual evaluation time	100ms
$radius$	Size of the topology representing the 2D space	150
Set_{OP}	Operator or function set	$+, -, \times, /, e, \sqrt{}, sin, cos, log$
$Constants$	Set of constant values	-1, 1, 2, 3, 5
$RevisitPenalty$	Penalty for revisiting a node when loop is allowed	0.01
$reps$	Number of replicates for each experiment	50

Table 4.1: Parameter setup of SGP. Relevant parameters for LGP and TGP are chosen similarly

Index	EQ ID	Equation	SGP	LGP	Tree GP
1	I.10.7	$m = \frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}}$			
2	I.12.1	$F = \mu N_n$	Yes	Yes	Yes

Table 4.2: Symbolic regression equations sourced from Feynman's lectures on physics, alongside the performance of the algorithms studied for solving these problems

Table 4.2 (cont'd)

Index	EQ ID	Equation	SGP	LGP	Tree GP
3	I.12.2	$F = \frac{q_1 q_2}{4\pi\epsilon r^2}$	Yes	Yes	
4	I.12.4	$E_f = \frac{q_1}{4\pi\epsilon r^2}$	Yes	Yes	
5	I.12.5	$F = q_2 E_f$	Yes	Yes	Yes
6	I.13.4	$K = \frac{1}{2}m(v^2 + u^2 + w^2)$			
7	I.14.3	$U = mgz$	Yes	Yes	Yes
8	I.14.4	$U = \frac{k_{spring}x^2}{2}$	Yes	Yes	Yes
9	I.15.3t	$t_1 = \frac{t - ux/c^2}{\sqrt{1-u^2/c^2}}$			
10	I.15.3x	$x_1 = \frac{x - ut}{\sqrt{1-u^2/c^2}}$			
11	I.16.6	$v_1 = \frac{u+v}{1+uv/c^2}$			
12	I.18.4	$r = \frac{m_1 r_1 + m_2 r_2}{m_1 + m_2}$			
13	I.24.6	$E = \frac{1}{4}m(\omega^2 + \omega_0^2)x^2$			
14	I.27.6	$f_f = \frac{1}{\frac{1}{d_1} + \frac{n}{d_2}}$	Yes	Yes	
15	I.29.4	$k = \frac{\omega}{c}$	Yes	Yes	Yes
16	I.30.3	$I^* = I_0^* \frac{sin^2(n\phi/2)}{sin^2(\phi/2)}$			
17	I.32.5	$P = \frac{q^2 a^2}{6\pi\epsilon c^3}$			
18	I.34.8	$\omega = \frac{qvB}{p}$	Yes	Yes	Yes
19	I.37.4	$I^* = I_1 + I_2 + 2\sqrt{I_1 I_2}cos\delta$			

Table 4.2 (cont'd)

Index	EQ ID	Equation	SGP	LGP	Tree GP
20	I.40.1	$n = n_0 e^{(-\frac{mgx}{k_b T})}$			
21	I.44.4	$E = nk_bT\ln(\frac{V_2}{V_1})$			
22	I.6.20	$f = e^{-\frac{\theta^2}{2\sigma^2}}/\sqrt{2\pi\sigma^2}$			
23	I.6.20a	$f = e^{-\frac{\theta^2}{2}}/\sqrt{2\pi}$		Yes	
24	I.6.20b	$f = e^{-\frac{(\theta-\theta_1)^2}{2\sigma^2}}/\sqrt{2\pi\sigma^2}$			
25	I.8.14	$d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$			
26	I.9.18	$F = \frac{Gm_1m_2}{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}$			
27	I.11.19	$A = x_1y_1 + x_2y_2 + x_3y_3$			
28	I.12.11	$F = q(E_f + Bvsin\theta)$			
29	I.13.12	$U = Gm_1m_2(\frac{1}{r_2} - \frac{1}{r_1})$			
30	I.15.10	$p = \frac{m_0v}{\sqrt{1-v^2/c^2}}$			
31	I.18.12	$\mathcal{T} = rFsin\theta$	Yes	Yes	
32	I.18.16	$L = mrvsin\theta$	Yes	Yes	
33	I.25.13	$V_e = \frac{q}{C}$	Yes	Yes	Yes
34	I.29.16	$x = \sqrt{x_1^2+x_2^2-2x_1x_2cos(\theta_1-\theta_2)}$			
35	I.32.17	$P = (\frac{1}{2}\epsilon cE_f^2)(8\pi r^2/3)(\omega^4/(\omega^2 - \omega_0^2)^2)$			

Table 4.2 (cont'd)

Index	EQ ID	Equation	SGP	LGP	Tree GP
36	I.34.10	$\omega = \frac{\omega_0}{1-v/c}$	Yes	Yes	
37	I.34.14	$\omega = \frac{1+v/c}{\sqrt{1-v^2/c^2}}\omega_0$			
38	I.34.27	$E = h\omega$	Yes	Yes	Yes
39	I.38.12	$r = \frac{4\pi\epsilon\hbar^2}{mq^2}$	Yes	Yes	
40	I.39.10	$E = \frac{3}{2}pFV$	Yes	Yes	Yes
41	I.39.11	$E = \frac{1}{\gamma-1}pFV$	Yes	Yes	
42	I.39.22	$P_F = \frac{nk_bT}{V}$	Yes	Yes	
43	I.41.16	$L_{rad} = \frac{\hbar\omega^3}{\pi^2c^2(e^{\frac{\hbar\omega}{k_bT}}-1)}$			
44	I.43.16	$v = \frac{\mu_{drift}qV_e}{d}$	Yes	Yes	Yes
45	I.43.31	$D = \mu_e k_b \mathcal{T}$	Yes	Yes	Yes
46	I.43.43	$\kappa = \frac{1}{\gamma-1}\frac{k_bv}{A}$	Yes	Yes	
47	I.47.23	$c = \sqrt{\frac{\gamma pr}{\rho}}$	Yes	Yes	
48	I.48.20	$E = \frac{mc^2}{\sqrt{1-v^2/c^2}}$			
49	I.50.26	$x = x_1[cos(\omega t) + acos(\omega t)^2]$			
50	II.8.7	$E = \frac{3}{5}\frac{q^2}{4\pi\epsilon d}$	Yes	Yes	Yes
51	II.10.9	$E_f = \frac{\sigma_{den}}{\epsilon}\frac{1}{1+\chi}$	Yes	Yes	
52	II.11.3	$x = \frac{qE_f}{m(\omega_0^2-\omega^2)}$			

Table 4.2 (cont'd)

Index	**EQ ID**	**Equation**	**SGP**	**LGP**	**Tree GP**
53	II.15.4	$E = -\mu_M B cos\theta$	Yes	Yes	Yes
54	II.15.5	$E = -p_d E_f cos\theta$	Yes	Yes	Yes
55	II.2.42	$P = \frac{\kappa(T_2-T_1)A}{d}$	Yes	Yes	
56	II.3.24	$F_E = \frac{P}{4\pi r^2}$	Yes	Yes	
57	II.34.2	$\mu_M = \frac{qvr}{2}$	Yes	Yes	
58	II.34.2a	$I = \frac{qv}{2\pi r}$	Yes	Yes	
59	II.37.1	$E = \mu_M(1+\chi)B$	Yes	Yes	Yes
60	II.38.3	$F = \frac{YAx}{d}$	Yes	Yes	Yes
61	II.4.23	$V_e = \frac{q}{4\pi\epsilon r}$	Yes	Yes	
62	II.6.15b	$E_f = \frac{3}{4\pi\epsilon}\frac{P_d}{r^3} cos\theta sin\theta$			
63	II.8.31	$E_{den} = \frac{\epsilon E_f^2}{2}$	Yes	Yes	Yes
64	II.11.20	$P^* = \frac{n_\rho p_d^2 E_f}{3k_b T}$	Yes	Yes	
65	II.11.28	$\theta = 1 + \frac{na}{1-(na/3)}$	Yes	Yes	
66	II.13.17	$B = \frac{1}{4\pi\epsilon c^2}\frac{2I}{r}$	Yes	Yes	
67	II.13.23	$\rho_c = \frac{\rho_{c_0}}{\sqrt{1-v^2/c^2}}$			
68	II.13.34	$j = \frac{\rho c_0 v}{\sqrt{1-v^2/c^2}}$			
69	II.21.32	$V_e = \frac{q}{4\pi\epsilon r(1-v/c)}$			

Table 4.2 (cont'd)

Index	**EQ ID**	**Equation**	**SGP**	**LGP**	**Tree GP**
70	II.27.16	$F_E = \epsilon c E_f^2$	Yes	Yes	
71	II.27.18	$E_{den} = \epsilon E_f^2$	Yes	Yes	Yes
72	II.34.11	$\omega = \frac{gqB}{2m}$	Yes	Yes	
73	II.34.29a	$\mu_M = \frac{qh}{4\pi m}$	Yes	Yes	Yes
74	II.34.29b	$E = \frac{g\mu_M B J_z}{\hbar}$	Yes	Yes	
75	II.35.18	$n = \frac{n_0}{exp(\mu_m B/k_b T)) + exp(-\mu_m B/(k_b T)))}$			
76	II.36.38	$f = \frac{\mu_m B}{k_b T} + \frac{\mu_m \alpha M}{\epsilon c^2 k_b T}$			
77	II.38.14	$\mu_S = \frac{Y}{2(1+\sigma)}$	Yes	Yes	
78	II.4.32	$n = \frac{1}{e^{\frac{\hbar\omega}{k_b T}} - 1}$			
79	II.4.33	$E = \frac{\hbar\omega}{e^{\frac{\hbar\omega}{k_b T}} - 1}$			
80	III.7.38	$\omega = \frac{2\mu_M B}{\hbar}$	Yes	Yes	Yes
81	III.8.54	$p_\gamma = sin(\frac{Et}{\hbar})^2$		Yes	
82	III.9.52	$p_\gamma = \frac{p_d E_f t}{\hbar} \frac{sin((\omega-\omega_0)t/2)^2}{((\omega-\omega_0)t/2)^2}$			
83	III.10.19	$\mu_M = \sqrt{B_x^2 + B_y^2 + B_z^2}$			
84	III.12.43	$L = n\hbar$	Yes	Yes	Yes
85	III.13.18	$v = \frac{2Ed^2k}{\hbar}$	Yes	Yes	
86	III.14.14	$I = I_0(e^{\frac{qV_e}{k_b T}} - 1)$	Yes	Yes	

Table 4.2 (cont'd)

Index	EQ ID	Equation	SGP	LGP	Tree GP
87	III.15.12	$E = 2U(1 - cos(kd))$	Yes	Yes	
88	III.15.14	$m = \frac{\hbar^2}{2Ed^2}$	Yes	Yes	
89	III.15.27	$k = \frac{2\pi\alpha}{nd}$	Yes	Yes	Yes
90	III.17.37	$f = \beta(1 + \alpha cos\theta)$	Yes	Yes	
91	III.19.51	$E = \frac{-mq^4}{2(4\pi\epsilon)^2\hbar^2}\frac{1}{n^2}$			
92	III.21.20	$j = \frac{-\rho c_0 q A_{vec}}{m}$	Yes	Yes	Yes

4.4 Results:

Experimental parameters for comparing the three systems were empirically chosen. While the configuration of LGP and SGP differ mainly in that SGP introduces spatial elements to the system, the LGP programs remain identical across both systems. For evaluating individuals in all three systems, we use a correlation-based fitness metric with the goal of minimizing the equation:

$$f = 1 - r^2$$

Here, f denotes the fitness value of an individual, and r represents the Pearson correlation coefficient [Cohen et al., 2009]. This coefficient, calculated between a set of measured and estimated values for solving each equation, ranges from -1 to 1. An r value of 0 signifies no correlation between the data set and the GP model; a value of 1 implies a perfect positive correlation, while -1 indicates a perfect negative correlation.

Table 4.2 provides a summary of the results from this experiment. A *Yes* value in an

algorithm column signifies the algorithm's success in identifying at least one generalized solution for the corresponding equation. Of the 92 problems, both SGP and LGP delivered comparable performances, outperforming TGP. Specifically, SGP solved 52 problems, LGP 54, and TGP 22. Equations number 23 and 81 were uniquely solved by LGP but remained unsolved by SGP. However, it's worth noting that only a handful of the 50 LGP replicates were able to find a solution for these two equations, highlighting their complexity.

Extending the search to 1,000 generations yielded no significant differences in the results. Most individuals either resolved the problem within the first 100 generations or failed to do so entirely. Each individual was evaluated using 30 data points within the range of (0, 5]. Looking at table 4.2 most of the more trivial equations were solved by the three algorithms.

4.5 Methods

Methods for running the experiments conducted in this chapter are explained here:

4.5.1 DEAP Framework

DEAP [Fortin et al., 2012] is an evolutionary computation framework which uses a design principle similar to SGP in several occasions. For example, through utilizing and configuring two objects of *creator* and *toolbox*, it serves as a centralized unit that puts together different modules responsible for different parts of the evolutionary algorithm. This enables users to replace different modules of the EA to construct custom algorithms suitable for solving problems. This framework comes with the implementations of algorithms such as GA, TGP, Evolution Strategies [Beyer and Schwefel, 2002] and Particle Swarm Optimization algorithm [Poli et al., 2007] while enabling users to craft and construct their own type of EA.

4.5.2 Pearson r and a Correlation-Based Fitness Function

Given two variables representing two series of points, Pearson Correlation Coefficient r determines the strength and the direction of the relation between the two series. The following formula is used to calculate the Pearson r coefficient:

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \sum(Y_i - \bar{Y})^2}}$$

in which X_i and Y_i are the two series of points, and $\bar{X}$ and $\bar{Y}$ represent the mean values for X_i and Y_i respectively. r is a value between -1 and 1. Values higher than 0.5 indicate a strong positive correlation, values lower than -0.5 indicate a strong negative correlation. A value of 0, indicates no correlation between the two series while values of 1 and -1, indicate a perfect positive and negative correlation between the two series, respectively. In order to calculate the r value for solving SR problems in SGP, each individual model was evaluated 30 times for each solving each equation. The estimated outputs by SGP and the actual output for each given sets of inputs were stored creating two series of numbers containing 30 data points. Next, using the implementation of Pearson r in the stats module of the scipy [Virtanen et al., 2020] public Python library, the correlation value between the two series were calculated. We then use the formula discussed in section 4.4 to calculate our fitness value.

4.6 Conclusion

This chapter presented an in-depth exploration of the application of SGP in addressing Symbolic Regression problems, a domain traditionally targeted by GP algorithms. In the experimental study, using a benchmark consisting of 92 equations from the Feynman lectures on Physics, a comparison between SGP, LGP, and TGP was performed.

The results of these experiments indicate the effectiveness of SGP in solving SR problems. SGP demonstrated a competitive performance, successfully solving 52 out of the 92 problems, closely rivaling LGP, which solved 54, and significantly outperforming TGP, which solved only 22.

The use of the Pearson correlation coefficient as a fitness metric proved to be a robust method for evaluating the performance of the algorithms. On an experimental study not included in this chapter, this metric proved to have significantly better results to a RMSE-

based metric. It's interesting to note that the majority of the problems were either solved within the first 100 generations or not at all, suggesting that the initial stages of the evolutionary process are crucial in these GP systems. Using larger populations often yielded better results; however, to ensure the experiment's feasibility with respect to computational cost, a population size of 300 was chosen. Moreover, the fact that only a few LGP replicates could solve the more complex equations like numbers 23 and 81 highlights the challenging nature of these problems and the potential need for further algorithmic refinement or more extended evolutionary runs with different configurations.

In conclusion, this chapter demonstrates the viability of SGP in solving SR problems and opens opportunities for future research through enhancing the utilized algorithms. The close performance of SGP and LGP, along with the distinct advantage over TGP, positions SGP as a promising tool in the field of symbolic regression. Future work could explore the optimization of SGP parameters, the integration of other types of evolutionary algorithms within SGP, and the application of SGP to a broader range of problems beyond the scope of symbolic regression.

Chapter 5

SGP for Decision Making Problems

Parts of this chapter are reproduced with permission from Springer Nature from [Miralavy and Banzhaf, 2023c]. A small portion of this chapter is adopted from [Miralavy and Banzhaf, 2023a].

5.1 Introduction

In the previous chapters we utilized SGP in its spatial mode for solving a series of SR problems. We also performed a spatial analysis on the system, investigating localization and quantifying two instances of diversity. An evident characteristic of SGP is how it allows the individual topologies to form free of constraint. However, in the spatial mode, this order is deterministic and is not subject to change with regards to the problem inputs, significantly undermining SGP's power in it's spatial mode for solving decision-making problems. In this chapter, we study an extension of SGP which is adept in forming dynamic conditional pathways when determining the order of the execution of the LGP programs within individuals. The primary contribution of this chapter is introducing SGP in its programmatical mode. Same as the spatial mode, this GP system is controlled by a 2D space that evolves LGP programs. In the spatial mode of SGP, *space* plays the primary role in determining the order of execution of LGP programs, however, in the programmatical mode of the system, it is not the only factor

in doing so. In each individual, the program nodes form a network of interactions, responsible for regulating the order of execution of the LGP programs based on their spatial properties and the internal dynamics of the system. If necessary, the flexible representation of SGP allows for controlling the evolution of iterative behavior to develop more compact models. To show the effectiveness of the proposed system, we utilize SGP to solve different classes of problems which require decision-making and compare it to two common GP paradigms.

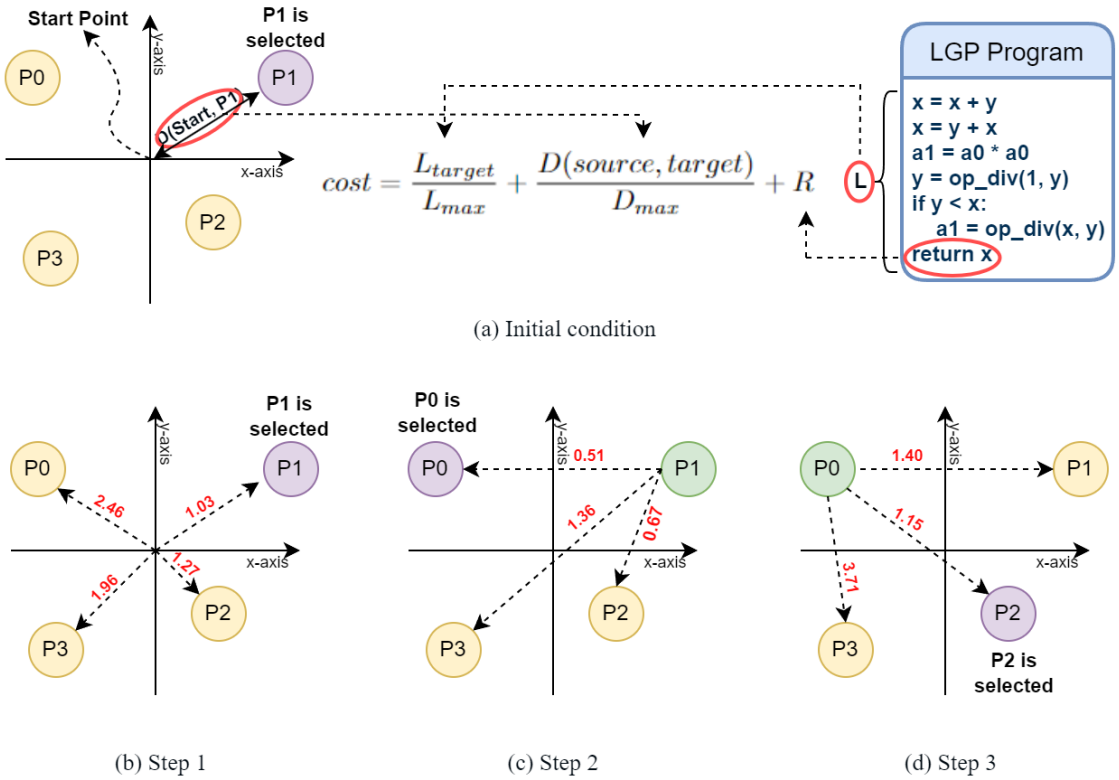


Figure 5.1: Model representation and interpretation steps for an SGP model with 4 programs in its programmatical mode a) Different contributions to the cost function. b) Step 1: P1 is selected since it has the lowest traverse cost from the starting point. Red values indicate cost c) Step 2: P1 is the source point and P0 with the lowest traverse cost is selected. d) Step 3: P0 is the starting point and P2 is selected

5.2 Programmatical Spatial Genetic Programming

As discussed, SGP models are program nodes spread in a 2D coordinate system. The aim of the SGP interpreter is to choose the order of program executions until a termination condition

is met while minimizing the traversing cost between programs. In the programmatical mode of SGP, a different cost function is employed to calculate the cost of moving from the source coordinate (starting from $(0, 0)$ as null program) to other program nodes. In each step, a weighted network of interactions between all the program nodes is formed in which the weights are the cost of traversing from a source coordinate to a destination program node. Unlike the spatial mode, these weights do not solely rely on the proximity between programs but alter as the source coordinate or the internal state of the system change.

Similar to the spatial mode, the program with the lowest traverse cost is then selected to execute prior to the others. If termination conditions are not met, the same process repeats. The position of the most recently executed program is then set to be the source coordinate to determine the next program to be executed. In Figure 5.1a, an overview of an SGP model is illustrated in its initial conditions. Each node represents a program and is labeled with the program name. Each program contains instructions that manipulate internal memory registers shared between all programs and outputs a single value corresponding to an internal register, an input, or a constant value. In step 1 (Figure 5.1b), the cost of traversing from $(0, 0)$ to every other node is calculated (details of which can be found in the next section). It is evident in the figure that the cost values calculated in each step do not only follow a proximity rule. For example, P1 is not the closest node to the starting point $(0, 0)$. However, since P1 has the lowest cost, it is selected for execution. In the next step (Figure 5.1c), P1 is the source point for calculating the costs to every other node, and therefore P0 is chosen for execution. The same principle continues until a termination condition is reached. A similar algorithm to the spatial mode, indicates how SGP selects the next program in line for execution. All of the individual programs are stored in a list. A loop on the program list is performed to calculate the cost of traveling to each program. Safeguards for protecting against infinite cost values and revisiting a node in case of loop-free configuration are in place to prevent invalid selections. The program with the lowest traverse cost is chosen to be executed.

In its programmatic mode, there are similarities between Tangled Programming Graphs (TPG) [Kelly and Heywood, 2017] and the SGP systems, as both evolve computer programs and the relationships between them in the form of a graph, with different input values resulting in different product graphs to be utilized. TPGs have been previously used for solving visual reinforcement learning problems, such as Atari games, and have produced results comparable to deep learning algorithms. TPGs are among the works closest to the idea of SGP in the literature, as they are constructed based on mechanisms that control the execution flow of programs until a terminal state is reached; however, there are key differences between the two systems. In SGP, the execution order is determined by minimizing a traversal cost value between the source program and every other program in the system. Furthermore, SGP supports iterative behaviors by allowing programs to execute more than once. In contrast, TPGs use a bidding system among teams of programs to determine the execution pathways. SGP is controlled by a 2D space, making the spatial properties of the nodes important for selecting the subsequent programs to execute. Finally, unlike TPGs, and more like conventional GP systems, SGP uses a population of mutually exclusive individuals.

5.3 The Cost Function

The cost function considers the spatial and internal states of the system to calculate the cost of traversing to a given program node based on a source coordinate.

$$cost = \frac{L_{target}}{L_{max}} + \frac{D(source, target)}{D_{max}} + R$$

In which R denotes the current value of the parameter set to be the output of the target LGP program. L denotes program length, L_{max} denotes the maximum program length allowed, $D()$ is an internal function that returns the Euclidean distance between two coordinates, and D_{max} denotes the maximum possible distance between two nodes of SGP. Each LGP program terminates with a return statement that outputs a numerical value for R. The current value of R cannot be pre-computed and at every step highly depends on

the previously executed programs and how the internal registers have been manipulated prior to the cost calculation step. To retrieve the different R values associated with various programs, it is not necessary to execute all of these programs in full. Instead, only the last statement of each program is executed. If a target program has the lowest traversal cost and is therefore chosen for full execution, then all of its statements are executed in order. The impact of program length and the distance between nodes are normalized; however, the value of R is not bounded to any range and depends on the problem inputs. This design decision might increase the impact of R on selecting the next program significantly; however, the cost function is configurable and can be modified to normalize the scale of R. It is prevalent for models evolved in this mode to take different execution routes with different sets of given inputs, forming dynamic solution graphs. This feature enables the opportunity to evolve localization in the system so that different sections of an SGP model respond to different sets of stimuli, a known characteristic of the brain in natural organisms.

5.4 Outputs, Termination Conditions and Model Execution

An imperative SGP system has four means of producing outputs. First is the numeric value returned by the last executed SGP program. Second, SGP also outputs the system's internal state, which is all the register values (initially set to 0) manipulated during the run-time of a model. Third, SGP operators are allowed to manipulate an external file, a computational object, or a third-party environment. Finally, it is possible to associate terminal programs with discrete actions or outputs. In other words, if a terminal program is reached, the action associated with that program is performed in the problem environment ending the individual execution. Depending on the model inputs, a different final program might get selected and thus produce a different action.

Similar to the spatial mode, multiple conditions can end the execution of a model. Each model program has a chance to be a terminal node and end the execution. Suppose a model

does not have any terminal program. In that case, a limit equal to the total number of programs in that model is set, breaking the execution if the count of executed programs exceeds that limit. Execution also ends if there are no more candidate programs or if the execution time exceeds a time threshold.

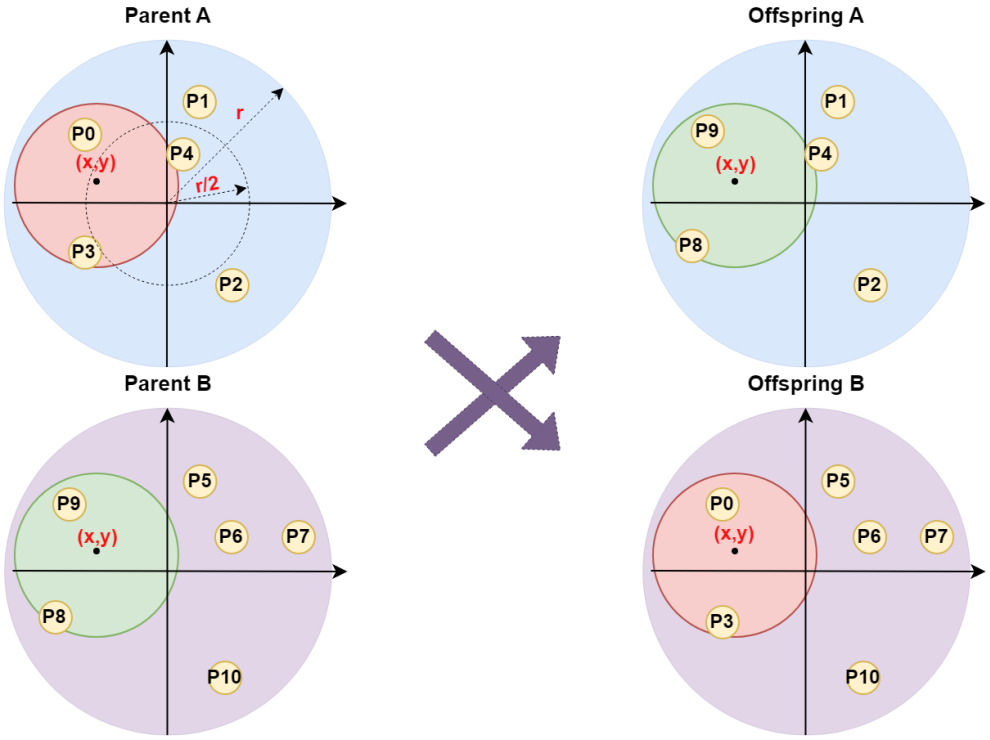


Figure 5.2: Crossover between two SGP models. Suppose (x, y) is the randomly chosen point within $\frac{r}{2}$ distance of the center (0, 0). Programs within $\frac{r}{2}$ distance of (x, y) form S_i of parent A (red circle) and parent B (green circle), and the rest of the programs form S_o (blue circle for parent A and purple circle for parent B). Offspring A is a combination of the programs in S_i of parent A and S_o of parent B and Offspring B is a combination of the programs in S_i of parent B and S_o of parent A

5.5 Evolution of Models and the Genetic Operators

Initially, a population of random SGP individual models is generated, in which the number of programs in each model, their length, and the initial coordination of the nodes within an allowed 2D space are randomly chosen. Next, an object pool of operators and operands is created from which the operator and operand(s) of each statement or instruction are

randomly selected. If there is no suitable operand for an operator, it will be removed from the selection pool. Operator and operand objects are reusable and therefore do not add to the computational cost of the system.

After evaluating models in each generation of the evolution, tournament selection is applied to the population. The two best competitor models are selected and have a chance to crossover to produce two offsprings or directly make it to the next generation of models after mutation. If the crossover happens, a mutation with a chance is also applied to the two new offsprings.

The 2D space of the SGP models is bounded by a radius parameter r, meaning that the program coordinates must be within r distance from $(0, 0)$. A random coordinate point within $\frac{r}{2}$ distance from $(0, 0)$ is selected to be utilized while performing crossover between two individual models. Let us denote the set of programs within $\frac{r}{2}$ distance from the randomly chosen point of an individual with S_i and the set of programs outside that radius with S_o. Then, in the crossover between parent A and B, every program in S_i of parent A and S_o of parent B form one offspring while the rest of the programs form the other offspring (Figure 5.2). There is no limit to the number of programs that are impacted by the crossover operator.

After crossover, there is a chance for every program of each individual to undergo mutation. SGP mutations can happen on a structural level, i.e., altering a program location or switching a program type (input program to output or vice versa), or on a statement level, i.e., altering the LGP programs. There are three types of structural mutations. 1) A program's coordination can change by performing a random walk with a fixed random step size. 2) The program type can alter from input to output or vice versa. 3) A program can be added or removed from/to the system. These modifications, along with an LGP mutation that targets the return value of the programs, are responsible for changing the behavior of how the programs will be selected for execution. There are three types of LGP mutations, which add statements to the program, delete a statement from the program or modify an existing statement if possible. By default, these mutations have an equal chance of occurring.

5.6 Conditional Return Statements

One of the abilities of SGP is to evolve rational pathways that change in response to the problem inputs. Conditional operators such as the basic *if* statements can help build a logic behind the return values of each program, forcing a different order of execution when different input values are given to the system. By default, however, SGP requires each program to have a final single return statement that cannot be connected to any other operators, such as being tied to a conditional *if* statement. Since allowing evolution to use a combination of conditional operators and internal state values to evolve conditional pathways is not trivial, we came up with the idea of replacing the normal return statements of the program with a custom conditional operator called RetCon (stands for Return Conditions). This operator forces a condition on the return statement in a way that if the condition is true, an internal state value or a constant value will be returned. Otherwise, another return value will be selected. The two return values could be the same.

5.7 Experiments and Results

In this section, we apply SGP to a set of problems classified into two case studies to analyze the behavior of the system by comparing different modes of otherwise identical SGP setups with classical TGP and LGP. The TGP included in the DEAP framework [Fortin et al., 2012] and the same LGP system used for the SGP programs were used to conduct the experiments. The use of RetCon in SGP facilitates the evolution of conditional pathways, making SGP models well-suited for addressing problems that require decision-making. The specific problem set for each case study was selected due to the presence of a decision-making component. The experimental setup used for solving the problems of this chapter is as described in table 5.1.

Parameter	Description	Value
g	Number of evolutionary generations	1000
$Size_{pop}$	Population size of the experiment	100
$Size_{tournament}$	Size of the tournament selection pool	5
e	Number of elites that are directly selected to be a part of the next generation	1
$m_{spatial}$	Spatial mutation rate	30%
m_{LGP}	LGP mutation rate	60%
C	Crossover rate	100%
$cost_i$	Cost function	$cost = \frac{L_{target}}{L_{max}} + \frac{D(source,target)}{D_{max}} + R$
$Topology$	Describes any topological limitation for placing nodes on the 2D space	circle
$Type_{output}$	Describes the output type of the system	20%
$Count_{evaluation}$	Number of evaluation of an individual model	10+
$Count_{registers}$	Number of internal system registers	inputs + 2
$initSize_{max}$	Maximum number of programs per individual during initialization	5
$Size_{max}$	Maximum number of programs per individual	10
$lgpInitSize_{max}$	Maximum number of statements per program during initialization	3
$lgpSize_{max}$	Maximum number of statements per program	8
T	Maximum individual evaluation time	200ms
$radius$	Size of the topology representing the 2D space	20
Set_{OP}	Operator or function set	Problem-dependent
$Constants$	Set of constant values	1, 2, 3
$RevisitPenalty$	Penalty for revisiting a node when loop is allowed	0.03
$reps$	Number of replicates for each experiment	50

Table 5.1: Parameter setup of SGP. Relevant parameters for LGP and TGP are chosen similarly

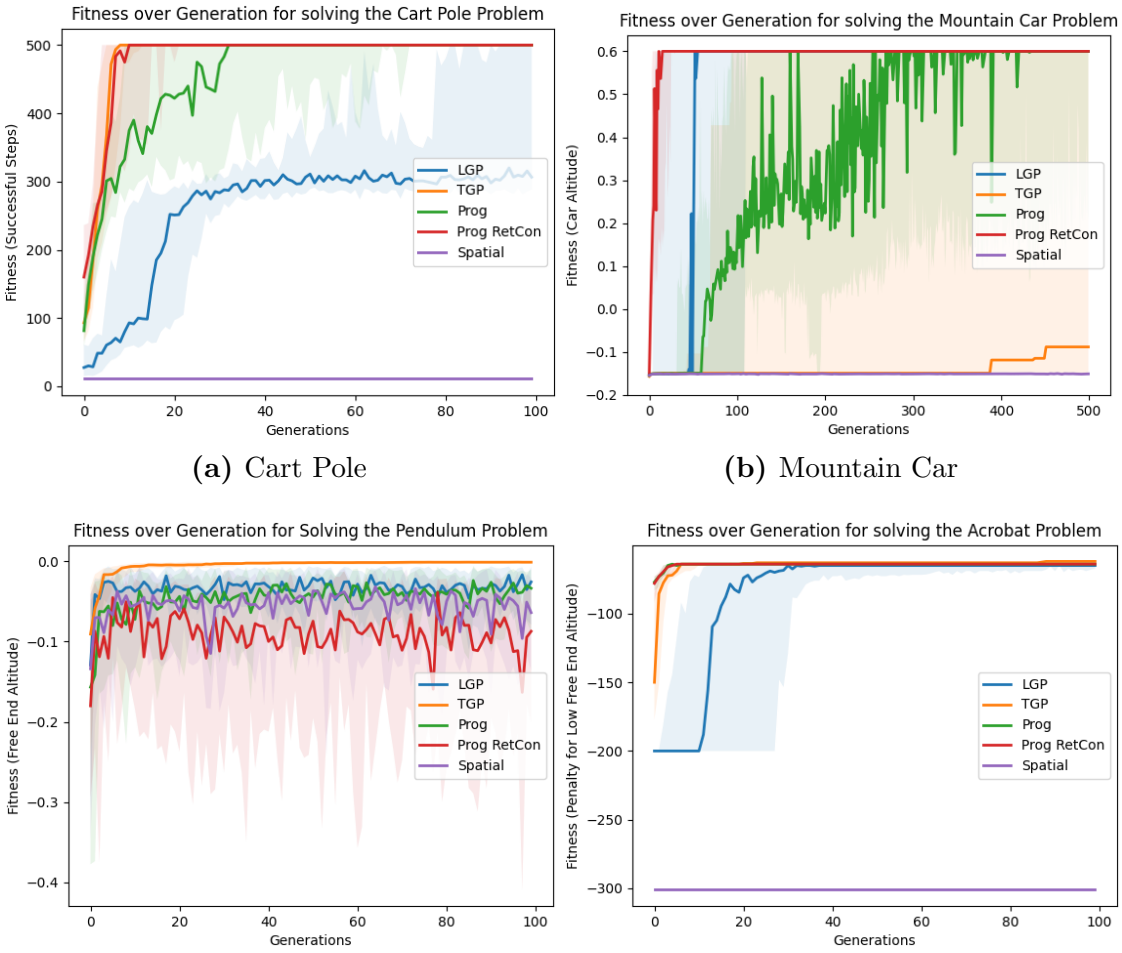
5.8 Results: Classic Control Problems

OpenAI Gym [Brockman et al., 2016] is a library of Reinforcement Learning problems in Python which helps with the development and comparison of problem-solving algorithms by providing a straightforward environment-to-algorithm API. In particular, we tackled Cart Pole, Mountain Car, Pendulum and the Acrobat problems from the Gym library which are detailed as follows:

- **The Cart Pole Problem:** This is a version of the cart pole problem introduced in [Barto et al., 1983]. In this problem, a pole is attached to a moving cart on a 2D track through a non-actuated joint. The goal of this problem is to prevent the pole from falling by accelerating the cart pole to the left and right (Figure 5.3a).
- **The Mountain Car Problem:** In this problem, a car is randomly placed at the bottom of a sinusoidal valley. The goal is to reach the end state (shown as a flag in Figure 5.3b) by accelerating toward the left and right to build the momentum to cross the valley. The description of this problem first appeared in the thesis of Andrew Moore [Moore, 1990].
- **The Pendulum Problem:** This is a famous problem in control theory in which a reverse pendulum is attached to a fixed point, and the goal is to torque the pendulum so it faces upwards in a way that its center of gravity is directly above the attached point (Figure 5.3c).
- **The Acrobat Problem:** In the acrobat problem, two links are attached, forming a chain while one end is free and the other is attached to a fixed point. The goal is to apply torque to the chain so it swings, and the free end reaches a certain height (shown as a line in Figure 5.3d). This problem is introduced in [Sutton, 1995].

(a) Cart Pole **(b)** Mountain Car **(c)** Pendulum **(d)** Acrobat

Figure 5.3: Four different classic control problems tackled in this chapter



(a) Cart Pole **(b)** Mountain Car

(c) Pendulum **(d)** Acrobat

Figure 5.4: The fitness over generations plot for solving the four classic control problems. a) Fitness equals to the number of steps in which the pole is held in an upright position. b) Fitness represents the car altitude at the end of each evaluation. c) Fitness equals to the altitude of the free end of the Pendulum at the end of each evaluation. d) Fitness indicates a -1 penalty for each step in which the free end has not passed the threshold line

Figure 5.4 shows the results for tackling the OpenAI Gym classic control problems. 50 replicate experiments with different random seed values were conducted for each of the four problems. The median fitness values over generations for the best-evolved models of each replicate are illustrated. The shaded areas represent the 25 and the 75 quantiles, while the solid lines represent the median. Three configurations of SGP are tested against these problems and are compared with classical TGP and LGP. *Prog* refers to the programmatical mode of the system; *Prog RetCon* indicates the usage of conditional return statements in the

programmatical mode, and *Spatial* refers to the spatial mode. In the spatial mode, the usage of RetCon operators does not make a difference since the return statements of the programs do not change the execution order. All of the experiments are run for 1000 generations; however, depending on the problem, after a certain number of generations, the fitness values cease to change, and therefore, a portion of the generations are selected to ease the analysis of the results. Finally, even though all the classic control problems are deterministic, the starting conditions are slightly randomized (e.g., the position of the car in the mountain car problem) to help the problem solvers find a generalized solution. For all of the experiments, *if*, *assign*, and basic math operators are used as the function/operator set of LGP and SGP.

To solve the Cart Pole problem, SGP is configured to use discrete outputs in which each individual must consist of two terminal nodes, each associated with an action of either accelerating the cart towards left or right. 5.4a shows the results for solving the Cart Pole problem. SGP with RetCon and TGP solve this problem in less than 20 generations. Programmatical settings without RetCon also solve the problem. However, it takes more generations to solve, and the shaded green area shows that it takes more time for all the individuals in all the replicates to be able to solve this problem while all the individuals of the replicates for the RetCon settings solve the problem in less than 30 generations. LGP also solves the problem but its performance is not as good as the rest of the approaches The spatial setting fails to solve the task over all generations since, in this setting, the network inputs do not change the execution order of the graph. In other words, the same discrete action is always taken; therefore, constant fitness is achieved over generations.

Same as the Cart Pole problem, for the Mountain Car problem, SGP is configured to use discrete outputs. As illustrated in Figure 5.4b the RetCon outperforms the other two configurations by solving the problem for all the replicates in less than 50 generations. LGP performs slightly worse, solving the problem in approximately 60 generations. The programmatical setting without RetCon has a cold start, but the replicates mostly solve the problem at around 450 generations. However, the difference between the fitness of the best

models among all the replicates varies greatly. The shaded orange area shows that there are individuals in the TGP approach that solve the problem but the median results are worse than the other approaches in 500 generations. Once again, the spatial mode fails to solve the problem while producing a constant fitness.

The nature of the Pendulum problem is slightly different from the other problems since it requires a continuous output indicating the amount of torque applied. Unlike the other three problems, the spatial configuration performs comparable to the other approaches. TGP outperforms all other approaches; however as shown in Figure 5.4c fitness values of approximately 10^{-4} were achieved by the best individuals of all the approaches showing almost an upright position of the pendulum. The high fluctuation of the median line is due to the high impact of the random starting position of the pendulum on the outcome of the evaluation.

The final classic control problem tackled in this chapter is the Acrobat problem. As illustrated in Figure 5.4d, SGP manages to solve the problem in both programmatical modes with or without RetCon in less than 5 generations and improves its performance until 10 generations managing to reach the specified line in all of the best models in about 60 steps. Like the other discrete output problems, the spatial mode drastically fails by only producing a constant output. TGP has a slightly worse performance while LGP solves the problem in 40 generations.

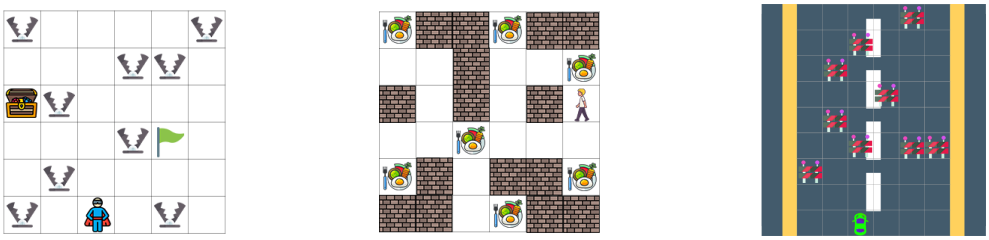
5.9 Results: Custom Toy Problems

The custom Toy Problems is a custom library of three Reinforcement Learning problems included with the SGP source code that can be briefly described as the following (Figure 5.5):

- **The Adventure Problem:** This problem is inspired by an Atari 2600 game called *Adventure* [Chance, 2022]
- **The Foraging Problem:** This is a famous classic Artificial Life problem in which an

agent has to gather all the food spread in a 2D grid. The tiles are often blocked by obstacles or walls (Figure 5.5b).

- **The Obstacle Avoidance Problem:** As illustrated in Figure 5.5c a car agent is driving on a road that is occasionally blocked by randomly appearing roadblocks. The car agent has to avoid hitting the roadblocks for a specified number of time steps.



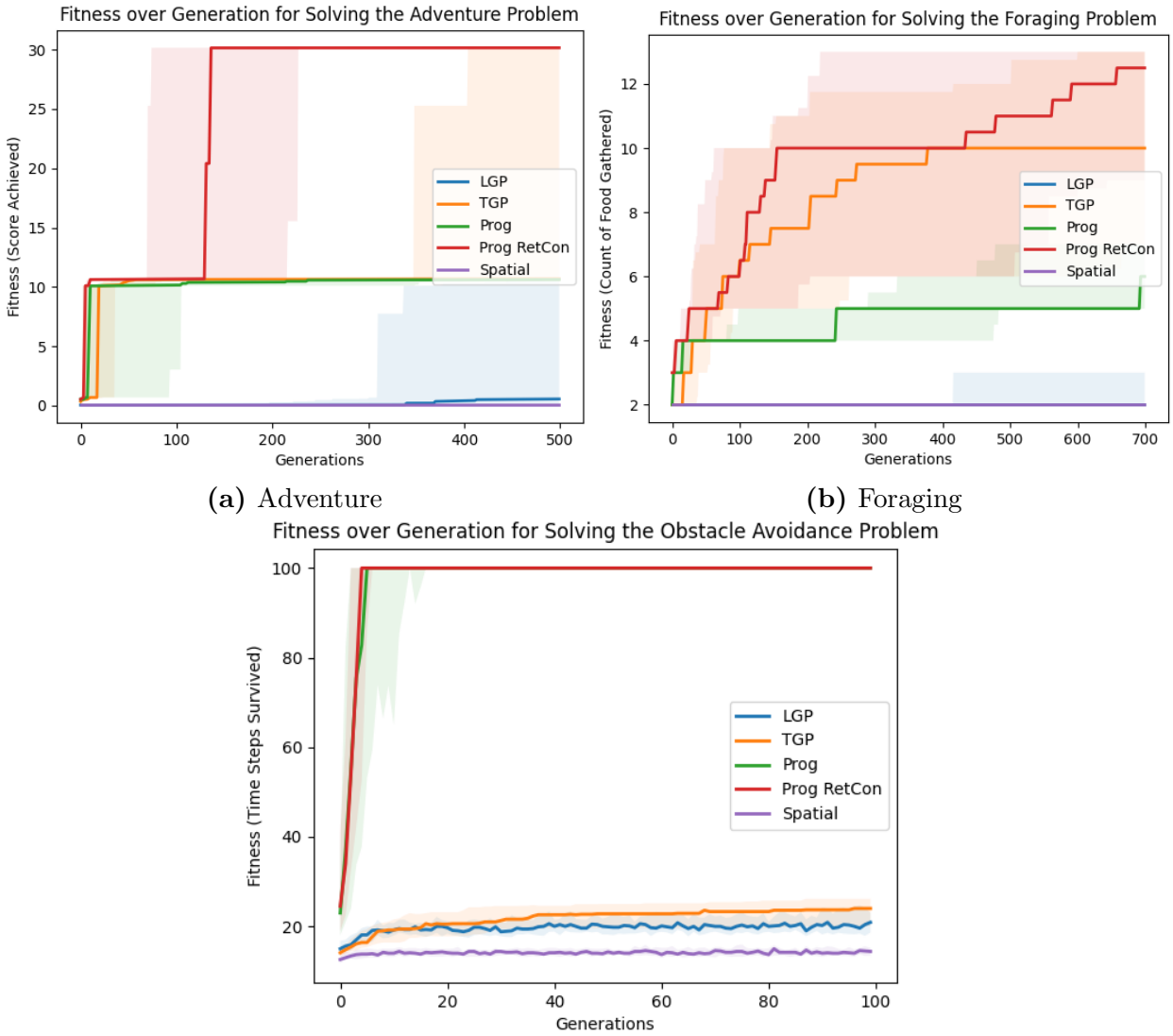
(a) Adventure **(b)** Foraging **(c)** Obstacle Avoidance

Figure 5.5: Three different toy problems. *Icons used in the images are from: flaticon.com*

Figure 5.6 depicts the results produced for solving the three Toy Problems for 50 replicates. The only non-deterministic problem is Obstacle Avoidance since the roadblocks spawn randomly.

In the adventure problem, the observation consists of 6 integer inputs corresponding to the agent's vision cone and a single bit corresponding to whether the agent has picked the treasure or not. The agent's vision cone shows two three-tile rows in front of the agent. The problem's action space consists of three discrete actions: moving one tile ahead, turning left, and turning right. All the entities in the problem grid and the empty tiles are coded with unique integer values and are visible to the agent. The agent can move to the treasure tile to automatically pick up the treasure. A small reward of 0.01 is given to the agents that move. The computational models are responsible for giving an agent instructions to solve the task through actions, and the individual's fitness equals the score the controlled agent achieves. A significant score of 10 is given to the agents that manage to grab the treasure, and a very significant score of 20 is given to the agents that reach the final destination while carrying the treasure. The simulation ends after 100 time steps or when the agent reaches the final destination or falls into a trap. The score is returned to the SGP evolver module as

the controlling model's fitness value. Figure 5.6a depicts the results produced for solving the Adventure problem over 500 generations.



(a) Adventure **(b)** Foraging

(c) Obstacle Avoidance

Figure 5.6: The fitness over generations plot for solving the three toy problems. a) Fitness indicates the score of the agent at the end of evaluation. b) Fitness is equal to the number of food gathered by the agent c) Fitness is the total number of time steps that the agent has survived the environment

As expected, the spatial configuration fails to solve a task with discrete output. The programmatical settings manage to evolve agents capable of picking up the treasure; however, they fail to reach the final destination. The RetCon settings, however, solve the problem

entirely in less than 250 generations. TGP slightly outperforms the Prog settings since in later generations, the best TGP individuals fully solve the problem. LGP on the other hand, only manages to find the treasure in the later generations but fails to completely solve the problem and the median line always stays low.

The Foraging problem has an observation space consisting of 6 inputs corresponding to the agent's vision cone. Like the adventure problem, the vision cone includes the two three-tile rows in front of the agent. The action space of the problem is the same as the Adventure problem consisting of three actions: moving, turning left, and right. A total of 20 food tiles are available for the agents to take, which are often placed at the end of a maze-like pattern in which the agent will have to return to the path taken to reach the food to get out (e.g., top left food in Figure 5.5b). The simulation is run for 200 time steps while no reward is considered for moving. Compared to other problems, this is a more challenging task to solve since the maze-like patterns make it quite difficult to gather all the food in the allowed time steps. Figure 5.6b shows the result for solving this problem. Same as most cases, the Programmatical settings with RetCon outperforms the other two modes while being able to gather as much as 14 food items at best among all the replicates. The changes in the median line seem to show evolution after 700 generations. Perhaps, running this task for a more extended period would help the system to completely solve the problem. The programmatical SGP needs substantially more time to evolve conditional logic to solve these types of problems only using a basic *if* statement, and the spatial mode fails to solve the task. The performance of TGP on this scenario is slightly worse than the RetCon settings while LGP only manages to perform better than the spatial mode after approximately 400 generations.

The observation space in the obstacle avoidance problem consists of 12 integer values corresponding to the vision cone of the car agent. This vision cone includes four three-tile rows in front of the agent. The action space of the problem is three discrete actions: moving left and right and doing nothing. To achieve a perfect score, the car agent must avoid all the

roadblocks for 100 time steps. The number of time steps before the car agent hits a roadblock is the fitness of the controlling SGP model. Figure 5.6c shows the results produced for solving this task. Both SGP programmatical configurations with or without RetCon manage to avoid all the obstacles in less than 20 generations. At the same time, it takes a bit longer for all the replicates to completely solve the problem for the setting without RetCon. The fluctuations in the case of spatial mode are due to the randomness of the roadblock patterns selected to appear in the far front of the car. TGP and LGP do not achieve good fitness levels on this problem but outperform the spatial mode.

5.10 Results: Impact of a Spatial Crossover on the Evolution of Programs

To check whether the mechanisms in the system impact the spatial properties of the SGP models, an experiment was conducted with a different crossover algorithm called the Spatial Crossover. This crossover is quite similar to the normal crossover used in SGP however, instead of choosing a random circular area to form S_i, programs that are located in the top right quadrant of the 2D space (x-coord and y-coord greater than or equal to 0) are selected to form S_i of the parent individuals. In this approach, always the same spatial portion of the individuals swap to form offspring. We tracked the position of all the individuals' programs (not just the best) in all the 50 replicates. The results are summarized in Table 5.2. SC stands for Spatial Crossover, NC stands for Normal Crossover and P1 and P2 refer to two arbitrary test problems. The 2D space of each individual is divided into four quadrants starting from the top right (Q1) and going clockwise to the top left (Q4). Results show that in the case of using the Spatial Crossover, programs tend to move out of the Q1 area in which the crossover is happening. This behavior is reflected by the significantly lower percentage of appearance of programs in Q1 compared to the case where the normal crossover is being applied. The employed spatial crossover swaps full LGP program nodes to or from a SGP model. While this method adds significant diversity early in the evolution, it can become destructive later,

as it is agnostic to the logic behind determining the execution flow of each model. The implication of this experiment is that nodes tend to move away from the crossover region.

Quadrant	RetCon P1 (NC)	RetCon P1 (SC)	RetCon P2 (NC)	RetCon P2 (SC)	Spatial P1 (NC)	Spatial P1 (SC)	Spatial P2 (NC)	Spatial P2 (SC)
Q1	25.69%	**12.63%**	26.23%	**10.98%**	27.51%	**8.98%**	34.17%	**6.8%**
Q2	22.09%	30.21%	22.84%	27.07%	21.68%	29.78%	26.48%	28.58%
Q3	27.87%	27.72%	23.79%	29.32%	21.66%	30.38%	17.68%	33.52%
Q4	24.34%	29.64%	27.14%	32.63%	29.14%	30.87%	21.67%	31.1%
Total	38310	40030	27.14%	38493	38566	41742	38059	41597

Table 5.2: Position of the final programs of all individuals in the latest generation using Normal Crossover (NC) and Spatial Crossover (SC) for two test problems

5.11 Results: Tackling the Santa Fe Ant Problem

Revisiting the inclusion of loops in SGP and examining localization at both the LGP and SGP levels, we tackled a variation of the classic Santa Fe Ant Problem, also referred to as the Artificial Ant problem. The common approach for addressing this challenge involves evolving a routine that directs the ant agent to maximize the number of food pieces consumed within a specified number of time steps. In contrast, our approach treated the problem as an external environment. We tasked SGP with not only evolving a routine but also formulating an iterative structure to execute the evolved procedure. SGP controls an ant agent capable of navigating a 2D grid, even though it lacks any knowledge about the food's location until it employs an action to detect the presence of food in a single grid cell ahead. The problem's objective is for the ant agent to maximize the amount of consumed food, which is scattered in the form of multiple separate trails.

The problem's action space comprises discrete actions: "move," "left," "right," and "sense." These actions are provided to SGP as callable operators that engage with the problem's environment. Essentially, when these methods are invoked, they either alter the state of the ant agent within the 2D space or probe the surroundings for available food in front of the ant, returning relevant sensory data.

In the experimental setup, SGP was equipped with basic mathematical operators, the discrete actions previously described, an "if" operator, and an "assign" operator. The output from the "sense" method could be directly employed as the condition in the "if" operator.

We carried out experiments using four configurations of SGP: programmatical with RetCon settings, either with or without loops, and a Spatial configuration, also available with or without loops.

For the problem addressed, evolution can produce conditions that detect food in front of the ant agent and, based on the result, prompt the executing program to return a specific numerical value. As the cost function is influenced by the return values of the programs, RetCons facilitate the evolution of conditional execution orders, simplifying finding fitter models for solving decision-making problems.

Each individual was restricted to a maximum of 10 programs. SGP had a limit of 600 actions before the problem was terminated, and a fitness score equivalent to the number of food pieces consumed by the ant agent was computed. Figure 5.7 showcases the results for addressing the Santa-Fe Ant variation. To solve this problem, the SGP models had to develop a routine enabling the ant agent to strategically move to gather the food on each trail while simultaneously evolving an iterative structure in charge of repeatedly executing the routine. The RetCon, Spatial, and Spatial Loop configurations exhibited similar performances, with the Spatial Loop setup marginally surpassing the other two in the earlier generations. Conversely, the RetCon Loop configuration consistently outperformed the other setups across the generations, efficiently utilizing its iterative structure to gather 54 of the available 80 food items in a concise representation.

The smallest SGP model, consisting of just four programs, provides an intriguing study case for the Santa Fe Ant problem. While it may not have achieved the highest score in terms of collected food, its streamlined structure offers valuable insights. Notably, there's evidence of program localization, where distinct programs specialize in specific tasks. The primary aim of our current assessment is to delve deeper into the workings and efficiency of this concise model.

The illustrated programs in Figure 5.8 provide a view of the chosen individual's operational functions. Upon inspecting the evolved programs p0 and p1, it becomes clear that they are

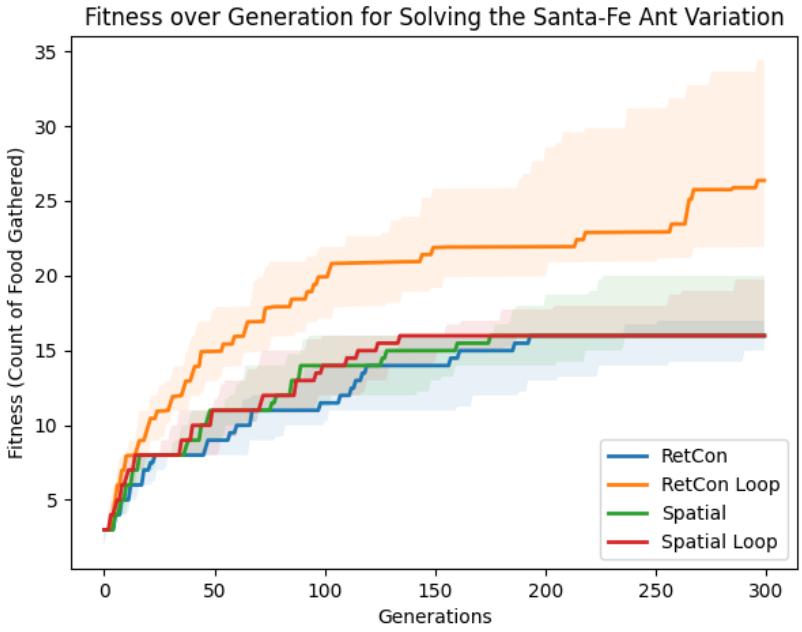


Figure 5.7: The fitness over generations plot for the Santa-Fe Ant variation. The plots represent the median performance of the top models over 50 repetitions of each experiment across various evolutionary generations. Four distinct SGP configurations were employed for this problem. "RetCon" indicates the Programmatical mode using the RetCon operator. "RetCon Loop" is akin to RetCon but permits the repetition of the same program multiple times. "Spatial" denotes the Spatial mode, while "Spatial Loop" is the Spatial mode that incorporates loops

comprised of function calls directing the ant agent to progress straight and turn either right or left, respectively. When deployed in a precise sequence, these programs enable the ant to traverse the grid, ensuring that it does not retrace its steps and effectively covering multiple directions.

This evolved individual demonstrates an interesting iterative pattern, depicted in Figure 5.9. It initiates with p0 from the starting coordinates (0, 0), transitioning to pattern a as shown in Figure 5.9a. Here, the execution bounces between P0 and P1, directing the ant to cover the grid, collect any discovered food, and make a turn before each program's termination.

Subsequently, pattern b (visible in Figure 5.9b) takes over, with p3 diverting from p0. This leads to a repetitive interplay between p0 and p1. After several such iterations, the

```
def p0(self):
    global simulator, a0, a1, a2, a3
    move(simulator)
    move(simulator)
    move(simulator)
    move(simulator)
    right(simulator)
    move(simulator)
    if a2 >= a0:
        return a3
    else:
        return a2
```

(a) P0

```
def p1(self):
    global simulator, a0, a1, a2, a3
    a2 = 1
    move(simulator)
    move(simulator)
    move(simulator)
    left(simulator)
    a3 = a2 - 3
    if a2 > 3:
        return 1
    else:
        return a0
```

(b) P1

```
def p2(self):
    global simulator, a0, a1, a2, a3
    a2 = op_div(a1, 2)
    if 1 <= 2:
        return 3
    else:
        return a3
```

(c) P2

```
def p3(self):
    global simulator, a0, a1, a2, a3
    move(simulator)
    if 1 >= a3:
        return 1
    else:
        return a2
```

(d) P3

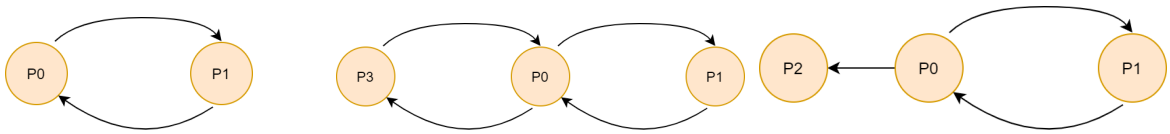
Figure 5.8: LGP programs associated with the SGP nodes of the studied individual model. *simulator* is an object controlling the problem environment. Four internal register variables (a0 to a3) are shared between the programs. *op_div* is a protected division operator which returns 1 if division by zero occurs. P2 is the only terminal program

pointer loops back to p3, causing a repeat of this same pattern.

The concluding pattern, *c*, is displayed in Figure 5.9c. This pattern surfaces when the model's other maneuvers have been exhausted, often triggered by a penalty for revisiting the same grid sections. After several back-and-forths between p0 and p1, the system finally settles on the terminal program, p2.

This intricate substitutions of patterns underscores the model's adaptive power to maneuver based on the environment's feedback and the problem's inherent goals. Notably, the clear differentiation in the roles of p0 and p1 with regards to their task or behavior highlights a manifestation of modularity at the LGP level, where the two programs distinctively manage

straight movements and directional shifts.



(a) Iterative Pattern 1 **(b)** Iterative Pattern 2 **(c)** Termination Pattern

Figure 5.9: Iterative patterns evolved for the selected individual

5.12 Conclusion

This chapter introduced a new mode of SGP, a GP paradigm which accounts for the dimension of space as a first-order effect to optimize. SGP can work in two modes of spatial and programmatical, which bring unique characteristics to the system, allowing it to evolve static and dynamic graphs, respectively. The impact of these two operation modes was tested against two classes of problems while introducing conditional return statements. SGP was tested against four classic control problems of OpenAI Gym library. RetCon's ability to quickly evolve conditional statements to choose the right pathway of the graph by manipulating the weights of the underlying regulatory network was shown during these experiments. For all the cases except the Pendulum problem, RetCon quickly solved the control tasks. The Pendulum problem required less decision-making and more accuracy on the produced continuous outputs (amount of torque). The programmatical mode without RetCon was able to solve the control problems as well. However, it takes more time for evolution to evolve the factual conditional statements in the LGP programs to reflect the same decision-making structures. The spatial mode fails in producing discrete outputs while showing promise in the Pendulum problem that requires continuous outputs. This is because of the ability of the spatial mode to refrain from using too many conditional statements and rely more on the power of LGP to produce continuous outputs. SGP was compared to two other approaches of TGP and LGP. Except for the Pendulum task which had a more continuous nature, SGP outperformed the other two approaches in all cases.

Three custom Toy Problems were introduced in this chapter, on which SGP was tested. These problems had a larger observation space compared to the classic control problems. Comparing the three tested configurations, SGP produced a similar result to the control problems, with RetCon outperforming the two other configurations. The more complex observation space did not significantly impact the performance of SGP showing better performance than TGP and LGP. The Foraging problem was not completely solved; however, improvements in the fitness values showed the possibility of solving this problem if run for an extended period.

A shortcoming of SGP is not having enough control to create a balance in evolving structural elements and LGP programs simultaneously. Perhaps, the utilization of parallel island models that decouple focusing on the evolution of the structural elements and the SGP programs from the main population while interacting with it now and then could be helpful to achieve better results. Furthermore, a method for optimizing the system hyper-parameters during evolution could also be among the future directions of this work. As of now, the cost function used in the system adds the normalized distance and length to the return value of the SGP programs. This reduces the impact of distance and length compared to the possible return values. Perhaps a different cost function can result in more exciting results. Finally, to show the effectiveness of SGP, it is necessary to apply it to more realistic and complex problems and to compare it with state of the art problem solvers.

Chapter 6

Summary and Discussion

6.1 Introduction

In the final chapter, I summarize the content of each chapter of the thesis, discuss the significance of the work performed for each chapter, and discuss the future possible research opportunities. Finally, I conclude this thesis by the points I realized throughout conducting my experiments and research on the subject.

6.2 SGP Framework for Studying Space in Evolution

Chapter 2 provides a comprehensive overview of Spatial Genetic Programming (SGP) as tool for studying speciality in GP. SGP is a tool that leverages the addition of spatial properties to its individuals to enhance its problem-solving capabilities. This chapter begins with a discussion on the importance of spatial considerations in computational regimes and how space is often abstracted away in computational regimes. It proceeds to introduce the core components and mechanisms of the SGP algorithm, including an introduction to the mutational operators, the role of LGP programs within the system, and how the system evolves its individuals. The flexibility of SGP is highlighted by its ability to incorporate new operators for addressing specific problems. The translator component in SGP, which can

convert SGP models into Python scripts, increases the transparency and understand-ability of the evolved individuals.

Additionally, the chapter delves into the configuration parameters of the SGP system, while explaining each setting's role in tailoring the system for solving various computational tasks. It also outlines a suite of benchmark problems designed to assess the performance of different SGP or LGP setups. The chapter sets the stage for subsequent discussions on the experiments conducted using SGP across different problem classes.

This chapter advances the field of GP by integrating spatial structure into the evolutionary process, which is a relatively unexplored dimension in the field. Such investigations could result in the emergence of more robust and efficient problem-solver algorithms. By embedding spatial constraints and relationships within the SGP paradigm, SGP provides a novel approach to problem-solving that can potentially lead to solutions that are not only innovative but also more reflective of complex real-world systems where spatial relationships are key. Furthermore, the detailed configuration parameters and the inclusion of a diverse set of benchmark problems demonstrate the flexibility and potential applicability of SGP across various domains. The ability to translate evolved models into Python scripts enhances the practicality of the system, allowing for easier integration with existing software and further analysis. This research could pave the way for new applications in areas such as control systems, and game strategy development, where the spatial aspects of a problem are integral. Moreover, the contribution of this chapter can serve as a foundation for future studies that may investigate the impact of spatiality on evolutionary algorithms.

Accounting for a fundamental dimension such as space in the GP individuals is not a trivial task and requires enormous amount of implementation and consideration for design decisions hindering the research. This makes existence of tools which can accelerate the research process and remove the need for unnecessary implementations extremely important. Such scenarios are evident when witnessing how frameworks such as PyTorch [Paszke et al., 2019], TensorFlow [Abadi et al., 2016] or scikit-learn [Pedregosa et al., 2011] have been

used frequently in the literature. The SGP tool presented in this dissertation provides researchers with a modular framework in which each module can be replaced with newly implemented custom modules which in turn change the behavior of the underlying algorithm accordingly. The implementation of SGP is meant to be simple and self-explanatory and introduces a lot of research opportunities. The SGP tool is available online for free at: [https://github.com/elemenohpi/SpatialGP].

Given the variety of parameters and settings available within SGP, auto-tuning such variables to conduct experiments could be conducted to optimize these settings for various classes of problems. The power of SGP is highly dependant on the problem solver that is utilized as its nodes. There is potential to extend the research to hybridize SGP with other evolutionary algorithms or machine learning methods to create more powerful problem-solving frameworks.

Another promising direction is the application of SGP in dynamic environments where the evolutionary process could benefit from real-time adaptation to changing spatial constraints. This would require establishment of a link between the spatial properties of the programs within an individual and the spatial components of the environment. In addition, investigating the scalability of SGP to tackle large-scale problems and integrating multi-objective optimization criteria could provide valuable information. Finally, enhancing the usability of SGP through improved user interfaces and visualization tools would make it more accessible to researchers and practitioners across various scientific and engineering disciplines. Specifically, the SGP algorithm takes a step towards a more biologically-close algorithm. Ease of access in SGP could potentially help biologists to use this system to better understand and study the phenomena occurring in nature in which space plays an integral role.

6.3 Impact of Space in the Evolution of the SGP individuals

Chapter 3 presents an exploration into the effects of spatial components on the evolution and efficiency of SGP. An interesting observation in the chapter was how SGP programs utilize spatial proximity as a unique method for evolving iterative structures. It was observed that SGP systems inherently tend towards spatial organization, with a pronounced clustering of programs as the evolution progresses. The study also noted that spatial aspects impacted the diversity of the population, with LGP exhibiting greater fitness diversity, albeit not in structural diversity, which was more influenced by SGP's spatial operators. Various spatial topologies such as Circle, Lattice, Line, and Ring were tested, revealing that certain topologies could influence the evolutionary path and quality of solutions, with some performing better than others depending on the problem at hand.

The introduction of spatial evolutionary operators like spatial crossover and region-specific mutation rates added new dynamics, but didn't substantially enhance performance in the tested scenarios. Furthermore, extending the dimension of SGP into a 3D space, while the intention was to mirror the complexity of biological systems more closely and to increase control, did not improve outcomes over the 2D model. Overall, the chapter concludes that while spatial structures add complexity to GP, they also open up possibilities for improved diversity and analysis of the system.

By investigating the impact of spatial structures on the evolution of SGP programs, the study advances our understanding of how spatial elements can be strategically leveraged to enhance the diversity and performance of GP models. This is particularly important because the SGP model in this case, mimics the dynamics of natural evolution, where spatial distribution and diverse environmental conditions play critical roles in the development and adaptation of organisms. Generally, evolving iterative structures in EAs is a cumbersome task since it allows the necessary computations of the models to exponentially grow. The

proposed method in this chapter for evolving iterative structures solves many challenges in this area of the field.

The findings of this chapter highlights the potential of SGP to adapt to different spatial topologies, suggesting that the physical arrangement of individuals within a population can impact the evolutionary outcomes. While the extension to 3D space did not yield significant performance improvements, this line of study may inspire further research into multi-dimensional genetic programming frameworks. While measuring fitness diversity in this chapter, it was found that SGP executes less number of statements compared to LGP while the same maximum size was allocated for either systems' individuals. It would be interesting to perform the same fitness diversity measurement by repeating the experiment while assuring that the average number of executed statements between the two systems are on similar levels. Furthermore, maintaining a healthy amount of diversity in the population of an EA, allows the individual models to escape local optima to find fitter solutions. A very unique feature of SGP that becomes possible via the 2D space representing individuals, is how it is possible to indicate regional mutation rates. In theory, it is possible to utilize a combination of regional mutations rates and repositioning of SGP programs to assist with valley-crossing and to avoid premature convergence of individuals.

Building on the insights gained from this chapter, future research could explore several promising avenues. One immediate step could be to investigate the impact of spatial evolutionary operators in more depth, particularly in how they affect populations over longer evolutionary timescales and in more complex problem environments. Since the 3D spatial setup did not show marked improvements in this study, a deeper analysis of how dimensionality interacts with problem complexity could be insightful, potentially identifying scenarios where additional dimensions do yield benefits specially in cases that SGP programs need to learn properties from a 3D space to solve a target problem.

SGP can facilitate the research in areas such as neuro-evolution if neural networks are combined with it as the programs that control the outcome of the system. Additionally,

applying these spatially structured GP models to real-world problems, such as those in bioinformatics, network design, and artificial life simulations, could provide concrete benchmarks for their effectiveness and lead to practical applications of the research. A finding that was observed in this chapter indicated that SGP excels in solving loop problems that are consisted of a single high-degree term or can be simplified as such. It would be interesting to design more sophisticated evaluation metrics that could guide the evolution towards solving more instances of SR problems that need iteration. A possible research avenue that was briefly mentioned in this dissertation but not studied was performing a spatial analysis on population level rather than individual level to specify traits that evolve in a population of individuals. Such study could be potentially fruitful towards better understanding how speciation occurs in such populations through spatial visualization.

Space intuitively aids in visualization. For example, elements within a space can be easily depicted based on their spatial positions. Furthermore, as mentioned previously in this dissertation, it is more trivial to classify spatial elements by repositioning them in space. However, a very important characteristic of space that comes with elements being spread in different spatial positions is how it can make parallelization intuitive. This idea can be clearly observed in natural systems or even in computational ones, such as how distributed systems operate while each component of such systems is located in a different spatial position. This further increases the potential of utilizing island models in SGP, targeting different aspects of the algorithm on each island. The same principle can be applied to create a parallelized version of SGP in which programs execute concurrently in a temporospatial manner.

6.4 Solving Symbolic Regression Problems with SGP

Chapter 4 details an examination of SGP for solving Symbolic Regression problems, typically addressed by GP algorithms. The investigation benchmarks SGP against LGP and TGP through a series of 92 equations sourced from the Feynman lectures on Physics. The outcomes of this comparative analysis are telling that SGP successfully solves 52 of these equations,

closely trailing the 54 solved by LGP and significantly outperforming the 22 addressed by TGP. The adoption of the Pearson correlation coefficient as the fitness measure is a important to achieve such results in the study. The chapter concludes by confirming SGP's potential in solving SR problems, given its close performance to LGP and superior results over TGP.

The significance of these findings is in the potential refinement of SGP and its application across a wider array of problems. The insights contribute to the domain of spatial evolutionary computation, suggesting a robust methodology for solving regression problems and highlighting areas for future research, such as the optimization of SGP parameters and the exploration of other evolutionary algorithms within the framework.

This chapter represents a significant contribution to the field of evolutionary computation, specifically in the context of SR problems. By investigating the application of SGP, it extends the understanding of how different GP paradigms perform when tasked with finding symbolic representations for given data. The detailed comparison against LGP and TGP using a robust dataset from the Feynman lectures provides empirical evidence of SGP's capabilities. Notably, SGP's comparable results with LGP and its clear advantage over TGP in solving a majority of the test equations demonstrates its potential as a powerful tool for SR problems. This is especially significant in light of the complexity and diversity of the problems tackled, which are representative of real-world physics and mathematical challenges.

The chapter suggests several future directions for research in the field of evolutionary computation, particularly with the application of SGP to symbolic regression problems. One area of future investigation is the optimization of SGP parameters, which could involve fine-tuning the algorithm's settings to improve its efficiency and accuracy in deriving symbolic equations. Experimenting with different population sizes, mutation rates, and selection strategies could result in valuable insights into the best practices for SGP application. Furthermore, looking at the instances in which SGP failed to solve the problems indicates that usage of semantically-aware operators could be specially helpful in promoting the algorithm to solve the equations which were not solved in the experiments conducted in this work.

Another prospective is the integration of other types of evolutionary algorithms with SGP. This interdisciplinary approach could lead to hybrid models that leverage the strengths of various evolutionary strategies, potentially leading to breakthroughs in the ability to solve more complex SR problems. Furthermore, applying SGP to a wider array of problems beyond the scope of SR, such as optimization challenges in engineering or predictive modeling in finance, could demonstrate its versatility and effectiveness in other domains. The exploration of these pathways promises to expand the applicability of SGP and evolutionary computation as a whole.

6.5 Solving Decision-Making Problems with SGP

Chapter 5 points out a shortcoming of SGP in solving problems that requires decision making. In order to address this issue, the programmatical mode of the system is introduced. In this mode, the cost of travelling from a program node to the subsequent programs is impacted by the state of the internal registers of the system in addition to the proximity of the programs. The performance of this mode is tested against classic control problems from the OpenAI Gym library and three custom Toy Problems. A new operator called RetCon was introduced which further enhanced SGP's capability in forming conditional pathways making it more adept in solving the targeted class of problems. Comparison with the spatial mode of the system should significant improvements on the evolved solutions, also pointing out that the spatial mode is more suitable for solving continuous value problems.

This chapter also acknowledges limitations within SGP, such as the challenge of balancing the evolution of structure and LGP programs. Future directions are proposed to enhance the model, including the use of parallel island models and the optimization of system hyper-parameters during evolution. It is also pointed out that a different more sophisticated cost function might yield more substantial results. The chapter concludes with an emphasis on the need to test SGP against more realistic and complex problems to truly validate its effectiveness against state-of-the-art problem solvers.

The significance of this chapter is pointed out by introducing the programmatical mode of SGP. A simple change in the cost function, results in substantial changes on both the dynamics of how the individuals evolve and execute, significantly increasing the performance of the system when leveraged against decision-making problems. The introduction of conditional return statements within the SGP model allows for a more refined approach to problem-solving, showcasing its superior performance in a variety of test cases against established methods like TGP and LGP.

Moreover, the chapter sets the stage for further research by identifying potential improvements and directions for future work. The contemplation of alternative cost functions, the use of parallel evolution models, and the refinement of hyper-parameters underline the ongoing evolution of the SGP framework. This work paves the way for a new generation of evolutionary algorithms with enhanced decision-making capabilities, potentially impacting a wide range of fields from artificial intelligence to automated control systems.

Future steps in the field of Spatial Genetic Programming (SGP) as discussed in the text include the exploration of parallel island models, which might better balance the evolution of structural elements and LGP programs. By decoupling these aspects, researchers could focus on refining each component individually while still allowing for interaction and integration, potentially leading to more robust and efficient solutions. Additionally, the development of methods for dynamic hyperparameter optimization during the evolution process presents a promising avenue. This adaptive approach could further tailor the SGP framework to the specific challenges of different problems, enhancing the system's flexibility and performance.

The application of SGP to more complex and realistic problems stands out as a critical future step for the field. By benchmarking SGP against state-of-the-art problem solvers in real-world scenarios, researchers can validate and show the practical effectiveness of the approach.

6.6 Next Steps

The core objective of this dissertation was to dissect the consequences of introducing space as a fundamental concept in genetic programming individuals. Indeed, this study could only cover a small portion of the vast possibilities that come with the utilization of space in computational frameworks. In many cases, such as studying diversity, localization, or the introduction of iterative structures through SGP, the goal was not to focus on the specific topic as much as to present new ideas for future investigations.

SGP, as it currently stands, could be a potential tool for continuing research on spatial GP models. The reason for this statement is based on how this tool, as software, has evolved ever since this research was initiated. These changes were aimed at simplifying the usage of the system, increasing its potency for research by introducing analysis tools, including benchmarks that remove the need for implementations not related to the core of the research, and so on. However, with all the benefits that come with SGP, there is much more that can be done to improve it even further. For example, the implementation can be further optimized, and interfaces can be added to the tool to enable scientists from other disciplines to use the tool with ease. In evolutionary algorithms, in most cases, parts of the algorithm such as evaluation of individuals are mutually exclusive. This aspect allows for these algorithms to be parallelized. SGP is not an exception of this case. Parallelization of the evolutionary algorithm in SGP, and perhaps more, will result in a computationally faster framework allowing researchers to perform experiments in a shorter time.

An important consideration is how the spatial phenomena emerging through the evolution of individuals in SGP resemble nature. While it is extremely difficult, time-consuming, and costly to conduct experiments regarding natural evolution, it is much more feasible to study such experiments in a computational framework, which could be potentially facilitated through SGP. Unlike applications discussed in this dissertation which were more focused on computational problem-solving, there's an opportunity to mimic environmental pressure through regional elements presented in SGP. Such experiments in SGP could lead

to discoveries in the realm of natural evolution.

From a more engineering viewpoint, the idea of including spatial elements in computational algorithms to enhance their performance is not a new concept. In this dissertation, a few classes of problems were tackled, showing that SGP's spatial overhead is not a hindrance to performance in finding proper solutions. While SGP was compared to TGP and LGP in the current manuscript, it is important to extend such comparisons with other state-of-the-art algorithms such as Markov Brains or Neural Networks.

A possible future step towards further investigating spatial elements in SGP is to evolve individuals where their spatial organization is correlated with the spatial properties of the problem at hand. Tackling problems such as image processing where the position of the pixels determine patterns that form shapes, or problems that require physical elements to be positioned in a space such as circuit design could be a good step towards this goal.

In some cases, while solving high-degree equations similar to the $f = a^{40}$ problem, SGP found the problems very trivial, solving them in fewer than five evolutionary generations. This was possible only by utilizing spatial proximity, a property of the dimension of space. This area of research could be more fruitful than what was introduced in this dissertation. In near future, the plan is to maintain and improve the SGP framework, focusing on making it more accessible to the general audience by providing technical documentation and practical guides. Finally, when evolving iterative structures, limitations were put in place to avoid self-loops of nodes. It is interesting to see whether self-loops can be utilized in the system to evolve more elegant and abstract solutions for problems that require iteration.

Attributions and Other Contributions

Chapters of this work are extended text based on two manuscript that were authored by Iliya Miralavy and Wolfgang Banzhaf: [Miralavy and Banzhaf, 2023c] [Miralavy and Banzhaf, 2023a]

Miralavy contributed on the original idea, implementation of software, conducting experiments and writing the original manuscript. Banzhaf, contributed on improving the idea and on the writing and revision of the manuscripts.

I made other contributions to the EC community which are described as follows:

- Protein Optimization Engineering Tool (POET): POET is a GP tool which facilitates screening and mutagenesis in the process of Directed Evolution to aid protein engineers to find fitter proteins with respect to specific target protein functions. As a result of this research, the following manuscripts were prepared: [Miralavy et al., 2022], [Bricco et al., 2023], [Scalzitti et al., 2023]
- An Artificial Chemistry Implementation of a Gene Regulatory Network: This work discusses a biologically more realistic model for gene regulatory networks which incorporates Artificial Chemistry along with a spatial representation to model the interactions between regulatory proteins called the Transcription Factors and the regulatory sites of genes. The following manuscript describes this work: [Miralavy and Banzhaf, 2023b]
- Factorio Learning Environment (FLE): This study utilizes the engine of the video game Factorio to create a benchmark for solving various realistic problem scenarios. This work includes an interface that allows optimizers in any programming language to interact with the benchmark. This work is published as: [Reid et al., 2021]

BIBLIOGRAPHY

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283.

[Allen et al., 2015] Allen, B., Sample, C., Dementieva, Y., Medeiros, R. C., Paoletti, C., and Nowak, M. A. (2015). The molecular clock of neutral evolution can be accelerated or slowed by asymmetric spatial structure. *PLoS Computational Biology*, 11(2):e1004108.

[Amir Haeri et al., 2017] Amir Haeri, M., Ebadzadeh, M. M., and Folino, G. (2017). Statistical genetic programming for symbolic regression. *Applied Soft Computing*, 60:447–469.

[Aoki and Nagao, 1999] Aoki, S. and Nagao, T. (1999). Automatic construction of tree-structural image transformations using genetic programming. In *Proceedings 10th International Conference on Image Analysis and Processing*, pages 136–141. IEEE.

[Astarabadi and Ebadzadeh, 2019] Astarabadi, S. S. M. and Ebadzadeh, M. M. (2019). Genetic programming performance prediction and its application for symbolic regression problems. *Information Sciences*, 502:418–433.

[Augusto and Barbosa, 2000] Augusto, D. A. and Barbosa, H. J. (2000). Symbolic regression via genetic programming. In *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks*, pages 173–178. IEEE.

[Back and Schwefel, 1996] Back, T. and Schwefel, H.-P. (1996). Evolutionary computation: An overview. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 20–29. IEEE.

[Banzhaf et al., 1998] Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann Publishers Inc.

[Barto et al., 1983] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846.

[Beadle and Johnson, 2008] Beadle, L. and Johnson, C. G. (2008). Semantically driven crossover in genetic programming. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 111–116. IEEE.

[Beadle and Johnson, 2009] Beadle, L. and Johnson, C. G. (2009). Semantically driven mutation in genetic programming. In *2009 IEEE Congress on Evolutionary Computation*, pages 1336–1342. IEEE.

[Bender and Williamson, 2010] Bender, E. A. and Williamson, S. G. (2010). *Lists, Decisions and Graphs*. S. Gill Williamson.

[Beyer and Schwefel, 2002] Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies–a comprehensive introduction. *Natural Computing*, 1:3–52.

[Blickle, 2000] Blickle, T. (2000). Tournament selection. *Evolutionary Computation*, 1:181–186.

[Brameier et al., 2007] Brameier, M., Banzhaf, W., and Banzhaf, W. (2007). *Linear Genetic Programming*, volume 1. Springer.

[Bricco et al., 2023] Bricco, A. R., Miralavy, I., Bo, S., Perlman, O., Korenchan, D. E., Farrar, C. T., McMahon, M. T., Banzhaf, W., and Gilad, A. A. (2023). A genetic programming approach to engineering mri reporter genes. *ACS Synthetic Biology*, 12(4):1154–1163.

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv Preprint arXiv:1606.01540*.

[Chan, 2018] Chan, B. W.-C. (2018). Lenia - biology of artificial life. *arXiv Preprint arXiv:1812.05433*.

[Chance, 2022] Chance, G. (2022). Adventure - atari - atari 2600. Accessed: 2022-08-07.

[Chen et al., 2017] Chen, Q., Zhang, M., and Xue, B. (2017). Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. *IEEE Transactions on Evolutionary Computation*, 21(5):792–806.

[Chen and Konstantinides, 2022] Chen, Y.-C. and Konstantinides, N. (2022). Integration of spatial and temporal patterning in the invertebrate and vertebrate nervous system. *Frontiers in Neuroscience*, 16:854422.

[Chopard and Droz, 2005] Chopard, B. and Droz, M. (2005). Cellular automata modeling of physical systems. *Cellular Automata Modeling of Physical Systems*.

[Clegg et al., 2007] Clegg, J., Walker, J. A., and Miller, J. F. (2007). A new crossover technique for cartesian genetic programming. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pages 1580–1587.

[Cliffe et al., 2018] Cliffe, R. N., Scantlebury, D. M., Kennedy, S. J., Avey-Arroyo, J., Mindich, D., and Wilson, R. P. (2018). The metabolic response of the bradypus sloth to temperature. *PeerJ*, 6:e5600.

[Cohen et al., 2009] Cohen, I., Huang, Y., Chen, J., Benesty, J., Benesty, J., Chen, J., Huang, Y., and Cohen, I. (2009). Pearson correlation coefficient. *Noise Reduction in Speech*

Processing, pages 1–4.

[Cui et al., 2015] Cui, G., Li, M., Wang, Z., Ren, J., Jiao, D., and Ma, J. (2015). Analysis and evaluation of incentive mechanisms in p2p networks: A spatial evolutionary game theory perspective. *Concurrency and Computation: Practice and Experience*, 27(12):3044–3064.

[Dale and Fortin, 2014] Dale, M. R. and Fortin, M.-J. (2014). *Spatial Analysis: A Guide for Ecologists*. Cambridge University Press.

[Dang et al., 2017] Dang, D.-C., Friedrich, T., Kötzing, T., Krejca, M. S., Lehre, P. K., Oliveto, P. S., Sudholt, D., and Sutton, A. M. (2017). Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation*, 22(3):484–497.

[Darwin, 1909] Darwin, C. (1909). *The Origin of Species*. New York: PF Collier & Son.

[DeHon et al., 2007] DeHon, A., Giavitto, J.-L., and Gruau, F. (2007). 06361 executive report–computing media languages for space-oriented computation. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[Dell'Ovo et al., 2018] Dell'Ovo, M., Capolongo, S., and Oppio, A. (2018). Combining spatial analysis with mcda for the siting of healthcare facilities. *Land Use Policy*, 76:634–644.

[development team, 2020] development team, T. P. (2020). pandas-dev/pandas: Pandas.

[Dick and Whigham, 2013] Dick, G. and Whigham, P. A. (2013). Controlling bloat through parsimonious elitist replacement and spatial structure. In *European Conference on Genetic Programming*, pages 13–24. Springer.

[Dittrich and Elmenreich, 2015] Dittrich, T. and Elmenreich, W. (2015). Comparison of a spatially-structured cellular evolutionary algorithm to an evolutionary algorithm with panmictic population. In *2015 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES)*, pages 145–149. IEEE.

[Doerr et al., 2008] Doerr, B., Happ, E., and Klein, C. (2008). Crossover can provably be useful in evolutionary computation. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pages 539–546.

[Dorigo et al., 2006] Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

[Dumitrescu et al., 2000] Dumitrescu, D., Lazzerini, B., Jain, L. C., and Dumitrescu, A. (2000). *Evolutionary Computation*. CRC Press.

[Eiben and Smith, 2015] Eiben, A. E. and Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer.

[Enescu et al., 2019] Enescu, A., Andreica, A., and Diosan, L. (2019). Evolved cellular automata for edge detection. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 316–317.

[Fernandes et al., 2018] Fernandes, C. M., Laredo, J. L., and Rosa, A. C. (2018). Spatially structured evolutionary algorithms: Graph degree, population size and convergence speed. *Intelligent Distributed Computing XI*, pages 15–25.

[Feynman et al., 2011] Feynman, R. P., Leighton, R. B., and Sands, M. (2011). *The Feynman Lectures on Physics, Vol. I: The New Millennium Edition: Mainly Mechanics, Radiation, and Heat*, volume 1. Basic Books.

[Fogel, 1962] Fogel, L. J. (1962). *Autonomous Automata*. PhD thesis, University of California, Los Angeles. Dissertation (Ph.D.)–University of California, Los Angeles.

[Fortin et al., 2012] Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A. G., Parizeau, M., and Gagné, C. (2012). Deap: Evolutionary algorithms made easy. *J. Mach. Learn. Res.*, 13(1):2171–2175.

[Fotheringham and Brunsdon, 1999] Fotheringham, A. S. and Brunsdon, C. (1999). Local forms of spatial analysis. *Geographical Analysis*, 31(4):340–358.

[Fox et al., 2024] Fox, J. L., Chundawat, R. S., Kachel, S., Tallian, A., and Johansson, Ö. (2024). What is a snow leopard? behavior and ecology. In *Snow Leopards*, pages 15–29. Elsevier.

[Gear, 1993] Gear, C. (1993). Massive parallelism across space in odes. *Applied Numerical Mathematics*, 11(1-3):27–43.

[Gershenson et al., 2020] Gershenson, C., Trianni, V., Werfel, J., and Sayama, H. (2020). Self-organization and artificial life. *Artificial Life*, 26(3):391–408.

[Giabbanelli et al., 2019] Giabbanelli, P. J., Freeman, C., Devita, J. A., Rosso, N., and Brumme, Z. L. (2019). Mechanisms for cell-to-cell and cell-free spread of hiv-1 in cellular automata models. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 103–114.

[Goldberg, 1989] Goldberg, D. (1989). Genetic algorithms in search, optimization and machine learning,. addition-westly. *Reading MA*.

[Hancock et al., 2022] Hancock, Z. B., Lehmberg, E. S., and Blackmon, H. (2022). Phylogenetics in space: How continuous spatial structure impacts tree inference. *Molecular*

Phylogenetics and Evolution, 173:107505.

[Harding et al., 2013] Harding, S., Leitner, J., and Schmidhuber, J. (2013). Cartesian genetic programming for image processing. In Vladislavleva, E., Ritchie, M. D., and Moore, J. H., editors, *Genetic Programming Theory and Practice X*, pages 31–44. Springer.

[Harding and Miller, 2005] Harding, S. and Miller, J. F. (2005). Evolution of robot controller using cartesian genetic programming. In Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J., and Tomassini, M., editors, *European Conference on Genetic Programming*, pages 62–73. Springer.

[Harris et al., 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with numpy. *Nature*, 585(7825):357–362.

[Haut et al., 2022] Haut, N., Banzhaf, W., and Punch, B. (2022). Active learning improves performance on symbolic regression tasks in stackgp. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 550–553.

[Helmuth et al., 2017] Helmuth, T., McPhee, N. F., Pantridge, E., and Spector, L. (2017). Improving generalization of evolved programs through automatic simplification. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 937–944.

[Helmuth et al., 2014] Helmuth, T., Spector, L., and Matheson, J. (2014). Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643.

[Hickinbotham et al., 2021] Hickinbotham, S. J., Stepney, S., and Hogeweg, P. (2021). Nothing in evolution makes sense except in the light of parasitism: evolution of complex replication strategies. *Royal Society Open Science*, 8(8):210441.

[Hodan et al., 2021] Hodan, D., Mrazek, V., and Vasicek, Z. (2021). Semantically-oriented mutation operator in cartesian genetic programming for evolutionary circuit design. *Genetic Programming and Evolvable Machines*, 22(4):539–572.

[Hofman, 2014] Hofman, M. A. (2014). Evolution of the human brain: When bigger is better. *Frontiers in Neuroanatomy*, 8:15.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The University of Michigan Press.

[Hunter, 2007] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing*

in Science & Engineering, 9(3):90–95.

[Icke and Bongard, 2013] Icke, I. and Bongard, J. C. (2013). Improving genetic programming based symbolic regression using deterministic machine learning. In *2013 IEEE Congress on Evolutionary Computation*, pages 1763–1770. IEEE.

[iParadigms, 2023] iParadigms (2023). ithenticate. https://www.ithenticate.com. Accessed: 2023-11-03.

[Kelly and Heywood, 2017] Kelly, S. and Heywood, M. I. (2017). Emergent tangled graph representations for atari game playing agents. In McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., and García-Sánchez, P., editors, *European Conference on Genetic Programming*, pages 64–79. Springer.

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE.

[Khan et al., 2013] Khan, M. M., Ahmad, A. M., Khan, G. M., and Miller, J. F. (2013). Fast learning neural networks using cartesian genetic programming. *Neurocomputing*, 121:274–289.

[Kicinger et al., 2005] Kicinger, R., Arciszewski, T., and De Jong, K. (2005). Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers & Structures*, 83(23-24):1943–1978.

[Killingback and Doebeli, 1996] Killingback, T. and Doebeli, M. (1996). Spatial evolutionary game theory: Hawks and doves revisited. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 263(1374):1135–1144.

[Killingback and Doebeli, 1998] Killingback, T. and Doebeli, M. (1998). Self-organized criticality in spatial evolutionary game theory. *Journal of Theoretical Biology*, 191(3):335–340.

[Kondo and Miura, 2010] Kondo, S. and Miura, T. (2010). Reaction-diffusion model as a framework for understanding biological pattern formation. *Science*, 329(5999):1616–1620.

[Koza, 1992a] Koza, J. R. (1992a). *Genetic programming as a means for programming computers by natural selection*. ACM.

[Koza, 1992b] Koza, J. R. (1992b). *Genetic Programming: On the Programming of Computer by Means of Natural Selection*. MIT Press.

[Koza, 1994] Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

[Kuo et al., 2021] Kuo, Y. P., Arrieta, C. N., and Carja, O. (2021). A theory of evolutionary dynamics on any complex spatial structure. *BioRxiv*, pages 2021–02.

[Langdon et al., 2010] Langdon, W. B., Harman, M., and Jia, Y. (2010). Efficient multi-objective higher order mutation testing with genetic programming. *Journal of Systems and Software*, 83(12):2416–2430.

[Martínez et al., 2014] Martínez, Y., Trujillo, L., Naredo, E., and Legrand, P. (2014). A comparison of fitness-case sampling methods for symbolic regression with genetic programming. In *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, pages 201–212. Springer.

[McGinley et al., 2011] McGinley, B., Maher, J., O'Riordan, C., and Morgan, F. (2011). Maintaining healthy population diversity using adaptive crossover, mutation, and selection. *IEEE Transactions on Evolutionary Computation*, 15(5):692–714.

[Miller et al., 1995] Miller, B. L., Goldberg, D. E., et al. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9(3):193–212.

[Miller, 1999] Miller, J. F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142.

[Miller, 2020] Miller, J. F. (2020). Cartesian genetic programming: Its status and future. *Genetic Programming and Evolvable Machines*, 21(1):129–168.

[Miller et al., 2000] Miller, J. F., Job, D., and Vassilev, V. K. (2000). Principles in the evolutionary design of digital circuits—part i. *Genetic Programming and Evolvable Machines*, 1:7–35.

[Miralavy and Banzhaf, 2023a] Miralavy, I. and Banzhaf, W. (2023a). Analyzing the impact of space in evolving spatially represented genetic programming models. Manuscript in preparation for submission to Genetic Programming and Evolvable Machines.

[Miralavy and Banzhaf, 2023b] Miralavy, I. and Banzhaf, W. (2023b). An artificial chemistry implementation of a gene regulatory network. Accepted to appear.

[Miralavy and Banzhaf, 2023c] Miralavy, I. and Banzhaf, W. (2023c). Spatial genetic programming. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 260–275. Springer.

[Miralavy et al., 2022] Miralavy, I., Bricco, A. R., Gilad, A. A., and Banzhaf, W. (2022). Using genetic programming to predict and optimize protein function. *PeerJ Physical Chemistry*, 4:e24.

[Moore, 1990] Moore, A. W. (1990). Efficient memory-based learning for robot control. Technical report, University of Cambridge.

[MSU, 2023] MSU (2023). Institute for cyber-enabled research.

[Mundhenk et al., 2021] Mundhenk, T., Landajuela, M., Glatt, R., Santiago, C. P., Petersen, B. K., et al. (2021). Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. *Advances in Neural Information Processing Systems*, 34:24912–24923.

[Narlikar and Blelloch, 1997] Narlikar, G. J. and Blelloch, G. E. (1997). Space-efficient implementation of nested parallelism. *ACM SIGPLAN Notices*, 32(7):25–36.

[Nicolau et al., 2015] Nicolau, M., Agapitos, A., O'Neill, M., and Brabazon, A. (2015). Guidelines for defining benchmark problems in genetic programming. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1152–1159. IEEE.

[Ofria and Wilke, 2004] Ofria, C. and Wilke, C. O. (2004). Avida: A software platform for research in computational evolutionary biology. *Artificial Life*, 10(2):191–229.

[Oltean and Grosan, 2003] Oltean, M. and Grosan, C. (2003). A comparison of several linear genetic programming techniques. *Complex Systems*, 14(4):285–314.

[OpenAI, 2023] OpenAI (2023). Chatgpt (feb 13 version) [large language model].

[Pantridge and Spector, 2020] Pantridge, E. and Spector, L. (2020). Code building genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2020)*, pages 994–1002.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

[Perkis, 1994] Perkis, T. (1994). Stack-based genetic programming. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 148–153. IEEE.

[Piszcz and Soule, 2006] Piszcz, A. and Soule, T. (2006). A survey of mutation techniques in genetic programming. In *Proceedings of the 8th Annual Conference on Genetic and*

Evolutionary Computation, pages 951–952.

[Poli et al., 2007] Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization: An overview. *Swarm Intelligence*, 1:33–57.

[Poli and Langdon, 1998] Poli, R. and Langdon, W. B. (1998). *Genetic Programming with One-point Crossover*. Springer.

[Poli et al., 1998] Poli, R., Langdon, W. B., et al. (1998). On the search properties of different crossover operators in genetic programming. *Genetic Programming*, pages 293–301.

[Python, 2021] Python (2021). Python language reference, version 3.9. https://www.python.org. Accessed: 11/02/2023.

[Rechenberg, 1973] Rechenberg, I. (1973). Evolution strategies. *Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*.

[Reid et al., 2021] Reid, K. N., Miralavy, I., Kelly, S., Banzhaf, W., and Gondro, C. (2021). The factory must grow: automation in factorio. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 243–244.

[Richards and Amos, 2016] Richards, D. and Amos, M. (2016). Regulatory representations in architectural design. *Evolutionary Computation in Gene Regulatory Network Research*, pages 362–397.

[Roca et al., 2009] Roca, C. P., Cuesta, J. A., and Sánchez, A. (2009). Evolutionary game theory: Temporal and spatial effects beyond replicator dynamics. *Physics of Life Reviews*, 6(4):208–249.

[Sastry et al., 2005] Sastry, K., Goldberg, D., and Kendall, G. (2005). Genetic algorithms. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 97–125.

[Scalzitti et al., 2023] Scalzitti, N., Miralavy, I., Korenchan, D. E., Farrar, C. T., Gilad, A. A., and Banzhaf, W. (2023). Computational peptide discovery with a genetic programming approach. Manuscript is submitted to the Journal of Biological Engineering and is currently under review.

[Schranz et al., 2020] Schranz, M., Umlauft, M., Sende, M., and Elmenreich, W. (2020). Swarm robotic behaviors and current applications. *Frontiers in Robotics and AI*, page 36.

[Schubert et al., 2017] Schubert, E., Sander, J., Ester, M., Kriegel, H. P., and Xu, X. (2017). Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21.

[Schwefel, 1981] Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc.

[Searson et al., 2010] Searson, D. P., Leahy, D. E., and Willis, M. J. (2010). Gptips: an open source genetic programming toolbox for multigene symbolic regression. In *Proceedings of the International Multiconference of Engineers and Computer Scientists*, volume 1, pages 77–80. Citeseer.

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

[Sutton, 1995] Sutton, R. S. (1995). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D., Mozer, M., and Hasselmo, M., editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press.

[Tomassini, 2005] Tomassini, M. (2005). *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time*. Springer.

[Tomassini, 2010] Tomassini, M. (2010). Cellular evolutionary algorithms. In *Simulating Complex Systems by Cellular Automata*, pages 167–191. Springer.

[Tran et al., 2016] Tran, B., Zhang, M., and Xue, B. (2016). Multiple feature construction in classification on high-dimensional data using gp. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE.

[Turner and Miller, 2013] Turner, A. J. and Miller, J. F. (2013). Cartesian genetic programming encoded artificial neural networks: a comparison using three benchmarks. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, pages 1005–1012.

[Udrescu et al., 2020] Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., and Tegmark, M. (2020). Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *arXiv preprint arXiv:2006.10782*.

[Uy et al., 2009] Uy, N. Q., Hoai, N. X., and O'Neill, M. (2009). Semantics based mutation in genetic programming: The case for real-valued symbolic regression. In *15th International Conference on Soft Computing, Mendel*, volume 9, pages 73–91.

[Uy et al., 2011] Uy, N. Q., Hoai, N. X., O'Neill, M., McKay, R. I., and Galván-López, E. (2011). Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12:91–119.

[Vasicek and Sekanina, 2014] Vasicek, Z. and Sekanina, L. (2014). Evolutionary approach to approximate digital circuits design. *IEEE Transactions on Evolutionary Computation*,

19(3):432–444.

[Vayadande et al., 2022] Vayadande, K., Pokarne, R., Phaldesai, M., Bhuruk, T., Patil, T., and Kumar, P. (2022). Simulation of conway's game of life using cellular automata. *International Research Journal of Engineering and Technology (IRJET)*, 9(01).

[Vieu, 1997] Vieu, L. (1997). Spatial representation and reasoning in artificial intelligence. In *Spatial and Temporal Reasoning*, pages 5–41. Springer.

[Virtanen et al., 2020] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020). Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature Methods*, 17(3):261–272.

[Waskom, 2021] Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

[Woodward, 2003] Woodward, J. (2003). Ga or gp? that is not the question. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 2, pages 1056–1063 Vol.2.

[Yao and Hsu, 2009] Yao, M.-J. and Hsu, H.-W. (2009). A new spanning tree-based genetic algorithm for the design of multi-stage supply chain networks with nonlinear transportation costs. *Optimization and Engineering*, 10(2):219–237.

[Zhong et al., 2018] Zhong, J., Feng, L., Cai, W., and Ong, Y.-S. (2018). Multifactorial genetic programming for symbolic regression problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(11):4492–4505.