PhD Thesis: Modelling Evolvability in Genetic Programming

Submission to the School of Graduate Studies

Supervisor: Dr. Wolfgang Banzhaf

by
© *Benjamin Fowler*

A thesis proposal submitted to the
School of Graduate Studies
in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy

Department of Computer Science
Memorial University of Newfoundland

August 2018

St. John's Newfoundland

Abstract

We develop a tree-based genetic programming system, capable of modelling evolvability during evolution through artificial neural networks (ANN) and exploiting those networks to increase the generational fitness of the system. This thesis is empirically focused; we study the effects of evolvability selection under varying conditions to demonstrate the effectiveness of evolvability selection. Evolvability is the capacity of an individual to improve its future fitness. In genetic programming (GP), we typically measure how well a program performs a given task at its current capacity only. We improve upon GP by directly selecting for evolvability. We construct a system, Sample-Evolvability Genetic Programming (SEGP), that estimates the true evolvability of a program by conducting a limited number of evolvability samples. Evolvability is sampled by conducting a number of genetic operations upon a program and comparing the fitnesses of resulting programs with the original. SEGP is able to achieve an increase in fitness at a cost of increased computational complexity. We then construct a system which improves upon SEGP, Model-Evolvability Genetic Programming (MEGP), that models the true evolvability of a program by training an ANN to predict its evolvability. MEGP reduces the computational cost of sampling evolvability while maintaining the fitness gains. MEGP is empirically shown to improve generational fitness for a streaming domain, in exchange for an upfront increase in computational time.

Acknowledgements

I thank my supervisor, Dr. Wolfgang Banzhaf, for advice, guidance, and editing. I thank my parents, Scott and Kathleen Fowler, for their long-standing support of my academic pursuits. I thank my girlfriend, Jahyd Chubbs, for her continued support. I also thank Dr. Andrew Webb for naming the Modelling Evolvability in Genetic Programming system.

I thank ACENET for material support. Experiments were conducted on hardware provided by ACENET, in partnership with Compute Canada, a high performance computing (HPC) consortium for universities based in the Atlantic Provinces of Canada. ACENET is funded by the Atlantic Provinces Opportunities Agency, the Canada Foundation for Innovation, and the provincial governments of Nova Scotia, New Brunswick, and Newfoundland and Labrador.

Contents

List of Tables

List of Figures

Nomenclature

g	Evolvability Ceasing Generation
p	Evolvability Weight Parameter
*ap*EGP	*a priori* Evolvability Genetic Programming
ANN	Artificial Neural Network
ESC	Evolvability Selection Criteria
GP	Genetic Programming
HPC	High-Performance Computing
ME	Magnitude Evolvability
MEGP	Model-Evolvability Genetic Programming
NSWED	New South Wales Electricity Demand
OT#	OrderTree #
PPE	Positive-Probability Evolvability
PR	Probability Robustness
SEGP	Sample-Evolvability Genetic Programming
SGP	Standard Genetic Programming

Chapter 1

Introduction

Genetic programming (GP) is an evolution-inspired machine learning method where different programs compete with one another to perform a specified task [49]. The programs are usually initialized as a random starting population, and then the programs are tested using some fitness metric. Programs may also be initialized less randomly, as more functional programs [5]. Those that are more fit are more likely to be selected as parent programs for the next generation, where mutation and recombination modify the children. This process repeats until a program is fit enough or some other stopping condition is reached. How fitness is measured determines which programs are more likely to be selected. However, fitness, by itself, misses details which may be useful in determining which programs may actually be more useful to evolution. Some programs, judged to be more fit, may actually be less useful to evolutionary processes than those judged as less fit.

GP could be more effective or efficient if we select individuals not solely on their fitness but also on how they may contribute to evolutionary processes. It would be useful to select more evolvable individuals that may contribute more to the fitness of future generations. Evolvability is a measurement of this property. Evolvability indicates the capacity of an individual to improve its fitness [1]. We opt to define different types of evolvability based upon how it is measured, as detailed in Chapter 3. However, evolvability is computationally expensive to measure, which means that it is computationally impractical, in real-world applications, to measure evolvability for individuals and then use evolvability to aid selection processes. We therefore propose to model evolvability using GP properties that are computationally inexpensive to generate, and, once such models are developed, evolvability may be calculated and utilized in the GP selection process to increase the efficacy of evolution.

The viability of evolvability modelling is verified experimentally in three parts: *a priori* evolvability calculation, parallel evolvability sampling, and parallel evolvability modelling. First, we generate properties related to evolvability, as well as evolvability itself, *a priori* for a specific problem, using that data to develop a machine learning model for evolvability. An *a priori* environment provides idealized modelling conditions, but it lacks practicality. Secondly, we demonstrate that sampling for evolvability to use in selection, while evolution actually occurs, can provide fitness improvements. However, sampling is computationally expensive. Thirdly, we build evolvability models for evolvability, while evolution actually occurs.

1.1 Contributions of the Thesis

The primary contributions of this thesis are the development of *a priori* Evolvability Genetic Programming (*ap*EGP), Sample-Evolvability Genetic Programming (SEGP), and Model-Evolvability Genetic Programming (MEGP). We verify each system empirically. Additionally, we develop evolvability selection pressures. Previous work with evolvability in some of the domains explored in this thesis was published [20].

We use *ap*EGP to demonstrate that modelling evolvability in ideal conditions is possible, and further, that modelled evolvability can be used in fitness selection to achieve improved fitness results. We conduct experiments on GP benchmark problems. The results indicate that evolvability can be modelled accurately from easily (computationally inexpensive) generated evolutionary statistics. Further, modelled evolvability can be used in fitness selection to improve the true fitness. Since *a priori* evolutionary statistics are required to generate an evolvability model, *ap*EGP has practical reservations. It is necessary to run standard genetic programming (SGP) first to generate enough data to use in modelling. Furthermore, many runs may be necessary to obtain diversity in programs, which is required for an accurate evolvability model. Additionally, the course of evolution is altered by using evolvability in selection, so the underlying distribution of programs may no longer be representative, thus weakening the evolvability model.

To rectify these problems, we develop an evolvability system that runs in parallel with GP. SEGP approximates the true evolvability of an individual, to reduce the computational burden. Evolvability is approximated by conducting a limited number of genetic operations upon a program and measuring the resulting programs' fitnesses. Samples are conducted as evolution occurs. This increases computational complexity, as more fitness evaluations are required in order to evaluate evolvability. We evaluate SEGP on a synthetic domain and a real streaming domain. SEGP is able to outperform SGP on both domains, in terms of best generational true fitness. However, the additional computations required to achieve these improved fitness results are onerous.

We develop MEGP to improve upon the computational demands of SEGP. In MEGP, we conduct some initial evolvability sampling generations to build an evolvability model. We ensure that this evolvability model performs well by periodically comparing the predicted evolvability with the sampled evolvability. We use sampled evolvability in selection when it is calculated, otherwise we use the less computationally intensive predicted evolvability. We evaluate MEGP on the same domains as SEGP. MEGP outperforms SGP on those domains, in terms of best generational true fitness. Furthermore, it suffers only marginal best generational true fitness loss compared to SEGP. In return, computational complexity is greatly reduced, being only about twice that of SGP. In the streaming domain, the increase in computational complexity compared to SGP can be front-loaded, offering ongoing fitness benefits at almost no ongoing cost. The true fitness results in the streaming domain for both SEGP and MEGP are comparable to state-of-the-art machine learning methods.

In order to use evolvability to improve evolution, we use evolvability fitness pressures. The fitness function of SGP is modified to include evolvability for selection purposes. We develop several mechanisms to enforce varying amounts of evolvability pressure. This includes a decaying evolvability weight, an eventual drop of evolvability selection, and interleaved evolvability selection. Further, different measures of evolvability are used in selection, including the probability that a program will improve, and the average magnitude of the change in fitness.

1.2 Thesis Structure

Before we describe our experiments and their results, we review related literature in Chapter 2, describing GP and related concepts in detail. Chapter 3 describes evolvability metrics and fitness selection processes. Chapter 4 describes the methodology and results of preliminary experiments, using *a priori* evolvability modelling. Chapter 5 describes experiments in the OrderTree (OT) domain. Chapter 6 describes experiments in a real-world streaming domain. Finally, Chapter 7 offers a conclusion to the achieved results.

Chapter 2

Background and Related Work

This chapter describes the motivation for developing a genetic programming (GP) system incorporating an evolvability model. It describes related work, explains prior research regarding evolvability and related concepts, and describes relevant properties that may be used to model evolvability.

2.1 Genetic Programming Background

As mentioned in Chapter 1, GP is an evolution-inspired machine learning method where different programs compete with one another to perform a specified task [49]. The ultimate goal of artificial intelligence and machine learning, in general, is to able to give a computer a task, then have it learn and perform the task on its own [5]. GP would allow a computer to program itself to accomplish this goal. GP has not been able to accomplish this lofty goal. However, GP has practical uses. GP has produced at least seventy-six results that are capable of competing with directly-human solutions [50]. These results come from fields as diverse as circuit design, image recognition, and bioinformatics. In particular, GP can find competitive results where relationships among variables are not well understood, where there is a clear objective, but no human insights on how to achieve that objective. There has not been a particular focus on developing GP for the purpose of generalization [28, 61, 67]. There is more focus on reducing bloat. Bloat describes excessively large programs that occur when using GP, where much of the program is of negligible impact on fitness. Occam's Razor and Minimum Description Length principles indicate that larger programs should be worse than smaller programs, so it is thought that reducing bloat would improve generalization [5].

A widespread way of measuring efficacy in machine learning is generalization or predictive accuracy. A model has good generalization if it can accurately predict the label of unseen instances. This is a general problem in machine learning, where if we train our model on all possible instances, the resulting models will overfit to represent those instances. A model is overfit if it is specifically tuned to training instances, which means the model will be accurate at predicting the values of those instances, but poor at predicting values of unseen instances. Conventional machine learning uses learning as a paradigm, and evolutionary computation uses evolution as a paradigm; learning is adaptation at the individual level, and evolution is adaptation at the population level [52]. To prevent overfitting, one usually just splits the instance space between training and testing, to validate training models. Test instances are unobserved during the training of the algorithm. Once training is complete, the model's predictive power can be evaluated by comparing the predicted labels with the true labels of the test instances. The difficulty seems to be the evolutionary paradigm; namely, trying to solve one specific problem using all possible test cases, instead of building a learning system that can adapt to a class of similar problems. In the evolutionary paradigm, there is a lack of emphasis on building a learning system; instead, there is just one problem to be solved.

There are thoughts that reducing bloat would reduce overfit and *vice versa*, but other results question this [88]. Making GP more generalizable is not as simple as biasing the search toward smaller solutions to reduce bloat. Other machine learning methods, such as artificial neural networks (ANNs), support vector machines (SVMs), and Bayesian networks are more developed in terms of success in generalization/classification problems [10, 21].

The core difference between machine learning methods is how they search the problem space. GP randomly searches in many directions, forming a complex relationship between the individual programs, their resulting characteristics, and their impact on fitness. The analysis is facilitated by using a genotype-phenotype mapping, or a genotype-fitness mapping. A genotype in GP is the set of genes of a program. A gene in the program is the fundamental unit of information, whose characteristics vary by what representation is being used. There are many different representations for programs available for GP, including machine code, trees, and graphs. A gene in tree-based GP would be a node in the tree, and the genotype would be the tree. The actual observed characteristics of a program, the underlying program derived from the genotype, is called the phenotype. The phenotype is the program encoded in the genotype. The phenotype may or may not change significantly with changes in the genotype, or a small change in the genotype can drastically change the phenotype, so the distinction between them is important [58].

GP may be inspired by natural selection, but there are many components of biological systems that are not considered when using GP, such as epigenetics and evolvability [4, 37]. There is fierce debate about the role of evolvability in biological systems; whether it acts as a catalyst for natural selection or is a by-product of it [65]. Properties related to evolvability and robustness, such as self-repair, may emerge in artificial systems without modifying the underlying systems to encourage their emergence [64]. The lack of consensus on the role of evolvability in biological systems serves as an impetus for this work, which is to analyze the role of evolvability and other properties in GP. The lack of human understanding of the human-competitive results [50] serves as a further reason to study the structure of GP; understanding where GP succeeds and why could yield insights to improvements in other problems. Using genotype-phenotype or genotype-fitness mappings could also prove beneficial to the study of these properties [56]. In fitness landscapes, independent axes are genotypes, and dependent axes measure fitness. The shape of these landscapes indicates how fitness is distributed in the search space. Observable characteristics such peaks, valleys, and plateaus in the landscape indicate problem characteristics. For example, having many plateaus indicates that there are many genetic operations that have a neutral effect on fitness. The following sections more rigorously define evolvability and robustness and describe how we can exploit their properties to improve the efficacy or efficiency of GP.

2.2 Evolvability

Evolvability in GP refers to the ability of an individual or population of programs to produce higher fitness individuals [1]. The ability of the individual program to solve a specific problem by using testing instances judges individual fitness; such an approach encourages the evolution of programs which best solve a specific problem, though it does not encourage evolvability. Consequently, such solutions are not applicable when confronted with a similar problem. Additionally, it is difficult to evolve a solution to the new problem from our existing one. To encourage more evolvable programs, it would be beneficial to quantify evolvability and exploit the resulting quantities when judging fitness.

Biologically, evolvability is defined as the ability of a population to respond to selection [19]. In his review of other works, Pigliucci [65] comes the conclusion that evolvability, however it may be defined, itself evolves, but there is a lack of evidence to see if this is caused by natural selection or other evolutionary mechanisms. The genetic makeup of biological species influences the cross-over and mutation rates, which is strong evidence that biological evolvability evolves. The biological perspective of evolvability is important, as it may indicate if we need to select for evolvability during GP or if it should occur on its own. Altenberg [2] notes that evolutionary computation brought about more biological-based evolutionary interest in evolvability; evolvability in organisms was simply presumed to exist. Altenberg further notes that there were 170 papers published in 2013 alone that mention the evolution of evolvability. To encourage more evolvable programs, it would be beneficial to quantify evolvability and exploit the resulting quantities when judging fitness. Kattan and Ong [44] use Bayesian inference to adjust fitness functions in order to encourage evolvability. Using genotype-phenotype or genotype-fitness mappings could also prove beneficial to the study of these properties [56]. Properties related to evolvability and robustness, such as self-repair, may emerge in artificial systems without modifying the underlying systems to encourage their emergence [64].

We visualize evolvability in Figures 2.1 and 2.2. In Figure 2.1, we see a high evolvability phenotype. The chosen phenotype has low fitness, but is surrounded by higher-fitness phenotypes. The chosen phenotype is easily able to change to a higher fitness phenotype. In Figure 2.2, we see a low evolvability phenotype. The chosen phenotype has high fitness, but is surrounded by lower-fitness phenotypes. The chosen phenotype is more likely to become lower fitness. In standard genetic programming (SGP), comparing the two central phenotypes, we would consider the higher fitness, lower evolvability phenotype to be better. We may gain on fitness relative to the other

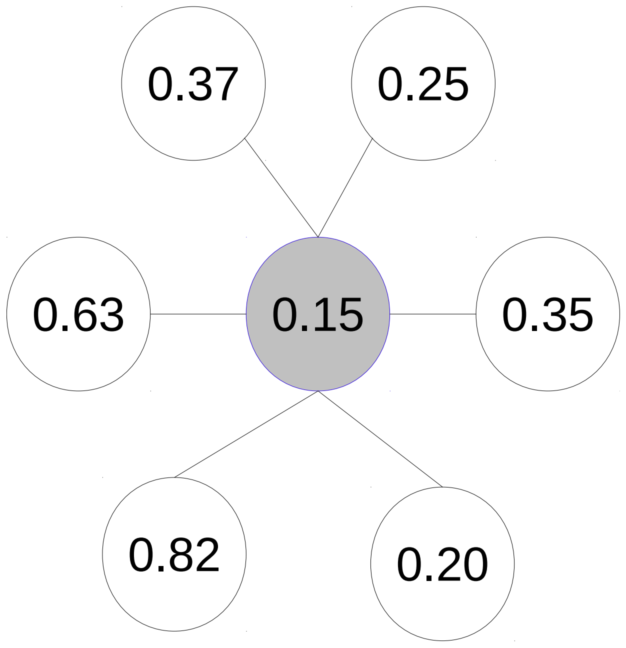

Figure 2.1: Phenotype-space graph with a central chosen phenome with high evolvability and low fitness. Each node represents a phenotype, a physical characteristic mapped from a genotype. The number in each phenotype indicates its fitness, which we are attempting to maximize. The edges represent genetic operations, such as mutation, that will change the chosen phenotype into another of its neighbours. Genetic operations lead to fitness improvements, so the chosen individual in this Figure has high evolvability.

phenotype, but the high evolvability phenotype has much greater capacity for future fitness. This demonstrates the primary motivation for selecting for evolvability. In SGP, evolvability is ignored.

We are concerned with maximizing fitness in SGP. To maximize fitness we favour programs that currently have greater fitness. There are various selection methods [5, 49, 66] that apply varying amounts of selection pressure. However, selection is inherently driven by differences in fitness. This procedure ignores other information and follows an assumption based upon natural selection: the fittest individuals should be more likely to be selected to produce the proceeding generation, and such offspring are more likely to be more fit, so gradually, good fitness individuals result.

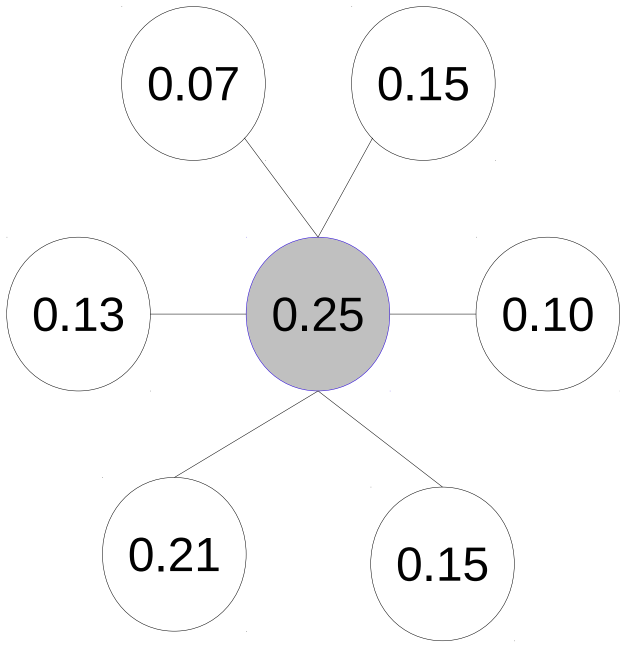

Figure 2.2: Phenotype-space graph with a central chosen phenotype with low evolvability and high fitness. Genetic operations lead to fitness loss, so the chosen individual has low evolvability. However, the chosen phenotype in this Figure has a higher fitness than the central phenotype of Figure 2.1.

This process does not directly consider structural properties of programs, such as bloat [76]. Instead, selection is meant to allow more desirable structural properties, which allow greater fitness, to emerge [3]. The process further ignores how changes in genotype change the phenotype [24], how this affects fitness, and how this might skip optima in the fitness search space [79]. Evolvability is related to these structural properties; by analyzing their interrelatedness, we should have insight to improve GP by accommodating them, in lieu of ignoring them, by measuring and selecting for evolvability. Basset *et al.* [6] postulate that bloat occurs because offspring are not effectively inheriting the phenotype traits from their parents. The notion is that ideally, we want to perform a cross-over on the phenotype, not the genotype.

Pragmatically, we can improve upon exhaustive searches to measure evolvability by using sampling or estimation [90]. This is much more computationally feasible than exhaustive search. In either case, why not simply keep the resulting most fit individual? Sampling still adds a significant computational burden compared to SGP, as sufficient samples are required to estimate evolvability, but even adding a single sample doubles the computational time required in SGP. This is because the main unit of work in GP is fitness evaluation, and each sample requires its own fitness evaluation. As such, evolvability is too computationally expensive to measure directly. Instead, current literature efforts to exploit evolvability do so indirectly, without having to measure it, such as defining new evolvability metrics [82] and characterizing evolvability's relatedness to other properties [35]. There has been some success in determining how much to select for evolvability, but only under limited circumstances [91, 92]. Li *et al.* [54] have had success balancing fitness selection with diversity metrics, using multi-objective optimization. Multi-objective approaches using Pareto dominance or hypervolume indicators, with various objective criteria, are well-studied in the literature, generally targeting concepts related to evolvability, such as diversity, rather than evolvability itself [29, 77].

The consequences of selecting fitness without considering any structural properties can be demonstrated by the parity problem. The parity problem is commonly used in GP [9, 32, 16, 49, 63, 74, 97]. The parity problem is counting the number of 1's that occur in a bitstring of a specified length, and returning 0 for an even number of occurrences, and 1 for an odd number of occurrences. It is sensible to use the entire instance space as training instances. The simplest solution in tree-based GP is simply using the exclusive-or operation (XOR) on each input. We see that Figure 2.3 has a solution for the 3-parity problem. OR and AND represent their respective boolean operations, 0 and 1 represent their respective constant inputs, and $X1, X2, X3$ represent inputs of the bitstring. Considering only fitness, Figure 2.3 is a perfect solution. A portion of this solution is not contributing anything to fitness, either positive or negative; this is bloat. Considering only fitness, this program is as good as a solution that has the bloated portion removed. Bloat is an extensively covered topic in GP [5]. It was considered to be correlated with overfitting and functional complexity; however, recent research has shown this not to be the case [87]. Furthermore, eliminating bloat may not be beneficial [76].

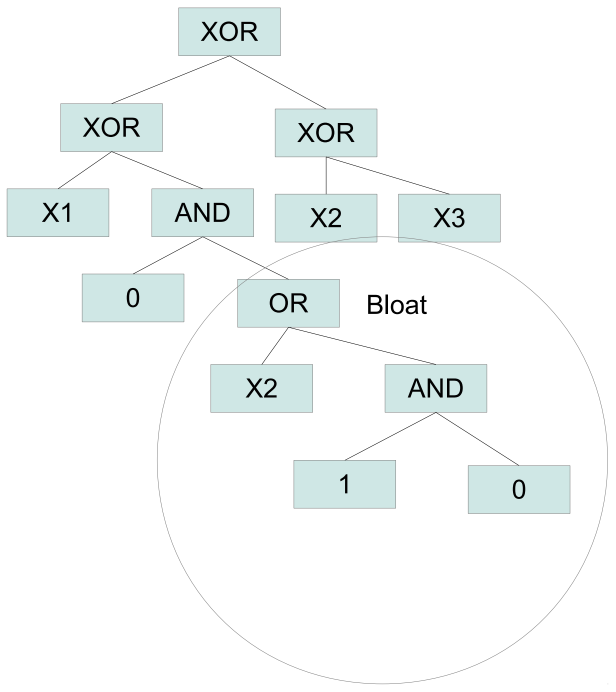

Figure 2.3: Bloat in the parity problem. The problem is solved without the bloat, but the bloated portion contributes no negative or positive fitness, and so will remain.

Consider the 3-parity problem, concerning evolvability. In Figure 2.4, we have a solution which produced a fitness of 0; every input sequences produces the incorrect output. However, consider how close the solution is to the ideal solution; if the constant input 1 were to be changed to 0, we would have the best solution. If we were to consider a genotype-fitness landscape, then Figure 2.4 would represent the minimum fitness, with single operations that could yield the maximum fitness. Such information is completely ignored in SGP; we only see the fitness of 0. Figure 2.4 is a high evolvability solution, in theory, since it can easily move to higher fitness.

Contrast such a solution to an alternative of higher fitness, Figure 2.5. Consider its complexity, and how difficult it may be to transform it into the ideal solution. As fitness is concerned, Figure 2.5 is a much better solution than Figure 2.4, though we observe that it should have low evolvability, in theory. Measuring evolvability would involve the automatic recognition of the potential for greater fitness that Figure 2.4 represents, and exploiting it would involve using those measurements properly to reduce the number of generations GP required to find a good solution while improving the final fitness of those solutions. The question is this: can the potential for greater fitness without explicitly testing fitness values?

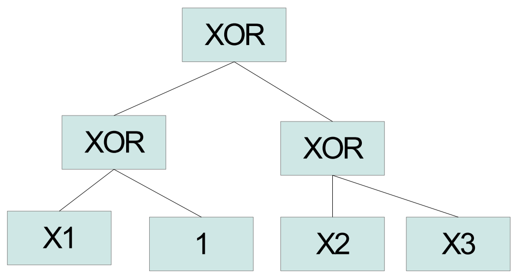

Figure 2.4: A perfectly unfit solution to the 3-parity problem, of high evolvability.

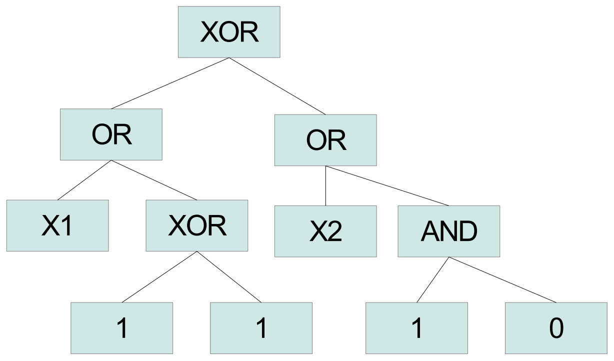

Figure 2.5: A high-fitness solution to the 3-parity problem, of low evolvability.

Altenberg [1] describes a method of measuring evolvability through a transmission function. Essentially, one considers all possible results of genetic operations on an individual and computes the average fitness of those results. This is an intuitive method of measuring evolvability; an individual is highly evolvable if its potential offspring have high fitness. It also requires extensive computations; instead of conducting one genetic operation on an individual, we need to conduct all possible operations. Then, each fitness case needs to be evaluated for each such operation. Thus, measuring evolvability in this matter would require many orders of magnitude more computational effort than SGP. Furthermore, measuring evolvability in this matter only considers the average potential fitness, which may be disproportionately low; there may be clusters of operations which produce very fit individuals, but many more operations which produce poor fitness individuals. There may even be an ideal solution reachable in one operation. The average does not indicate the distribution of how often fitness is improved and by how much. Smith *et al.* [79] propose several variants considering the relative fitness of offspring, and one considering only a constant percentage of the top-performing offspring.

Furthermore, such definitions only consider evolvability from offspring of single generation at a time. That is, if two or more genetic operations yield high fitness solutions, it is not encapsulated by these metrics. While these metrics may be intuitive, it is impractical to use them to alter development in GP; they require too much additional computation. To calculate them, we need to calculate all possible offspring for every program we have as well as their fitnesses. If we were willing to do that, it would be simpler to select the greater fitness offspring anyway. This provides motivation for a more exploitable evolvability metric.

2.3 Robustness

Robustness in GP refers to the ability of an individual or population of programs to retain functionality despite perturbations that occur during evolution [39]. An evolvable individual becomes more fit more easily, and a robust individual does not become less fit more easily. Evolvability concerns fitness gains, where robustness concerns maintaining fitness. Robustness competes with evolvability, but both qualities can facilitate faster evolution. By effectively measuring both robustness and evolvability we can generate individuals with both properties, or at least analyze the tradeoff between them. This will lower the computational cost of GP, which has many practical ramifications.

Consider again the 3-parity problem, this time concerning robustness. In Figure 2.6, we have a solution which is nearly correct; it merely ignores the $X1$ input, while the other inputs are computed correctly. However, consider what operations could take place to worsen the solution; there are many damaging changes that could occur. If we were to consider a genotype-fitness landscape, then Figure 2.6 would represent fair fitness, with many operations that would reduce fitness. As with evolvability, such information is completely ignored in SGP. Figure 2.6 is a brittle since it can easily move to lower fitness.

Contrast such a solution to an alternative of lower fitness, given in Figure 2.7. Consider its complexity, and how difficult it may be to transform it to worse solutions. Once again, as for fitness is concerned, Figure 2.6 is a much better solution than Figure 2.7, though we observe it has lower robustness. Measuring robustness would involve the automatic recognition of the potential for fitness retention that Figure 2.7 represents, and exploiting it would involve using those measurements properly to reduce the number of generations GP requires to find a good solution since worse solutions are less likely to be developed. The question is this: can we measure the potential for fitness retention without explicitly testing fitness values?

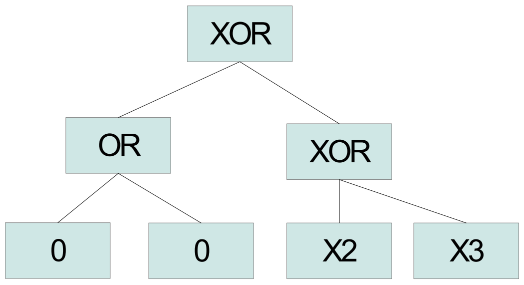

Figure 2.6: A fair fitness solution to the 3-parity problem, of low robustness.

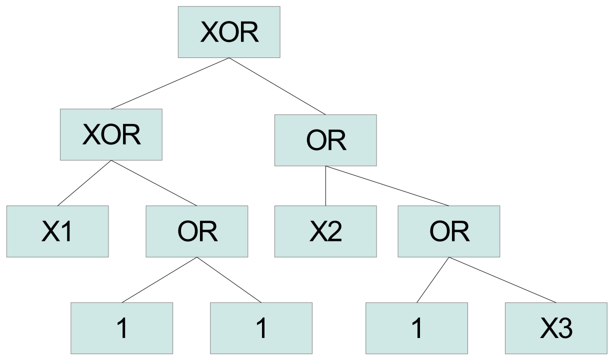

Figure 2.7: A low fitness solution to the 3-parity problem, of high robustness.

As mentioned earlier, a solution with high evolvability is one that becomes more fit more easily, while a solution with a high robustness is one that does not become less fit more easily. Metrics for robustness involve counting how many neutral genetic operations are possible against how all possible genetic operations [25, 38, 39]. A neutral operation is one where the genotype changes, but fitness and/or the phenotype does not change. More neutral operations indicate greater robustness. Biologically, by genotype, robustness and evolvability compete with each other [89]. Results from biological perspectives and computational perspectives indicate that robustness and evolvability may be negatively correlated at the genotype level, but support one another at the phenotype level, when considering mutation [38, 39, 89]. However, this may indicate that a negative correlation may still exist in representations where the genotype and phenotype are equivalent. Schulte *et al.* [73] survey the mutational robustness of 22 pieces of software and find they are highly robust; 37% of the mutations had neutral effects on functionality. This indicates that robustness may emerge inadvertently through selection in both biological and artificial systems. However, natural systems still have more evolvability and robustness than observed in man-made systems [81].

Intuitively, a robust individual is likely more resistant to change than one that is not, where an evolvable individual is more likely to change to a more fit individual. These appear to be competing factors. A greater understanding of their interoperability should yield more efficient and accurate GP. If evolvability and robustness are negatively correlated, such that obtaining both highly evolvable and robust individuals is difficult, evolvability and robustness may benefit from being favoured at different generations; *e.g.*, evolvable when fitness is low in the early generations, robust when fitness is high in later generations. This would reduce the total number of generations required to reach an acceptable solution by increasing the rate at which we explore the solution search space in early generations and reducing it to check for more optimal solutions later in learning, similar to adjusting the learning rate parameter in ANNs [96].

2.4 Dormancy and Locality

In GP, there can be significant sections of programs which, in addition to not affecting fitness, provide no change in output, regardless of input. These sections are referred to as introns in the GP literature. Introns may be categorized by their behaviour; Nordin *et al.* [60] propose several categories, based on whether their lack of contribution of fitness is due to the fitness cases themselves or apply to the entire problem domain, and whether cross-over operations can introduce a change in fitness. Identifying all introns is computationally expensive. However, it is computationally inexpensive to identify a certain type of intron, that occurs when a code section is never executed for any fitness case; these are dormant sections [41]. In tree-based or cartesian GP, these nodes are referred to as dormant nodes and can account for the majority of the nodes, around $90\% - 95\%$ [41, 59]. Dormancy is a type of bloat, and since it does not affect fitness, it is not normally encouraged or discouraged through fitness selection in GP. Despite the apparent uselessness of dormant sections of code, dormancy is helpful; if dormant nodes are detected and removed, performance actually suffers, and more generations are required to reach comparable solutions [41]. The potential use of dormant nodes is characterized in Figure 2.8. A full fitness solution to the problem has is found and is contained within a dormant section of the tree. It does not contribute to fitness, and so simply removing that section would not affect fitness. However, if a cross-over operation occurred at the proper node, we would have a perfect solution. Essentially, we have a high evolvability solution, once more indicating the interrelatedness of these structural concepts and indicating that normal fitness testing ignores them. The amount of dormancy is a structural property in GP, which may relate to evolvability and robustness. Individuals may have useful dormant code, unaccounted by fitness. The dormant code may allow fitness to improve more easily, or improve the fitness stability of the individual.

Locality is another structural property in GP, relating to evolvability, robustness, and genotype-phenotype mappings. A problem has high locality if neighbouring genotypes correspond to neighbouring phenotypes [24]. High locality problems are generally easier to solve. Low locality indicates a more rugged search space, which indicates a more difficult search. Furthermore, the ruggedness describes how robust and evolvable the search space is [25, 43, 79, 90]. Neutral genetic operations represent plateaus in the search space. Evolvability and robustness act as counterparts; steep inclines indicate great fitness gains moving toward optima, but also great fitness losses moving away from optima. There is motivation to organize all the structural properties together, to analyze their interactions, for they all affect problem difficulty,

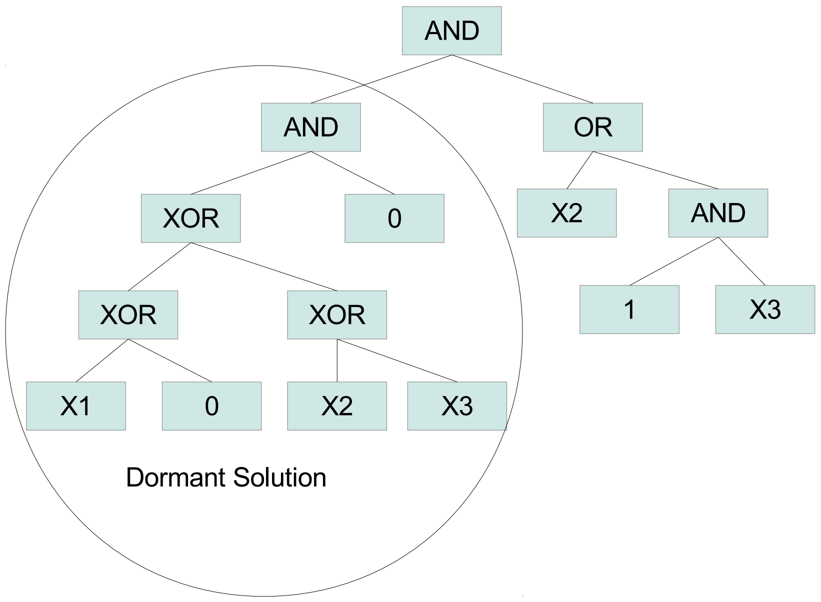

Figure 2.8: A dormant solution to the 3-parity problem.

the efficiency of the search, and the efficacy of the search.

Chapter 3

Experimental Approach

This chapter describes the theoretical foundations for the experimental approach of evolvability-influenced fitness selection. The experiments follow a development of evolvability evaluation concepts culminating in the development of Model-Evolvability Genetic Programming (MEGP). We design evolvability metrics, models, and selection pressures to achieve fitness improvements. The *a priori* Evolvability Genetic Programming (*ap*EGP) system is developed initially to demonstrate that accurately modelling evolvability using easily generated GP statistics is possible. We then develop Sample-Evolvability Genetic Programming (SEGP) and MEGP in parallel. We continually refine the GP systems to achieve further fitness and computational time improvements. We empirically evaluate these refinements in Chapters 4, 5, and 6 but we detail the theory of the experimental approach here.

3.1 *ap*EGP

We demonstrate the potential of evolvability modelling with *ap*EGP in Chapter 4. The ultimate goal of this work is to develop a system that can model evolvability in parallel to evolution. To demonstrate that evolvability modelling using easily generated evolutionary statistics is possible, we first show that evolvability can be modelled accurately in idealized conditions. Using a large amount of evolutionary statistics generated *a priori*, we can confirm that such statistics can sufficiently predict the evolvability of an individual. Without such confirmation, evolvability modelling is moot.

We design and implement a tree-based GP system that records measurements of evolvability by sampling, along with records of other structural properties for every population that occurs during evolution. Further, we model evolvability using these records, then exploit their predicted values during evolution, in order to develop a faster and more efficient GP system. This is accomplished by modifying an existing tree-based GP system to track and record additional structural elements for specific problems. Once generated, evolvability is modelled using machine learning algorithms.

Evolvability models are incorporated into the existing tree-based GP system, to predict evolvability without the need to sample them. Then, the predicted values will be used to influence selection beyond the standard fitness measurements. Initially, we consider the various structural properties of solutions generated by genetic programming (GP) for a small parity problem, and a contrived regression problem consisting of a single input variable. These problems are not known for their complexity, though they are commonly used as benchmark problems. These simple conditions provide a useful environment to evaluate the potential of evolvability modelling and selection. Generating good models for evolvability before evolution is not eminently practical (why would we run evolution in its entirety first to generate training instances for our machine learning algorithms ahead of time?), but it can demonstrate which are the most relevant statistics required for modelling, and the most appropriate learning

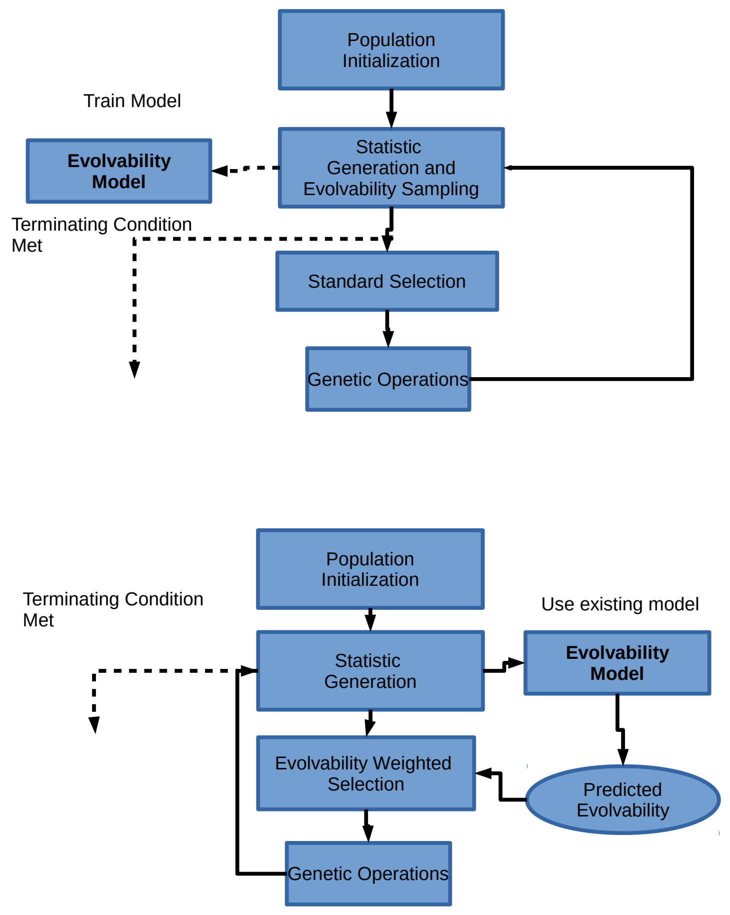
Population Initialization
Train Model
Evolvability Model
Statistic Generation and Evolvability Sampling
Terminating Condition Met
Standard Selection
Genetic Operations
Population Initialization
Terminating Condition Met
Use existing model
Statistic Generation
Evolvability Model
Evolvability Weighted Selection
Predicted Evolvability
Genetic Operations

Figure 3.1: Flow chart of the *ap*EGP system. *ap*EGP consists of two processes. In the first, we build an evolvability model. We initialize a population of programs. We generate evolutionary statistics for each program. We proceed with selection based on fitness, conduct genetic operations, and repeat until end conditions are met (which is either perfect fitness or a specific number of generations occur). We train an evolvability model on the evolutionary statistics. In the second, we conduct evolution again, only this time, selection is altered by evolvability. Evolvability is predicted for each program, as they occur, using the evolvability model.

parameter settings. The flow of the *ap*EGP system is shown in Figure 3.1.

3.2 SEGP and MEGP

Following the creation of *a priori* evolvability models, we demonstrate that evolvability calculated (or sampled) as evolution occurs can provide fitness improvements. Using calculated evolvability is computationally expensive, but it can indicate some model-agnostic parameters, such as the strength of the selection of evolvability, under set conditions. Model-agnostic parameters possible should be determined before introducing potential errors by using models for evolvability. We use SEGP in combination with varying selection mechanisms and evolvability metrics to ensure we can improve on the true fitness of standard genetic programming (SGP). This establishes the benefits of evolvability selection without the potential accuracy loss introduced by a model. The flow of the SEGP system is shown in Figure 3.2.

These results inform the next phase of experiments, using MEGP, where models are training in parallel to evolution. Dynamically-built models have practical significance. The previous experiments establish the best parameters to build the models, and the best conditions to exploit evolvability. Experiments with the dynamic models make further adjustments to establish the effectiveness of modelling evolvability, considering the most efficient ways to exploit evolvability during evolution. The flow of the MEGP system is shown in Figure 3.3.

SEGP is a necessary component of MEGP. Further, we evaluate the effectiveness of evolvability models as they correspond to fitness improvements relative to sampled evolvability. For these reasons, SEGP and MEGP are deployed in parallel on experiments in Chapter 5 and Chapter 6. SEGP establishes that sampled evolvability can be used in selection to improve fitness results, and MEGP improves upon those

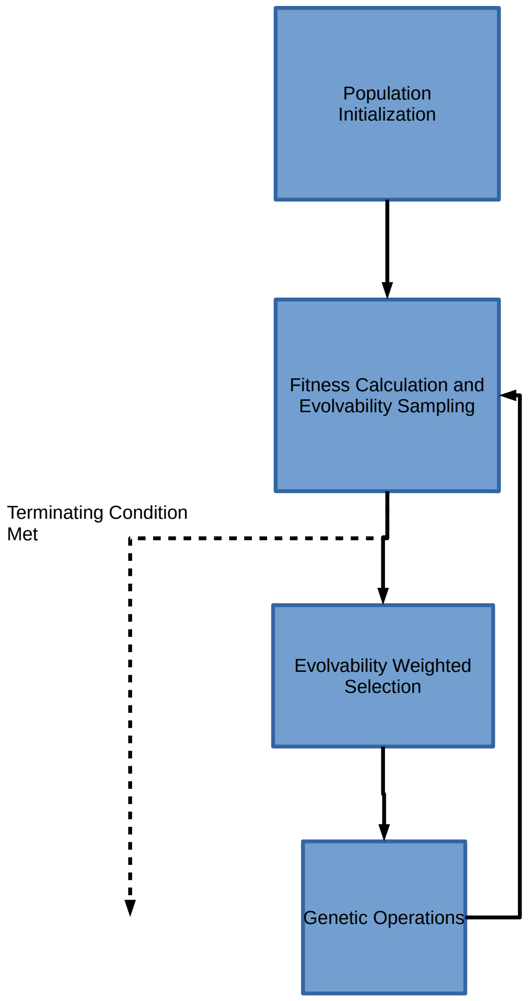
Population Initialization
Fitness Calculation and Evolvability Sampling
Terminating Condition Met
Evolvability Weighted Selection
Genetic Operations

Figure 3.2: Flow chart of the SEGP system. We initialize a population of programs. We generate evolutionary statistics and sample evolvability values for each program. We proceed with selection based on fitness and sampled evolvability. Then we conduct genetic operations, and repeat until end conditions are met (which is either perfect fitness or a specific number of generations occur).

results by substantially reducing the computational time required to provide them.

3.3 Selection Mechanisms

We use evolvability in fitness selection to find individuals that are more capable than fitness alone can indicate. As discussed in Chapter 2, fitness may be missing details that indicate a more capable individual. Evolvability can compensate for some of the missing detail. However, evolvability alone should not be used to judge the capability of an individual. Selecting for mere capacity of improvement leads to low fitness individuals, for it is a trivial matter to improve their fitness when they cannot be any less fit. Furthermore, at evolution's end, we are only concerned with the highest true fitness individual. Fitness and true fitness are equivalent in problems such as parity or OrderTree (OT). Using fitness alone to select the best individual when evolution ends is sensible for such problems. True fitness is not known, or should not be used, in concept shift problems such as electricity demand prediction. True fitness should not be used since the purpose of the model is to predict classes of unseen instances. We must build models that can generalize, not merely classify what is already known. More evolvable individuals may generalize better. Nevertheless, fitness cannot be abandoned completely. Even where true fitness is used, selecting for evolvability provides benefits. A more evolvable population is less likely to be trapped in local fitness minima and its fitness is more likely to improve.

However, we must determine how important evolvability is relative to fitness and if this importance varies over time or domains. We devise a weighted sum of evolvability and fitness called the Evolvability Selection Criteria (ESC). The weights are determined experimentally. With large weights, fitness is overpowered by evolvability, such that we select for evolvability and use fitness as a mere tiebreaker. This is a

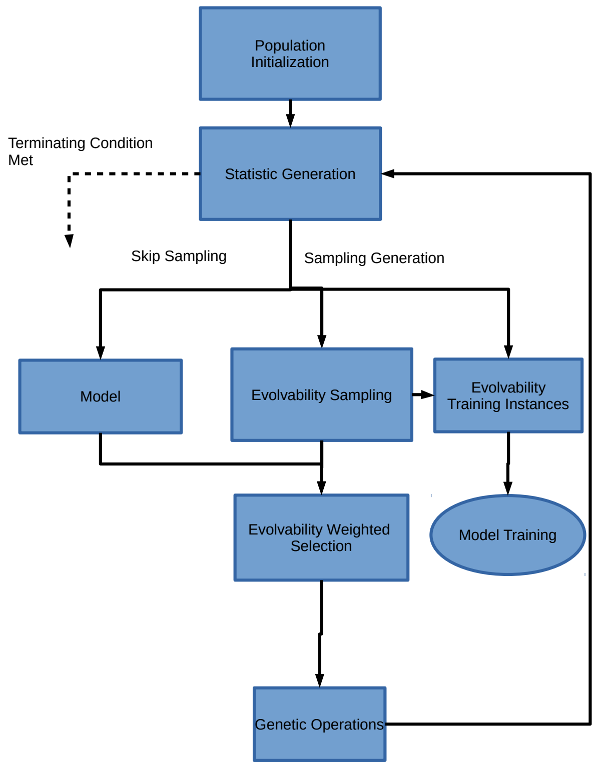
Population Initialization
Statistic Generation
Terminating Condition Met
Skip Sampling
Sampling Generation
Model
Evolvability Sampling
Evolvability Training Instances
Evolvability Weighted Selection
Model Training
Genetic Operations

Figure 3.3: Flow chart of the MEGP system. We initialize a population of programs. We generate evolutionary statistics for each program. We then determine if we are conducting an evolvability sampling generation. Evolvability samples are needed to train the evolvability model. The first few generations sample evolvability, and sampling may occur later to ensure the model is sufficiently accurate. If sampling occurs, we collect the program statistics and evolvability labels to create training instances, which are then used to train the model. Sampled evolvability is used in selection, where it occurs. If sampling is not conducted, the program statistics are fed to the model to generate predicted evolvability labels, which are then used in selection. Then we conduct genetic operations, and repeat until end conditions are met (which is either perfect fitness or a specific number of generations occur).

lexicographical ordering where evolvability is selected first. With small weights, evolvability is overpowered by fitness, such that we select for fitness and use evolvability as a tiebreaker. A balance of evolvability and fitness exists. We search for the balance by applying varying weights at varying times.

We favour evolvability more in earlier evolutionary generations in problems that use true fitness since fitness is clearly most significant at evolution's end. Stronger early evolvability and weaker late evolvability is similar to simulated annealing, which adjusts temperature according to a schedule, allowing for less optimal (low fitness) solutions earlier, and enforcing more optimal (high fitness) solutions later [48]. Research into optimizing annealing schedules may provide insight into evolvability selection. We favour a more consistent evolvability selection in all other domains. ESC includes, but is not limited to, weights decaying over time, random usage, generational interleaving, and generational cut-off points.

3.4 GP Statistics

GP offers many potential statistics that are generated as evolution occurs. We may use any statistic that can aid the prediction of evolvability. However, as the purpose of modelling evolvability is to reduce the time complexity inherent in sampling evolvability, any statistics we use should not require significant overhead to generate. Furthermore, statistics that may seem indicative of evolvability may be too sparsely generated to be of practical use. For example, a full frequency count of each node type may strongly predict evolvability, but requires so many attributes to track that a model has difficulty functioning accurately. The most relevant statistics are determined by feature selection. GP statistics that we use are generation, size, height, number of terminal nodes, number of functional nodes, dormancy ratio, previous fitness, and fitness.

3.5 Evolvability Metrics

We measure evolvability using several methods. The true evolvability of a function requires measuring the fitnesses of all its possible offspring. This is computationally infeasible. We sample for evolvability instead. Evolvability is sampled by conducting a certain number of genetic operations, at random (as it would occur during standard evolution), and measuring the fitnesses of the resulting offspring. More samples translates to greater accuracy, at the cost of increased computational time. Each sample increases computational time significantly since the main unit of work in GP is typically fitness evaluation. We find how many samples are required to achieve fitness improvements. We model evolvability based on the sampled evolvability, so more error is introduced into the model based on the error of the sampled evolvability. If the sampled evolvability is too inaccurate, the model may fail to predict well, as it trains on something that is too noisy.

There are variations in how to measure sampled evolvability. Evolvability is defined as the capacity of an individual to improve, but this is vague. We develop evolvability metrics based on sampling, genetic operation, and other criteria. The first type of evolvability metric is the probability of improvement, Positive Probability Evolvability (PPE). PPE is the percent chance that an individual will improve its fitness when a genetic operation is conducted. The magnitude of improvement is ignored in PPE, and neutral changes are equivalent to negative ones. The second type of evolvability metric is the probably of a neutral or better change when a genetic operation is conducted. This is actually a measure of robustness; a measure of the resistance to negative change. As such, it is referred to as Probability Robustness

(PR). The third type of evolvability metric is the mean change in fitness, Magnitude Evolvability (ME). PPE is generally small, as few changes are positive ones. There is less chance of variation with PPE.

We conduct either mutation or crossover operations to measure evolvability. Mutation is favoured, as the results are independent of the population. Crossover evolvability depends on the population, and we can consider either the incoming fitness (change of the current individual) or outgoing fitness (change of the population).

Chapter 4

Preliminary Experiments

In this chapter, we describe experiments and results regarding parity and a regression problem using *a priori* Evolvability Genetic Programming (*ap*EGP). These experiments establish that building an accurate evolvability model is possible on some common genetic programming (GP) benchmark problems, and further, that selecting for evolvability, not just fitness, can be beneficial. These experiments provide encouraging results, indicating that more extensive research on modelling evolvability is justified. Evolvability models are constructed *a priori*, training with instances from evolution occurring in standard genetic programming (SGP) with similar experimental conditions.

Initial experiments focus on the small parity and simple regression domains. Further experiments extend to other classification problems, of artificial and real domains, that have performance results available using state-of-the-art tree-based GP. This verifies that performance gains obtained by modelling evolvability and robustness are broadly applicable and comparable to the best available methods in tree-based GP. Initial experiments involve these domains, and structural data generated by one run in each, under specific GP evolutionary parameters. SGP, that is, GP without consideration of evolvability, provides a baseline with which to compare *ap*EGP. The earliest generation in which a certain degree of fitness is attained is recorded, if one is found in the specified number of generations. This is repeated, to find a mean generation required to achieve a specified amount of fitness, as well as a rate of failure to achieve that fitness.

4.1 Parity

The six parity problem is used. That problem has 64 fitness cases. For the Boolean domain, the symbols used in the tree-based GP are: AND, OR, XOR, NOT, 0, 1, and one input for each of the six bits. Dormancy is tracked through short-circuiting of the AND and OR operations. That is, if the first input to an AND function is 0 for all fitness cases, or the first input to an OR function is 1 for all fitness cases, there is no need to evaluate the second input, and that entire subtree will be flagged as dormant.

The evolutionary parameters are as shown in Table 4.1. Evolvability is modelled by an artificial neural network (ANN), with the following learning parameters: a number of hidden nodes equal to half the sum of the number of attributes and classes used, a learning rate of 0.1, momentum of 0.01, a random seed of 0, a training time of 10000 epochs, a validation set size of 10%, and a validation threshold of 20 epochs. There is further validation of the model through use of 10-fold cross-validation. The attributes, generated *a priori* by the GP system, are as follows: generation, tree size, function frequency, non-input terminal frequency, input frequency, dormancy ratio, previous fitness, and fitness. The class is evolvability, as determined by sampling, as the probability of a positive change in fitness within one cross-over operation of at least 5%.

Table 4.1: GP evolutionary parameters for the six parity domain.

Parameter	**Value**
Runs	500
Population Size	100
Generations	400
Crossover Probability	0.9
Tournament Size	3
Probability of Non-Terminal Crossover	0.9
Min Initial Depth	2
Standard Mutation Probability	0.1
Standard Mutation Max Regeneration Depth	2
Swap-point Mutation Probability	0.05
Max Initial Depth	6
Mutation Max Regen Depth	2
Max Depth	8
Swap Mutation Probability	0.1
Initial Grow Probability	0.5
Probability to Mutate a Function Node	0.5
Initial Evolvability Samples	100

Selection is altered by first selecting the individual with the greater fitness, as normal. If the individuals have identical fitness, select by evolvability. In the six parity problem, there are only 64 possible fitness results, and there are many fitness collisions. Fitness collisions are when individuals have equal fitness.

The ANN model had an accuracy of 81.8%, under 10-fold cross-validation. Under the evolvability class-split condition, about one-third of the instances reported positive, and the rest negative. That is, about one-third of the individuals, sampled for evolvability, had a 5% chance or greater of increasing their fitness with one cross-over operation. Under the experimental conditions, this would be sufficiently accurate, since evolvability was only significant during selection if the comparable individuals had equal fitness. Any guess that one would be more likely to improve than the other would be welcome, since that is an improvement over arbitrarily selection one instead

Table 4.2: The mean minimum generation to achieve the specified fitness levels for the six parity problem. The error range indicates the mean with 95% confidence, as generated using the Student's t-distribution. The P-Value is calculated using Student's t-test, which indicates whether those means are significantly different.

	70%	80%	90%	100%
SGP	34.72 ± 3.58	61.52 ± 5.83	93.49 ± 8.66	97.44 ± 9.63
*ap*EGP	26.00 ± 2.24	50.36 ± 4.83	77.76 ± 7.18	89.95 ± 8.46
P-Value	0.0001	0.0039	0.0061	0.2512

of the other. Fitness would always be more significant.

It is shown in Table 4.2 that *ap*EGP allows for faster convergence toward the optimal solution for the six parity problem. However, the difference in generations appears to increase as the threshold increases, but lowers to insignificance for the perfect fitness solution itself. Table 4.3 indicates that this may be due to *ap*EGP being able to find a perfect fitness solution where SGP fails to do so; the more difficult initial conditions are not overcome by SGP, and so do not count against the mean, but *ap*EGP succeeds, though it still requires more generations to do so. The effectiveness of *ap*EGP system is further demonstrated by the failure rates, which are much less in all cases. These results indicate that developing models *a priori* and predicting evolvability can increase evolutionary effectiveness.

Table 4.3: The failure rate of the specified systems to reach the specified fitness level before evolution is terminated for exceeding the maximum allowed generations.

	70%	80%	90%	100%
SGP	0.162	0.208	0.312	0.410
*ap*EGP	0.004	0.032	0.114	0.206

4.2 Regression

A contrived regression problem involving one input is used. The contrived formula is:

$$x \times (x \times (x \times (x + 37.67859) + 14.53219) - 8.3724)$$

The fitness cases are determined by sampling 100 random points between -1 and 1, inclusive. For the regression domain, the functional symbols used in the tree-based GP are: addition, subtraction, multiplication, protected division. Protected division returns 1, if the second input is less than 0.001. The terminals are the input, and random terminals valuing between -1 and 1, inclusive, may be generated. Dormancy is tracked through short-circuiting of protected division and multiplication where the first input is 0.

Table 4.4: GP evolutionary parameters for the regression domain.

Parameter	**Value**
Runs	300
Population Size	100
Generations	1000
Crossover Probability	0.9
Tournament Size	3
Probability of Non-Terminal Crossover	0.9
Min Initial Depth	2
Standard Mutation Probability	0.1
Standard Mutation Max Regeneration Depth	2
Swap-point Mutation Probability	0.05
Max Initial Depth	6
Mutation Max Regen Depth	2
Max Depth	8
Swap Mutation Probability	0.1
Initial Grow Probability	0.5
Probability to Mutate a Function Node	0.5
Initial Evolvability Samples	100

The evolutionary parameters are as shown in Table 4.4. Evolvability is modelled by an ANN, with the following learning parameters: a number of hidden nodes equal to half the sum of the number of attributes and classes used, a learning rate of 0.1, momentum of 0.01, a random seed of 0, a training time of 10000 epochs, a validation set size of 10%, and a validation threshold of 20 epochs. There is further validation of the model through use of 10-fold cross-validation. The attributes, generated *a priori* by the GP system, are as follows: generation, tree size, function frequency, non-input terminal frequency, input frequency, dormancy ratio, previous fitness, and fitness. The class is evolvability, as determined by sampling, as the probability of a positive change in fitness within one standard mutation operation.

Selection is altered by first determining if the comparable individuals have significantly similar fitness. If the difference in their fitness is less than 0.01, then instead of selecting by fitness, we consider the sum of fitness and evolvability for each individual.

The ANN model had a correlation coefficient of 0.9558, and a root mean squared error of 0.0282, under 10-fold cross-validation. Under the experimental conditions, this would be sufficiently accurate, since evolvability was only significant during selection if the comparable individuals had nearly equal fitness. Fitness is more likely to be significant, and is still under consideration, even when the comparable individuals are nearly equal. SGP achieves a mean best accuracy of 83.48%, and *ap*EGP 81.02%.

Table 4.5: The mean minimum generation to achieve the specified fitness levels for the contrived regression problem. SGP outperforms *ap*EGP for the regression problem.

	70%	80%	90%
SGP	272.5	303.0	405.6
*ap*EGP	278.6	343.7	439.7

4.3 Discussion

We find that lexicographical evolvability selection provides benefits to the six parity problem but pseudo-lexicographical evolvability selection does not provide benefits to the constructed regression problem. Fitness in GP has a tendency to plateau in the six parity problem. The population reaches points where it is difficult to improve upon fitness or has achieved optimal fitness. Evolvability selection encourages the evolution of a population that can escape these fitness plateaus. The parity search space is rugged and the population may converge to low fitness. The six parity results are contrasted by the regression problem results. The regression problem has a smoother search space and its fitness does not plateau easily. There exists some single genetic operations that yield improved fitness. Using pseudo-lexicographical evolvability selection does not increase the population's ability to escape local minima. The discrete search space of parity contrasts with the more continuous search space of regression. If it is always easy to achieve improved fitness in a problem then there is not enough evolvability differentiation between individuals to make a difference.

We hypothesize that evolvability is more desirable in earlier generations, where fitness has not yet converged and it is not yet inherently desirable to have high fitness. In later generations, fitness becomes more significant, as we desire to optimize fitness (and thus find the best solution to the problem) even at the expense of evolvability. The experiments in this chapter use lexicographical or pseudo-lexicographical evolvability selection, which instead provides uniform light evolvability selection pressure. The light pressure ensures that fitness is indeed prioritized in later generations. However, evolvability is not favoured over fitness in early generations. The light pressure still allows the parity problem to escape local minima but does not encourage a more evolvable population for the regression problem. Regression may benefit from stronger early evolvability pressure but light uniform evolvability selection pressure does not provide benefits.

Chapter 5

Order Tree Experiments

In this chapter, we describe experiments and results regarding the OrderTree (OT) domain. These experiments explore the ideal learning parameters for using evolvability in selection and expand on the preliminary experiments by conducting model building in parallel to evolution.

An extendible synthetic domain will be most useful for this work. White *et al.* [93] proposed a set of benchmark problems to replace previously-used, simple problems. Among the list of new synthetic, extendible problems is the OT problem [36]. A synthetic, extendible problem such as the OT problem allows for tunable problem difficulty; thus the conditions under which the use of evolvability is most beneficial may be more easily examined.

An OT domain may be defined as having a size of n. Function nodes and terminal nodes take on values of whole numbers on a range of $[0, n-1]$. Function nodes all take two arguments. The fitness of a solution is calculated in a top-down fashion. A node will add 1 to the total fitness of the solution if its numeric value is strictly greater than its parent's numeric value, and, in the restricted version of the OT problem, only if the parent is also adding to the total fitness of the solution. Thus, the optimal solution is an ordered tree, where the root is the functional node valued at 0, its children are valued at 1, and so on. The OT problem is useful because the difficulty is tunable to n, where difficulty may be increased by increasing n, thus increasing our functional and terminal set. Furthermore, node dormancy is easily determined as a by-product of fitness evaluation. Problem difficulty may be further tuned by adjusting how much fitness is contributed by each node; by weighing higher-valued nodes more greatly (i.e., by increasing fitness greater than 1 for any given node) the fitness structure may be changed. This alters the fitness landscape, and encourages higher-valued nodes to be selected, even though this interferes with finding the optimal solution. A more evolvable solution would still favour lower-valued nodes. This allows for tuning the desirability of evolvability. Tuning the OT problem in these two ways will demonstrate the problem conditions for the effectiveness of the proposed system.

5.1 Experiments

We conduct a series of experiments to determine the conditions under which evolvability may be used to produce improved fitness results in OT problems of varying difficulty. We begin by verifying that evolvability selection is useful with Sample-Evolvability Genetic Programming (SEGP). We verify that modelling evolvability in ideal conditions is possible with *a priori* Evolvability Genetic Programming (*ap*EGP). Then we verify that evolvability selection is still beneficial when modelling occurs in parallel with evolution, with Model-Evolvability Genetic Programming (MEGP). The experiments, their supporting figures and results are summarized in Table 5.1.

Table 5.1: Summary of OT experimental results.

Figures	**Parameters**	**Purpose**	**Results**
5.2-5.10	Evolvability weight, evolvability ceasing generation	Evaluate SEGP on increasingly difficult problems	Low evolvability weights and low evolvability ceasing generations preferred. More difficult problems benefit from more evolvability.
5.11-5.15	Evolvability weight, evolvability ceasing generation	Evaluate *ap*EGP on increasingly difficult problems	*ap*EGP performs comparably with SEGP.
5.16-5.21	Evolvability weight, evolvability ceasing generation	Optimize evolvability weight and evolvability ceasing generation on OT 8.	MEGP performs comparably with SEGP. Moderate evolvability weight and low evolvability ceasing generation produces the best performers.
5.22-5.25	Evolvability weight, evolvability cycling generations	Optimize evolvability cycling generation parameters.	Interleaved evolvability selection performs significantly better than evolvability ceasing generations.
5.26-5.28	Evolvability weight, evolvability cycling generations	Evaluate MEGP with cycling generations.	MEGP is still able to produce comparable results with interleaving evolvability generations.
5.29-5.32	Max training epochs, fitness improvement thresholds, non-decaying evolvability	Evaluate other evolvability pressure methods.	Alternative methods fail to perform as well as evolvability cycling.
5.33, 5.34	Evolvability weight, evolvability cycling generations	Compare best performers of SEGP and MEGP.	Both SEGP and MEGP perform significantly better than SGP, and comparably with each other. MEGP requires comparable time complexity to SEGP for the OT 8 problem, and both only require about twice as much time as SGP.

5.1.1 Sampling Accuracy

How many samples are needed to estimate evolvability for an individual? Calculating precise evolvability is computationally infeasible for practical genetic programming (GP). Instead of calculating all possible results of all possible genetic operations for any given individual genetic program, we elect to instead conduct sampling, where a random subset of all possible genetic operations is applied. Sampling can approximate the precise calculation of evolvability for a fraction of the computational cost. How many samples are necessary to produce a reasonable approximation of the correct evolvability, such that selection errors will occur less than 5% of the time? How accurate must the approximation be to achieve an improvement when using evolvability to guide selection? This subsection details experiments conducted to determine how many evolvability samples should be obtained for each individual, to balance accuracy and computational efficiency.

In order to answer these questions, we must first define more experimental parameters. Sampling accuracy experiments are conducting using the Open BEAGLE Puppy and WEKA based system. We need only compare the difference in evolvability given by various numbers of samples; we would expect more samples to represent true evolvability more accurately. A simple problem will be as useful as a difficult one, so the OT 4 problem is used. Several evolvability metrics exist. We opt to define evolvability as the probability of a mutation operation resulting in a strictly positive fitness change. This may differ from other metrics in two ways: the probability of change instead of magnitudes of change, and excluding neutral changes. Preliminary experiments indicated that selecting for the probability of a positive fitness change were more productive than when neutral changes were included. Similarly, they indicated that using probabilities instead of the average magnitude of fitness change were

Table 5.2: Evolutionary parameters for varying the number of samples, for the OT 4 problem.

Parameter	**Value**
Population Size	50
Crossover Probability	0.9
Tournament Size	3
Probability of Non-Terminal Crossover	0.9
Min Initial Depth	3
Standard Mutation Probability	0.05
Max Initial Depth	6
Mutation Max Regen Depth	2
Max Depth	6
Swap Mutation Probability	0.05
Initial Grow Probability	0.5
Probability to Mutate a Function Node	0.5

more productive. Mutation operations are considered, in order to evaluate evolvability of individuals without considering how the gene pool of the population would affect measurements, as it would measuring evolvability using cross-over operations. More samples are required to achieve a good approximation if we consider the average magnitude of change of fitness. Furthermore, selecting for the greater positive magnitude of fitness change will heavily bias evolution toward lower fitness individuals, as they have the greatest capacity for fitness improvement. We discount neutral changes, as this encourages a bias toward large trees in the OT problem, as they have many possible neutral mutations. Considering neutral changes to be equivalent to positive ones encourages robustness, but not evolvability. We use Positive Probability Evolvability (PPE) to evaluate evolvability in this chapter. For brevity, when we refer to evolvability in this chapter, we refer to PPE.

To determine how many samples are necessary to achieve a reasonable approximation of evolvability, we conduct the following experiment. We vary the number of samples while keeping other experimental conditions constant, and compare the

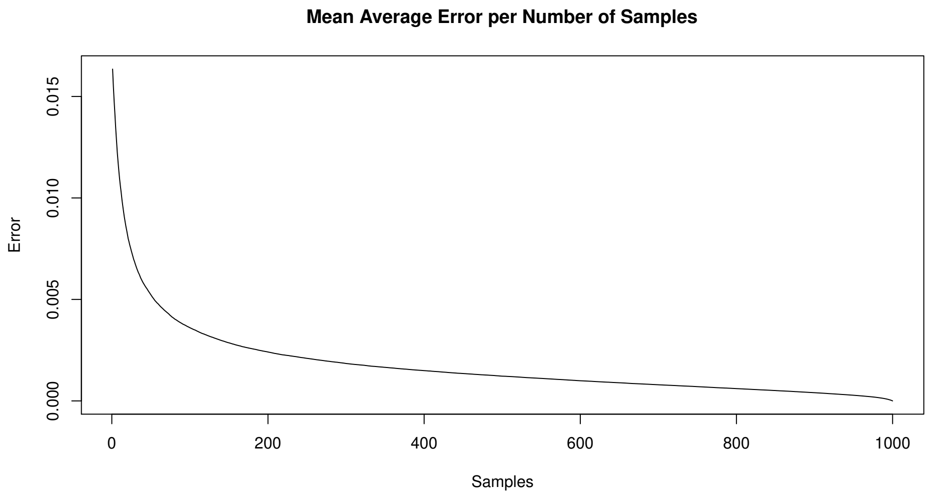

Figure 5.1: Mean Absolute Error of evolvability for number of samples compared to 1000 samples.

sampled evolvability to the strongest approximation (using the largest sample size). Even for a smaller OT problem, it is still computationally infeasible to calculate the correct evolvability. By selecting for fitness, higher fitness individuals are more likely to occur. If we select for evolvability, more evolvable individuals are likely to occur.

The experimental parameters are shown in Table 5.2. 10000 runs with different random seeds are completed for standard genetic programming (SGP), and 1000 runs for everything else. The maximum tree depth was raised for the 7th and 8th OT problems. These parameters are consistent throughout the experiments in this work. We define the mean absolute error of evolvability as follows:

$$\text{MAE} = \frac{1}{n} * \sum_{k=1}^{n} |e'_k - e_k| \tag{5.1}$$

where n is the number of runs, e'_k is the measured evolvability for 1000 samples, and e_k is the measured evolvability of the indicated number of samples.

Figure 5.1 shows that a reasonable approximation for evolvability occurs when the number of samples is about 100. Similar experiments for higher-order OT problems produces similar results (excluded for brevity). Since the purpose of evolvability for this system is to be used with an altered fitness function in order to guide selection, the required accuracy of sampling and modelling evolvability is proportional to the actual influence evolvability has on selection. Therefore, it is necessary to choose precisely how evolvability will guide selection in order to determine how many samples are sufficient to ensure accurate selection. This can be evaluated by using the modified fitness function, and compare which individuals are selected when using a reduced number of samples (or a model) for evolvability with individuals selected using a large number of samples. Discrepancies indicate that an individual was incorrectly selected.

5.1.2 Selection of Evolvability

This subsection describes the effectiveness of altering the fitness mechanism of SGP to consider evolvability in various ways. This will demonstrate the effectiveness of using sampled evolvability to improve GP. The significance of evolvability on selection will be monitored, so the optimal amount of selection can be used. Once the necessary conditions for the effectiveness of using evolvability in selection have been determined, it can be used to gauge the effectiveness of modelling evolvability.

We need to determine how to select for evolvability. To determine the optimal selection amount, we conduct the following experiment. We vary the SGP selection mechanism by using the sampled evolvability in various ways while keeping other experimental conditions consistent. There are several methods to guide selection with evolvability. One is a threshold for fitness; if the fitness of two individuals falls within a specific threshold, then we select the one with greater evolvability. Another is a weighted sum; we sum the fitness and evolvability, each weighted by a specified amount, and select individuals according to their weighted sum. We can allow a generational modifier for using a weighted sum; as the number of generations increase, we select less strongly for evolvability. Using a weighted sum and a generational modifier, we have, formally:

$$F' = \begin{cases} (f + \frac{e*p(g_{max}-g)}{g_{max}}) & \text{if} \quad g < g_{max} \\ f & \text{otherwise} \end{cases} \tag{5.2}$$

where F' is the adjusted fitness function, f is the standard fitness function, e is evolvability, p is the evolvability weight parameter, g_{max} is the evolvability ceasing generation parameter, and g is the current generation. This translates to the fitness function being modified by the probability of a change being positive multiplied by the weight parameter for the initial population, and where this modifier linearly approaches zero as the generation increases. Upon reaching zero, the modifier becomes zero for the remaining generations, rendering evolvability uninfluential. This is desirable because evolvability should become less significant as the number of generations increases, as standard fitness approaches optimal values. Maximizing standard fitness becomes the only goal when evolution completes. Eventually, we would just want to select for standard fitness. We conduct experiments for different OT problems under varying selection pressures (varying the weight and maximum generation parameters). The other experimental parameters are identical to the previous experiment, as shown in Table 5.2, however, the maximum initial depth and maximum depth are equal to 7 and 8 for the OT 7 and OT 8 problem, respectively.

Figures 5.2-5.10 show the average maximum fitness as the generation increases, for subsets of the tested problems. Confidence intervals of 95% as determined by

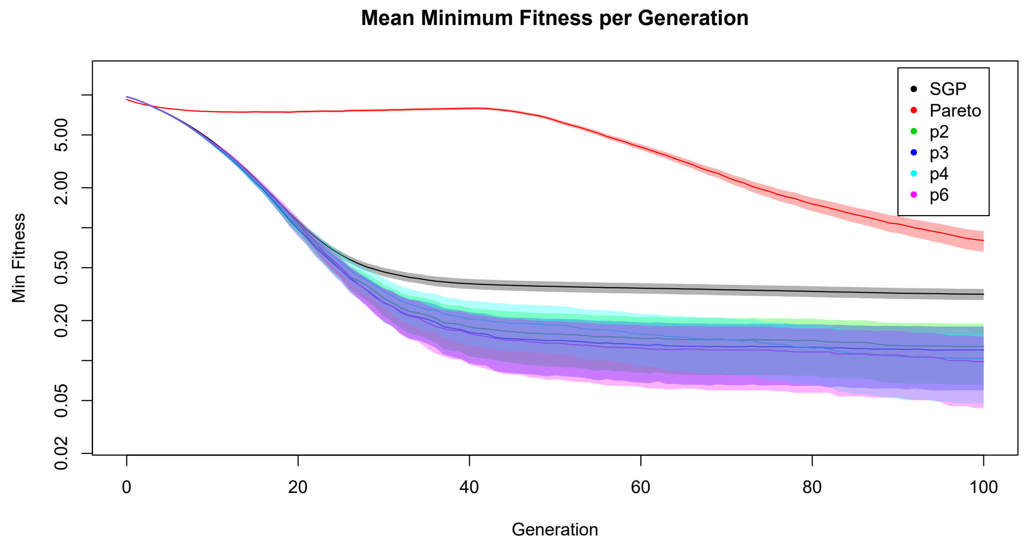

Figure 5.2: Fitness over generation for varying evolvability weights, as well as Pareto selection, for the OT 4 problem. Notationally "px" indicates the evolvability weight is x. OT4 is solved fairly easily by SGP on most runs, but using evolvability weights can find an optimal solution slightly more frequently. Pareto selection of evolvability and fitness simply performs worse. The evolvability weights differences do not provide significantly different results. Confidence intervals of 95% as determined by the Student's t-distribution are shown. Since more SGP runs are completed, its confidence interval is hardly visible.

Table 5.3: Probability of an incorrect selection comparing 100 samples with 1000 samples under various modified fitness functions over 100 runs.

Order	**p**	**g**	**Mean Selection Error**
4	7	10	0.34495%
5	10	N/A	2.6304%
6	5	N/A	3.6288%
7	10	N/A	4.338%
8	20	40	0.34230%

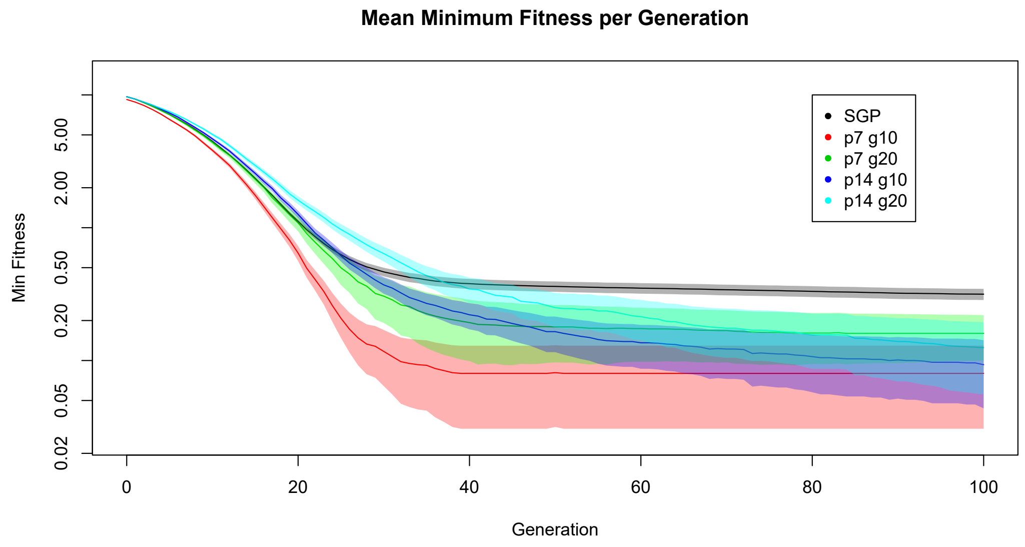

Figure 5.3: Fitness over generation for varying evolvability weights and evolvability ceasing generations for the OT 4 problem. Notationally "px" indicates the evolvability weight is x. Notationally "gx" indicates evolvability ceasing generation is x. OT4 is solved fairly easily by SGP on most runs, but using evolvability weights can find an optimal solution slightly more frequently. Ceasing evolvability selection earlier leads to equivalent eventual convergence to optimal solutions, but optimal solutions are found earlier.

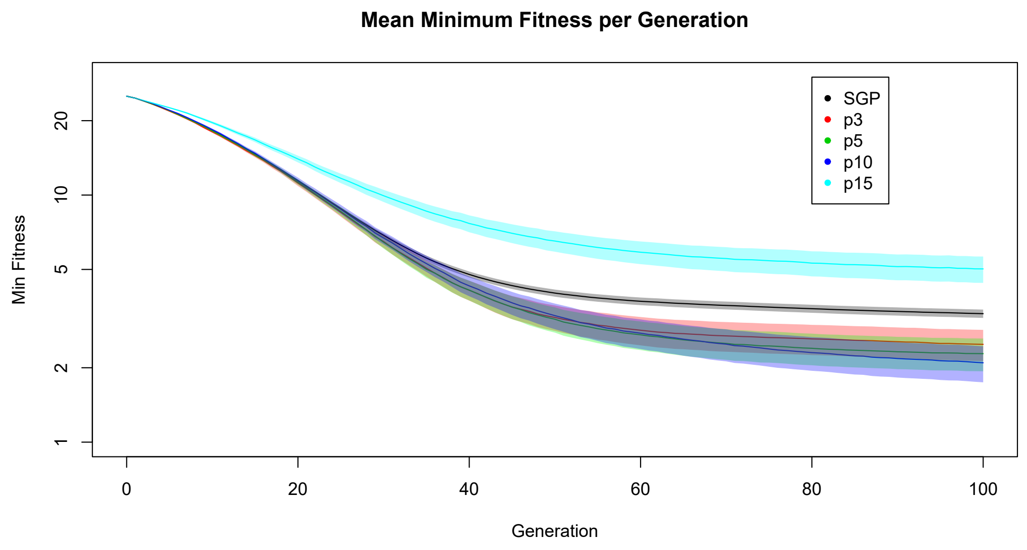

Figure 5.4: Fitness over generation for varying evolvability weights and evolvability ceasing generations for the OT 5 problem. OT5 is solved fairly easily by SGP on most runs, but using low evolvability weights can find an optimal solution slightly more frequently. Using high evolvability weights leads to underperformance; though the gap closes more quickly, it would take many more generations before it would outperform SGP.

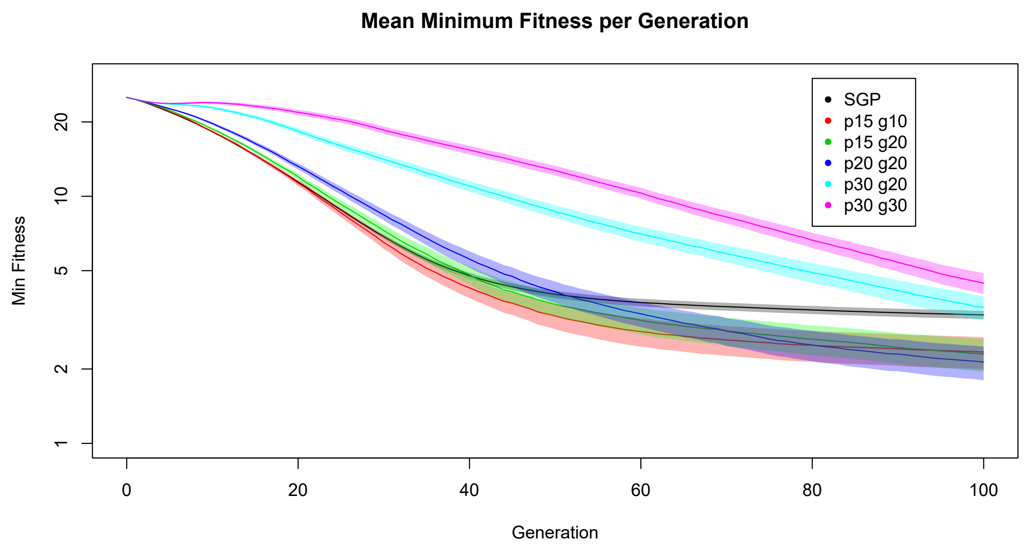

Figure 5.5: Fitness over generation for varying evolvability weights and evolvability ceasing generations for the OT 5 problem. Stronger evolvability selection yields slightly worse fitness performance. Strong initial selection pressure with a slower fade leads to a better fitness rate of change but is unable to catch up to the baseline in 100 generations. Low evolvability pressure with a faster fade outperforms SGP.

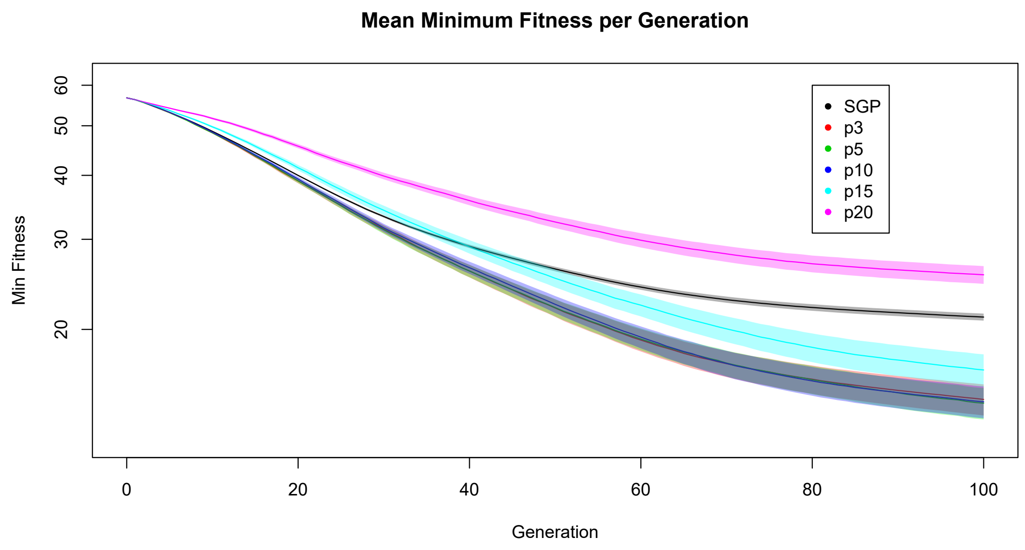

Figure 5.6: Fitness over generation for varying evolvability weights for the OT 6 problem. SGP has difficulty solving OT6 each run. Using evolvability weights allows GP to find an optimal solution slightly more frequently, but sufficiently large values cause fitness performance to drop.

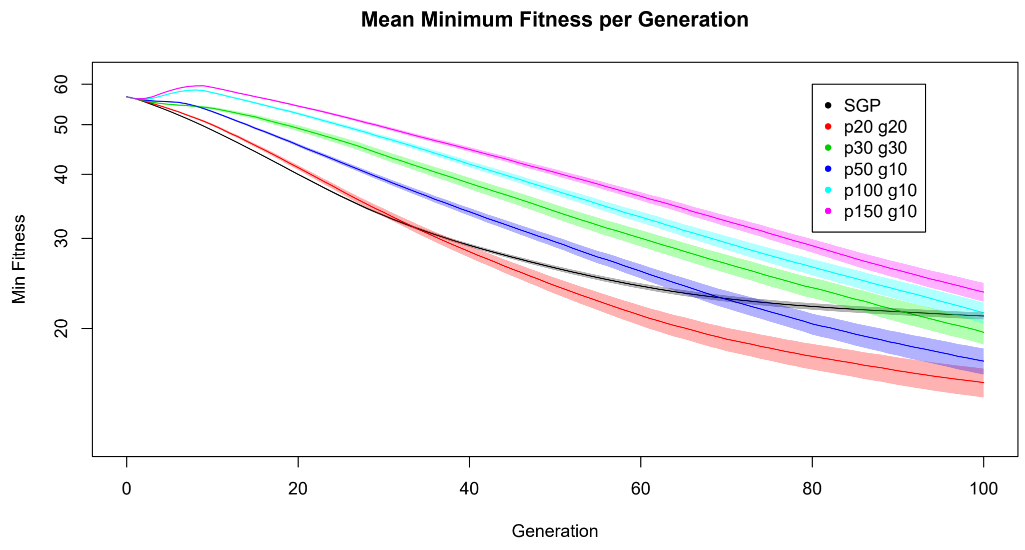

Figure 5.7: Fitness over generation for varying evolvability weights and evolvability ceasing generations for the OT 6 problem. Lower selection weights and lower evolvability ceasing generations yield better fitness performance. SGP outperforms evolvability selection in the short term, but evolvability selection continues to yield fitness gains even once it is no longer being selected for.

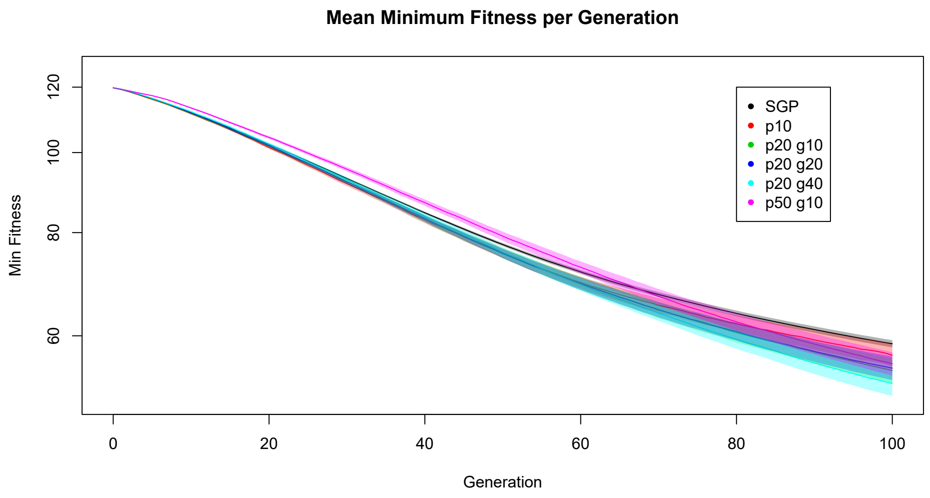

Figure 5.8: Fitness over generation for varying evolvability weights and evolvabilty ceasing generations for the OT 7 problem. The best performer uses a relatively large evolvability weight and a middling evolvability ceasing generation.

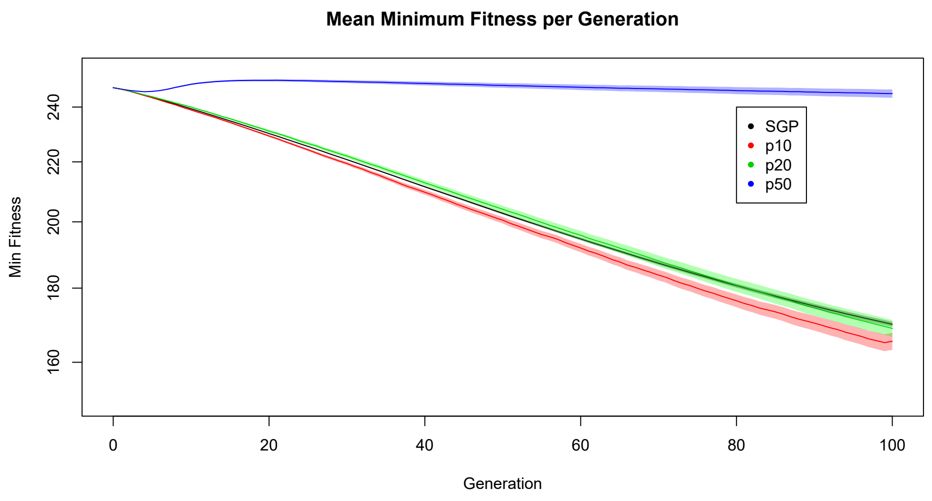

Figure 5.9: Fitness over generation for varying evolvability weights for the OT 8 problem. The best performer uses a relatively low evolvability weight.

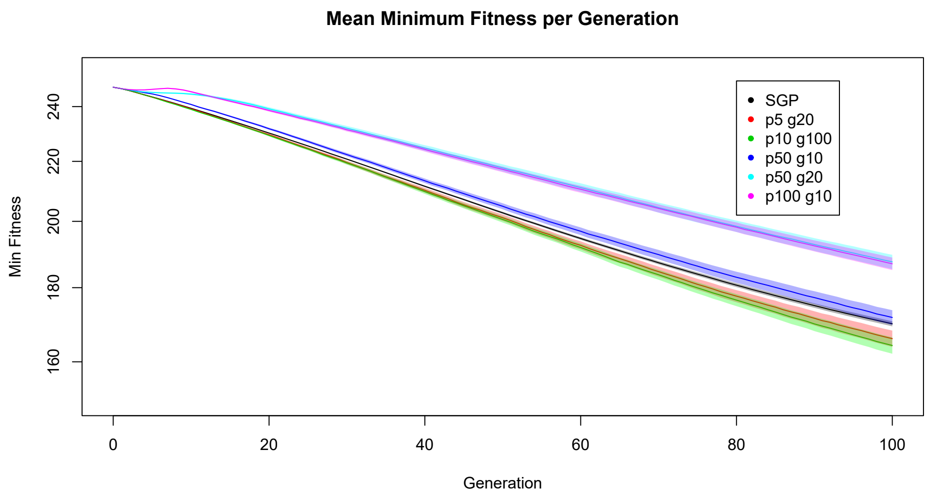

Figure 5.10: Fitness over generation for varying evolvability weights and evolvabilty ceasing generations for the OT 8 problem. The best performers use low evolvability weight.

the Student's t-distribution are shown. These results indicate that using modified fitness functions that use evolvability in addition to standard fitness outperform using standard fitness functions alone. The results demonstrate that *ap*EGP is able to achieve better or comparable fitness performance than SGP.

In particular, the results indicate the general appropriate proportion of evolvability to use for selection, as shown by the better performing selection pressures. Easier problems (the lower order OT problems) benefit from less evolvability pressure, in the form of lower evolvability weights or lower evolvability ceasing generations. Using a greater weight parameter is still useful, provided that a maximum generation parameter is specified, so fitness becomes more dominant as individuals approach higher fitness values. Using extreme values for a weight parameter, even tempered by small maximum generation parameter, does not produce fit results. For the higher difficulty problems, a greater emphasis on evolvability improves the results. We also note that selection based on Pareto-dominance, where fitness and evolvability are the two objectives, produces worse results than SGP. The fitness of Pareto selection follows a different trajectory than the other selection methods. Fitness improves slightly, then gradually worsens before improving more sharply. The initial random population is easy to improve upon with with both fitness and evolvability. Then, higher fitness individuals also have low evolvability, leading the population to appear to stagnate. However, while fitness stagnates, evolvability of the population is increasing, up until evolvability also stagnates. Then, improved fitness is more likely to be Pareto-dominant. Pareto selection may lead to a more evolvable population, but fitness suffers too much compared to other selection methods. The fitness trajectory may indicate that evolvability is more valuable in the middle generations. A subset of all the tested values for varying selection pressures are shown, for clarity.

We see in Table 5.3 that under the varying selection pressures, that 100 samples for evolvability differs from using 1000 samples less than 5% of the time. The tested selection pressures were some of the top performing selection methods for their order of problem, as shown in the previous experiments. Establishing a performance baseline for evolvability selection pressure allows us to proceed to model evolvability.

5.1.3 Modelling of Evolvability

We test the *ap*EGP system (see Figure 3.1) in this subsection. We demonstrated the effectiveness of using sampled evolvability in the previous section. We now build a model for evolvability and demonstrate its effectiveness. Firstly, we describe the attributes we use to build the machine learning models for evolvability. We record a number of attributes associated with individuals. These include generation, tree height, tree size, functional and terminal frequency, number of dormant nodes, dormancy ratio, previous standard fitness, fitness change, and standard fitness. These may all be recorded for each individual without onerous computational costs beyond standard fitness calculation. These attributes are subjected to attribute significance testing using WEKA, using the correlation-based filter method Correlation-Based Feature Selection [31], and further tests on WEKA classifiers. The most significant attributes were determined to be generation, size, function frequency, terminal frequency, number of dormant nodes, previous fitness, and fitness.

WEKA offers rapid use of many machine learning classifiers. In order to build a model, we provide training data and choose a classifier. We can generate training data by running SGP with the addition of evolvability sampling and selection; this will produce individuals which will be similar to those that will occur when using the model system, ensuring the models will be more accurate in practice. We can evaluate the effectiveness of the different models for evolvability by comparing the

Table 5.4: Probability of an incorrect selection comparing a multilayer perceptron model constructed with a varied number of training instances (themselves constructed under a varied number of evolvability samples) with 1000 evolvability samples under the 4th OT problem using the p7 g10 fitness function over 1000 runs.

Samples	**Training Instances**	**Mean Selection Error**
1000	2000	0.487524%
1000	4000	0.488446%
1000	8000	0.483173%
1000	40000	0.460605%

mean absolute error between them, also comparing this with the mean absolute error of the evolvability by varying number of samples. Various experiments indicate that a number of machine learning models are appropriate for this task, for they have similar mean absolute error rates. We select the multilayer perceptron, an artificial neural network (ANN), for verifying the effect of the number of training instances and the number of evolvability samples that are required for acceptable mean absolute error rates. Acceptable mean absolute error rates are those which indicate that erroneous selection will occur less than 5% of the time. We experiment to find the minimum required conditions to achieve this error rate. These experiments include varying the number of training instances, the number of evolvability samples used to generate those instances and measuring the frequency of selection error compared with 1000 samples of evolvability.

Once the conditions required for acceptable selection error rates have been determined, we test the system by comparing the top performing selective conditions in each OT problem, compared with SGP and the improvements made by using sampled evolvability, to indicate that modelling evolvability and modifying the standard fitness function, we can improve GP. This will indicate that modelling evolvability is viable.

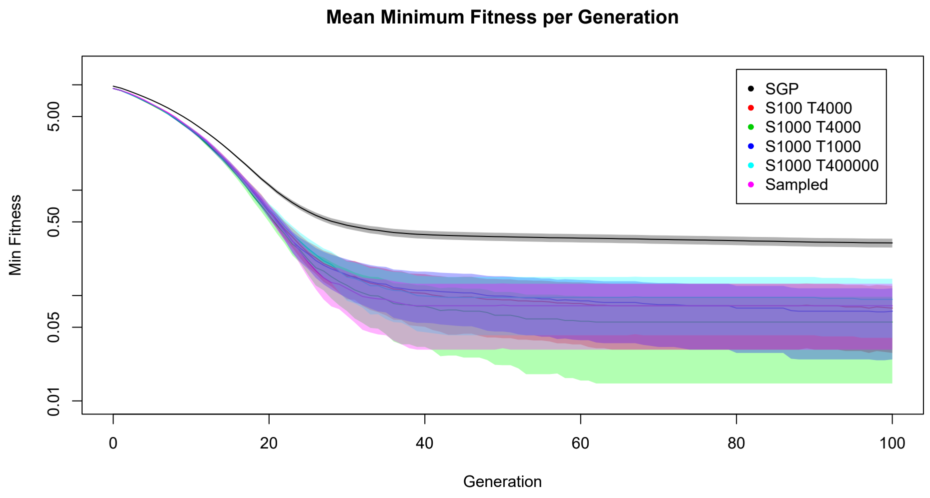

Figure 5.11: Fitness over generation for ANN models built from various amounts of training instances and various amounts of evolvability samples for the OT 4 problem. Notationally "sx" indicates the number of evolvability samples is x, and "Tx" indicates the number of training instances is x. The fewest samples and fewest training instances outperform SGP, and increases to either are not distinguishably better. They perform about as well as sampling evolvability.

We see in Table 5.4 that relatively few training instances are required to build an accurate model of evolvability. Very few selection errors are made when the evolvability used to train the model is accurate; when a large number of samples of evolvability are taken to generate the model. We see in Figure 5.11 that the models perform sufficiently well in practice. They are a statistical improvement over SGP and fare about as well as sampled evolvability. Even as few as 2000 training instances can build a successful model. Since a training instance is generated for each individual in the population for each generation, a single run with these settings generates 5000 training instances.

In Figures 5.12-5.15 we see that this trend holds in higher OT problems; modelling evolvability offers a statistically significant improvement over SGP, and performs about as well as using samples to calculate evolvability. Note that using models

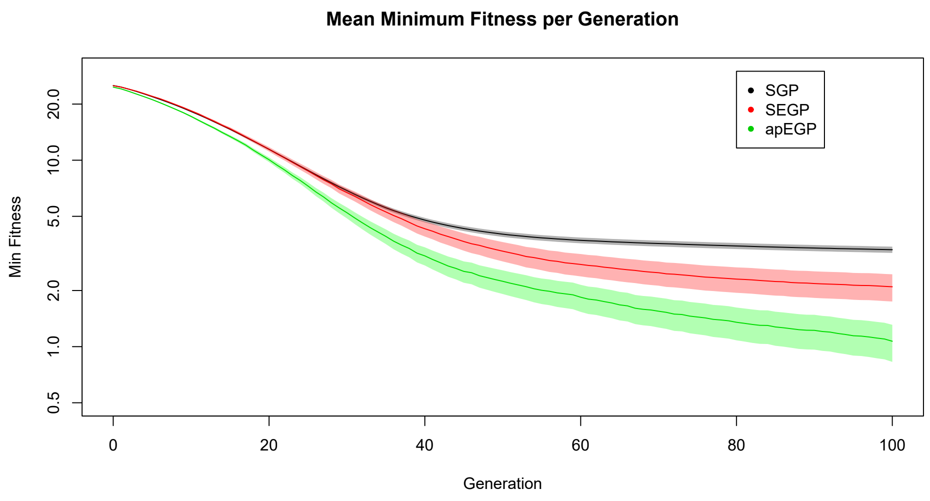

Figure 5.12: Fitness over generation comparing SGP, using SEGP and *ap*EGP for the OT 5 problem. Both evolvability use cases have a "p" value of 10. Modelling evolvability produces the best results.

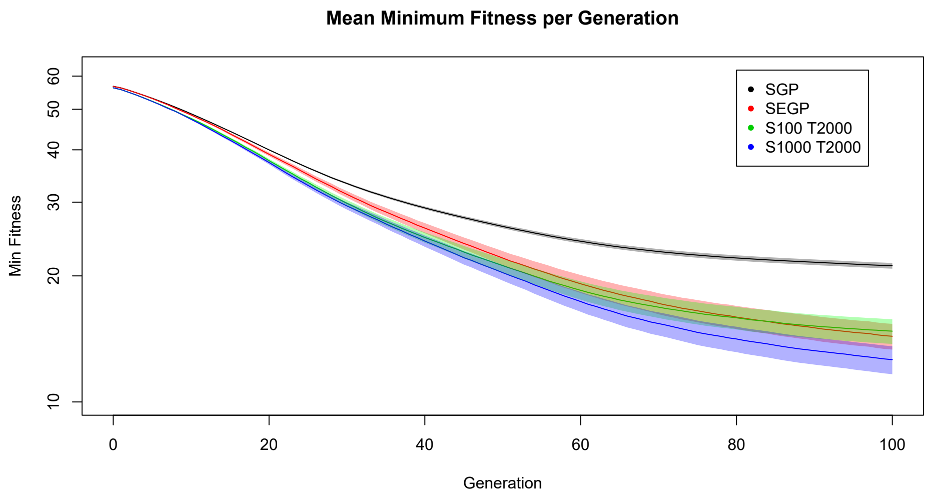

Figure 5.13: Fitness over generation comparing SGP, using SEGP and *ap*EGP for the OT 6 problem. Each evolvability use case has a "p" value of 5. Modelling evolvability produces the best results. More samples relative to OT4 produces better results.

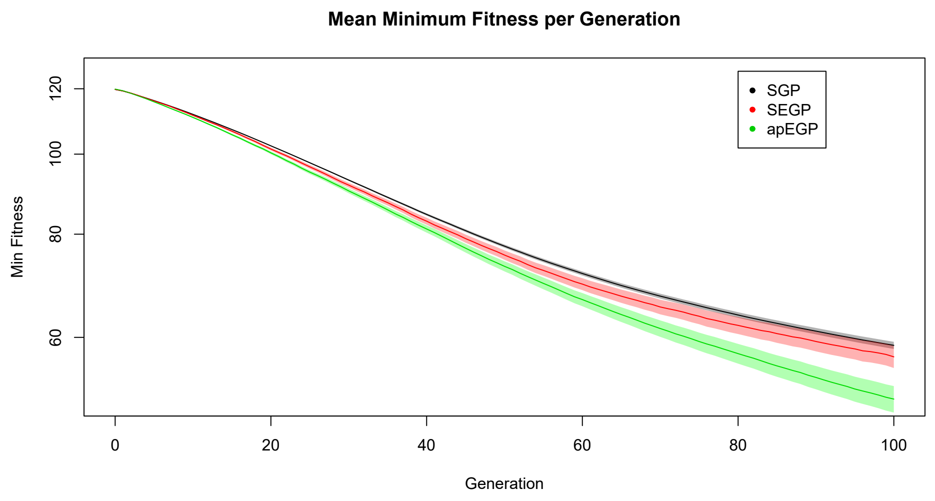

Figure 5.14: Fitness over generation comparing SGP, using SEGP and *ap*EGP for the OT 7 problem. Both evolvability use cases have a “p” value of 10. Modelling evolvability produces the best results.

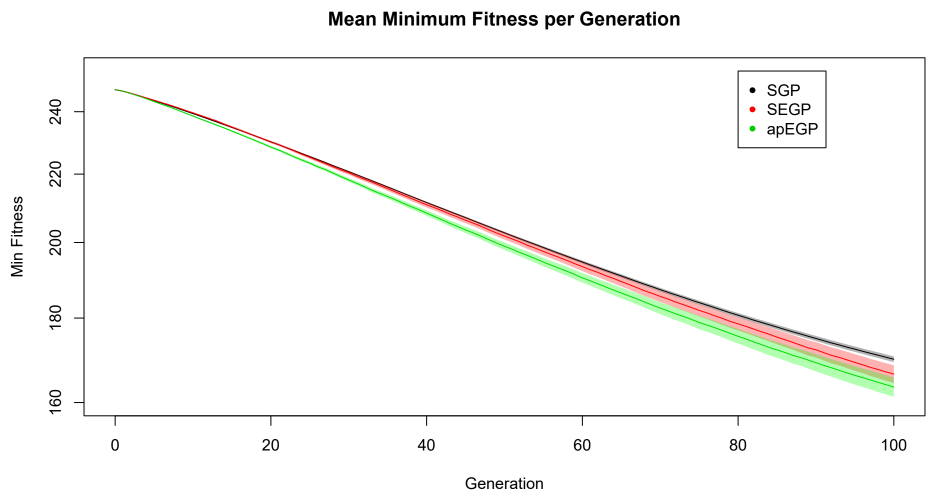

Figure 5.15: Fitness over generation comparing SGP, using SEGP and *ap*EGP for the OT 8 problem. Both evolvability use cases have a “p” value of 20 and a “g” value of 40. Modelling evolvability produces the best results.

built 1000 samples of evolvability even performs better than continually sampling evolvability 100 times for each individual.

5.1.4 Parallel Learning and Evolution

We have shown that models can be developed *a priori* which may provide benefits when used to alter normal fitness selection in order to emphasize evolvability, without significantly compromising efficacy when compared to sampled evolvability. We now move to implementing models that are developed during evolution. In particular, we move to a new system that can create evolvability models dynamically, MEGP. We need to train a new model for each new run, and alter those models as the population changes and generates new data. The existing system of Open BEAGLE Puppy and WEKA, used for *ap*EGP, does not meet our needs to train models in parallel to evolution, so we move to one based on EpochX and Encog. We verify that good evolvability models can still be developed under dynamic conditions. Further, we demonstrate that the additional computational time devoted to sampling evolvability and generating evolvability models produces sufficient gains in efficacy. As we move toward practical applications, we consider if the benefits of modelling evolvability are worth the costs. This subsection details the initial experiments with dynamic model building during evolution for the OT problem.

We conduct experiments on the OT 8 problem, with the experimental conditions shown in Table 5.5. We wish to verify the efficacy of the models in a dynamic environment. We compare the baseline, SGP, with SEGP and MEGP.

Using small selection pressures, according to some good performers of prior experiments, we find there is not much difference between SGP and using either SEGP or MEGP, as shown in Figure 5.16. Those results indicate that we should increase the se-

Table 5.5: Evolutionary and ANN parameters for parallel learning and evolution using the OT 8 problem.

Parameter	**Value**
Population Size	500
Crossover Probability	0.9
Tournament Size	6
Probability of Non-Terminal Crossover	0.9
Min Initial Depth	1
Standard Mutation Probability	0.1
Max Initial Depth	6
Mutation Max Regen Depth	2
Max Depth	9
Swap Mutation Probability	0
Initial Grow Probability	0.5
Probability to Mutate a Function Node	0.5
Number of Runs	1000
Maximum Generation	400
ANN Max Epoch	1000
ANN Early Stop Error Min	0.01
Evolvability Samples	100

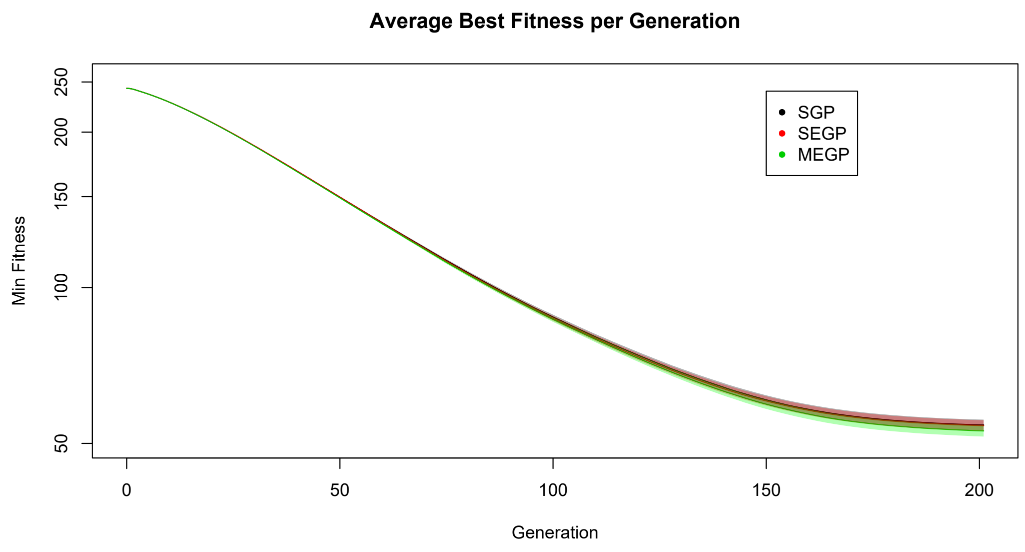

Figure 5.16: Fitness over generation comparing SGP, using sampled evolvability and modelled evolvability with a modified fitness function for the OT 8 problem, with evolvability selection pressure values of p20 g40. The evolvability pressure is too weak to amount to a significant difference.

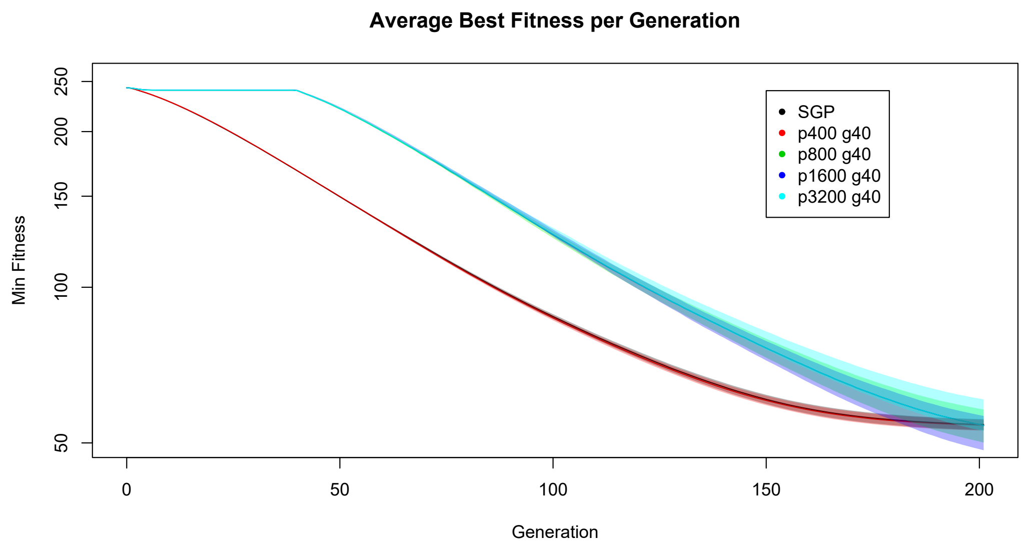

Figure 5.17: Fitness over generation comparing SGP, using SEGP for the OT 8 problem, with selection pressure values of g40 and various values for p. There is not significant difference when using large values for p, but a threshold exists. Using p400 is not significantly different from SGP. Evolvability selection needs sufficiently large p values to cause a difference.

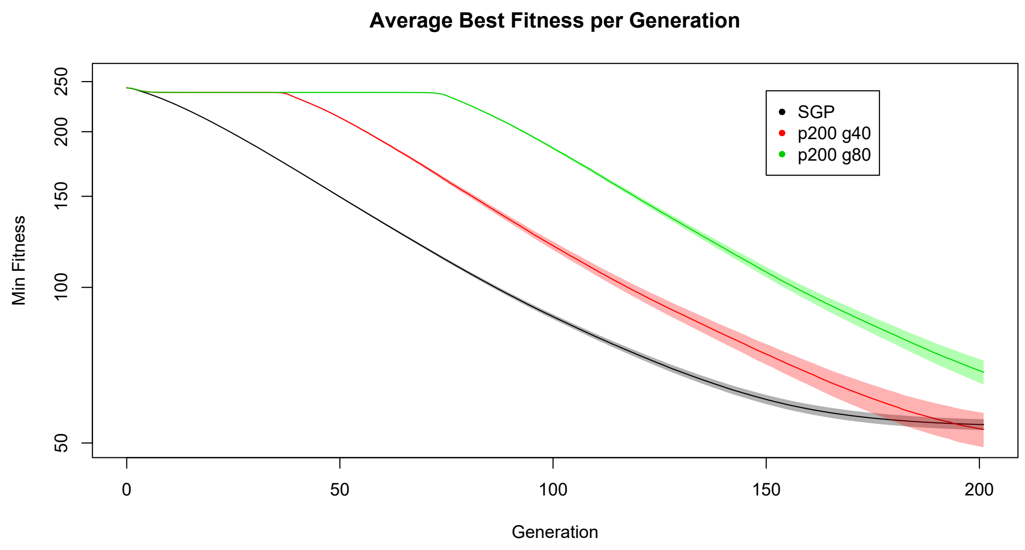

Figure 5.18: Fitness over generation comparing SGP, using SEGP for the OT 8 problem, with selection pressure values of p200 and various values for g. Allowing a higher value for g does increase the rate of change in fitness once evolvability selection is finished, but more generations are needed to catch up to a lower g.

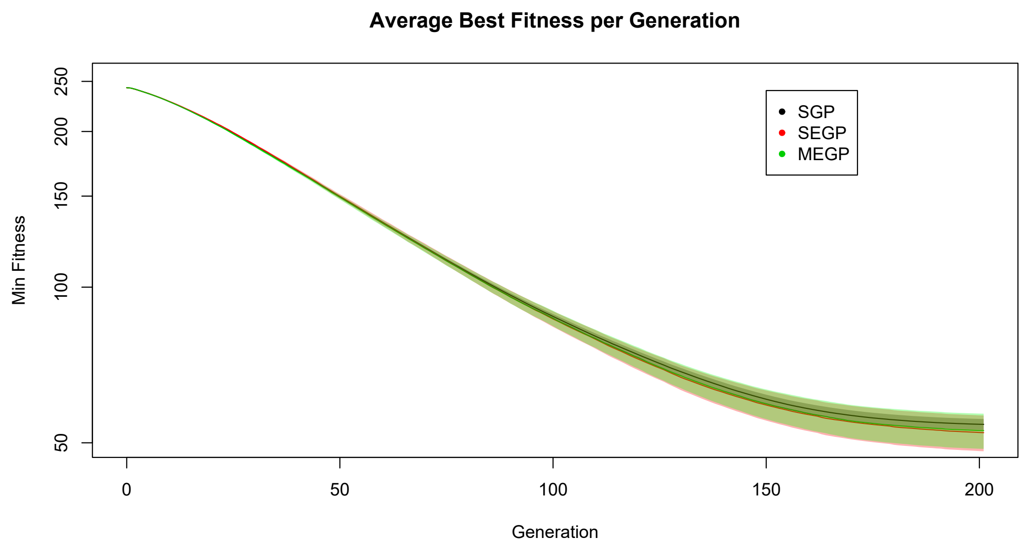

Figure 5.19: Fitness over generation comparing SGP, using SEGP and MEGP for the OT 8 problem, without evolvability decay. Both use p10 and g0. A low constant evolvability pressure does not cause much difference in fitness performance. MEGP does not degrade from SEGP.

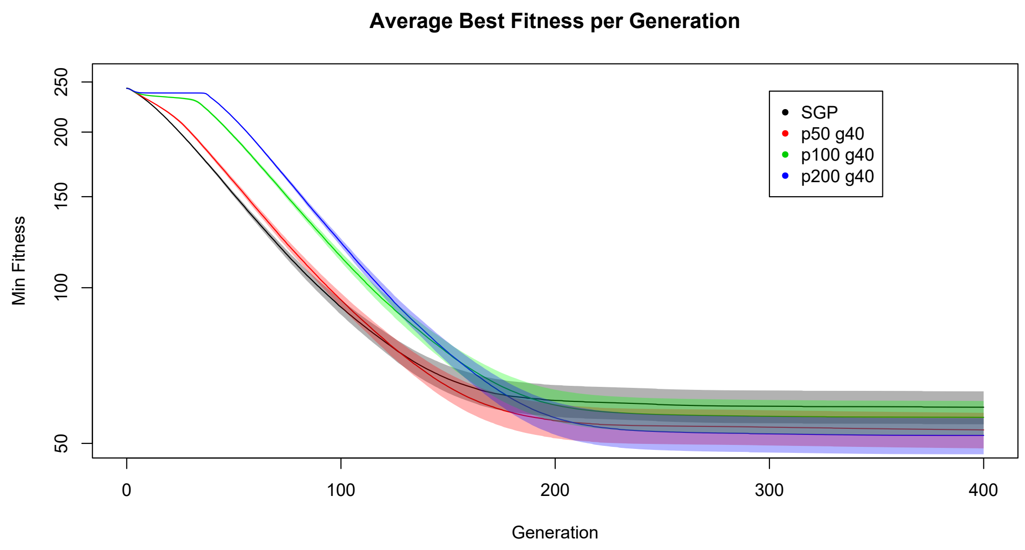

Figure 5.20: Fitness over generation comparing SGP and SEGP for the OT 8 problem with selection pressure values of g40 and varying values for p. Higher p values cause a fitness stall until generation 40. A p value of 50 performs well, for it does not face the plateau larger values face, while at least having an effect on fitness that lower values do not. Each are able to surpass SGP.

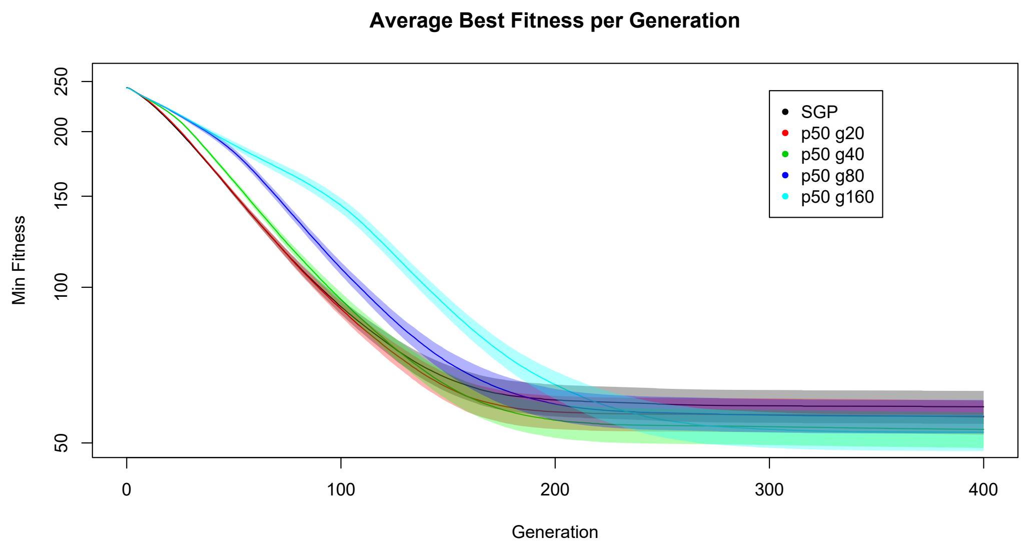

Figure 5.21: Fitness over generation comparing SGP and SEGP for the OT 8 problem with selection pressure values of p50 and varying values for g. The best performer is g40. Lesser values do not change enough from SGP, and greater values require too many additional generations to achieve comparable fitness.

lection pressure, in order to obtain more significant results. We investigate increasing the selection pressure, shown in Figure 5.17. Allowing a very strong pressure which decays gradually as the generation approaches 40, and then terminates entirely, we see that the population may actually benefit, though more generations are required to confirm that effect. However, the performance suffers a considerable amount in earlier generations. We consider a strong selection pressure with slower activating decays, in Figure 5.18. We see that the population favours a faster-activating decay, as the best fitness in the population does not rebound quickly enough to surpass the baseline if the onset of total decay is too slow. We can see that using high selection pressure is not sustainable without decay, as earlier generations will not improve their best fitness individual in a significant way. Attempting small selection pressure without any decay, we see in Figure 5.19 that constant low selection pressure does not significantly impact the results. We observe the long-term effects of moderate decay, with varying amounts of pressure in Figure 5.20. The relationship between selection pressure and fitness over time is demonstrated quite clearly; less selection pressure means better fitness early but slightly worse later. With enough generations to observe the change, we see that very strong selection pressure will produce the best results, with enough time. However, moderate pressure remains competitive, all outperform the baseline. The best performer is used as a basis for the next experiment, which varies the decay rate instead, with results shown in Figure 5.21. Once more, we see that stronger pressure through a slower decay leads to the best fitness in later generations, though faster decay is still competitive.

Figures 5.16-5.21 indicate that strong selection pressure with fairly rapid decay is most effective. Having to calculate evolvability or maintain evolvability models is fairly computationally expensive, so achieving similar results while only using evolvability for a portion of the total generations means we can save ourselves some

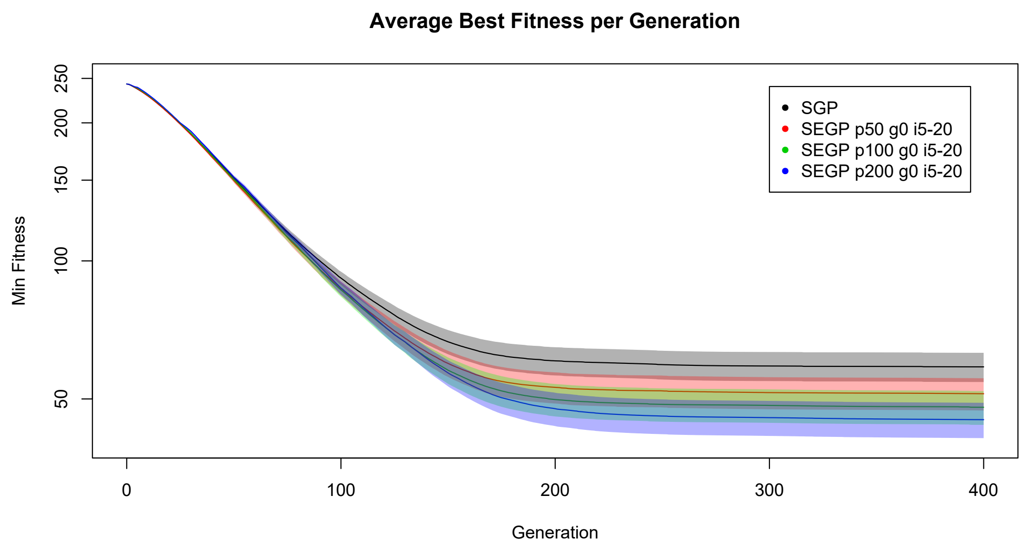

Figure 5.22: Fitness over generation comparing SGP and SEGP for the OT 8 problem with varying selection pressure values for p and cycling fitness with evolvability selection. The cycling notation $i5 - 20$ indicates 5 generations of evolvability use and 20 using standard fitness. Cycling outperforms previous experiments with OT8. Greater values for p are favoured relative to previous experiments without cycling.

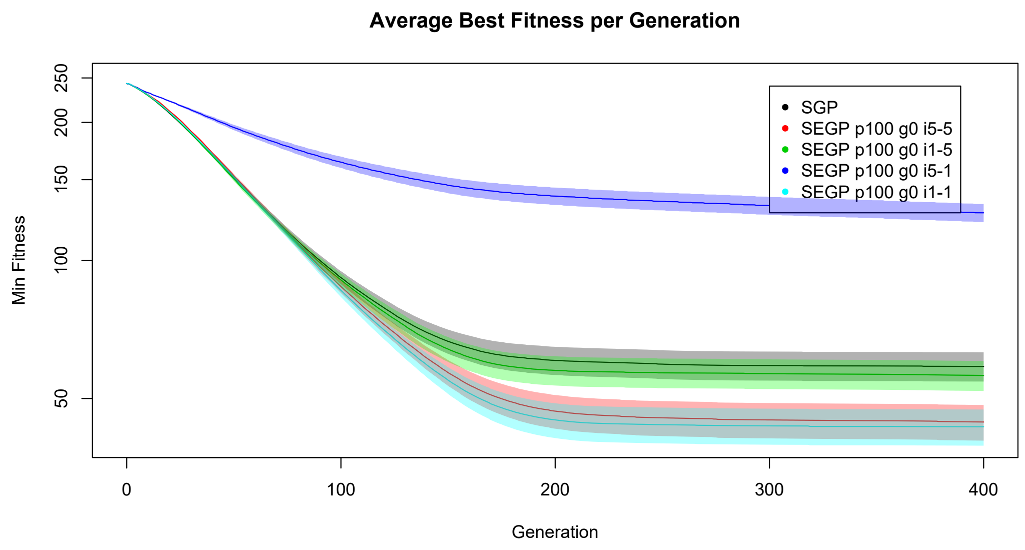

Figure 5.23: Fitness over generation comparing SGP and SEGP for the OT 8 problem with varying evolvability cycles. The most balanced cycles perform the best. Emphasis on fitness cycles causes little difference from SGP, and emphasis on evolvability cycles causes worse performance than SGP.

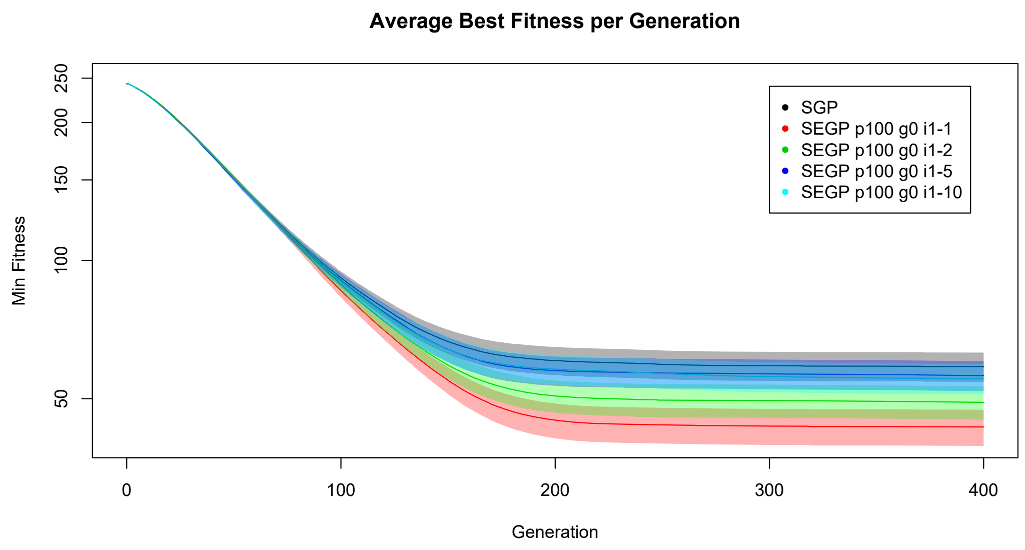

Figure 5.24: Fitness over generation comparing SGP and SEGP for the OT 8 problem with varying evolvability cycles. Equal emphasis on both fitness and evolvability produces the best results.

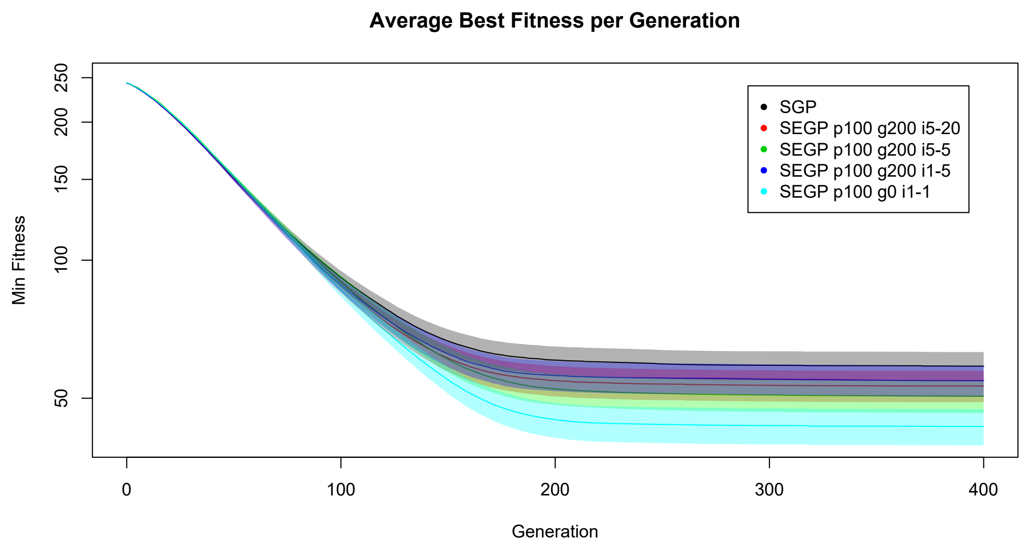

Figure 5.25: Fitness over generation comparing SGP and SEGP for the OT 8 problem with varying evolvability cycles and decaying evolvability pressure. Balanced cycles perform best. It is unnecessary to use decay when using cycles. The best performer uses rapid balanced cycles with no decay.

computation. However, if we only use evolvability for a small number of generations, and only initially, how many generations could we execute until the benefit is lost? Perhaps a better way of continuing is to use strong evolvability pressure for short periods of time, but not only for the first few generations, but interleaved in the following generations. To this end, we implement interleaved evolvability selection. We conduct a number of generations that uses evolvability selection pressure, and then a number of generations that use standard fitness, then cycling back. We conduct tests using a set number of cycling generations and varied amounts of selection pressure, without decay, in Figure 5.22. Using strong, interleaved evolvability pressure produces much better fitness values, and converges on them more quickly. Strong pressure is even more effective, and it only produces slightly worse results for earlier generations, in comparison to the gains made in later generations. Running experiments where the number of generations in the cycle varies, we see in Figure 5.23 using at least as many generations running standard fitness as there are evolvability-usage generations is required, or fitness suffers. The balance appears to be best when there are comparable amounts of each. Too few and evolvability is not enforced enough, so there are negligible gains. This is reinforced further with experiments varying only the number of generations where only fitness is used, in Figure 5.24. However, we need to consider the best performance ratio in the context of decay, as well. We see in Figure 5.25 that using decay is no longer necessary, as the function it was performing, the reduction in evolvability pressure over time, is compensated for by the interleaving. Non-decay performs better.

We have verified the efficacy of interleaved evolvability selection using sampled evolvability. We consider how interleaved selection will function with modelled evolvability. We conduct some initial experiments using similar parameters as our prior sampled evolvability experiments and find the results are quite similar, as shown in

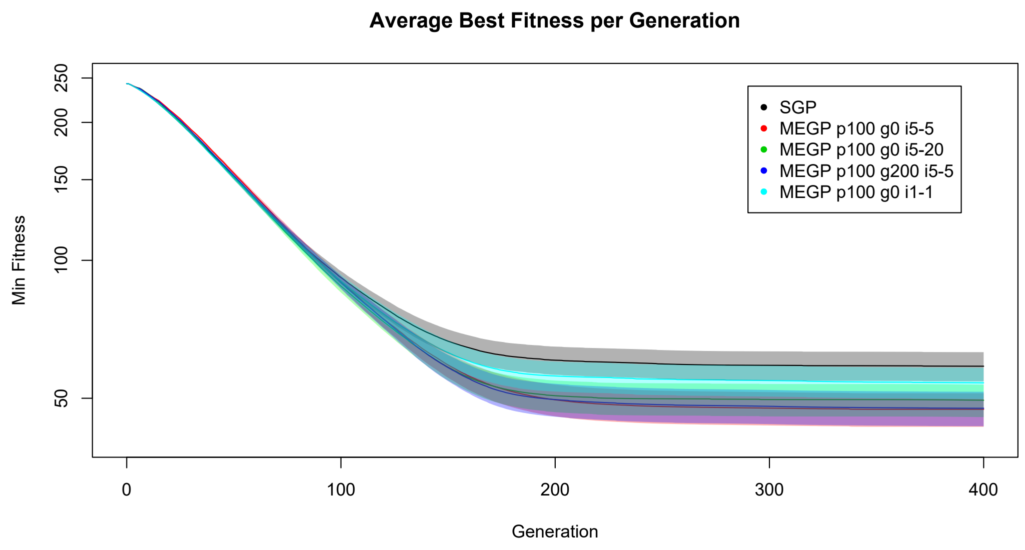

Figure 5.26: Fitness over generation comparing SGP and MEGP for the OT 8 problem with varying evolvability cycles. MEGP does not benefit from balanced cycles, relative to SEGP. Each performs about equally well, except for rapid cycles, which performs worse.

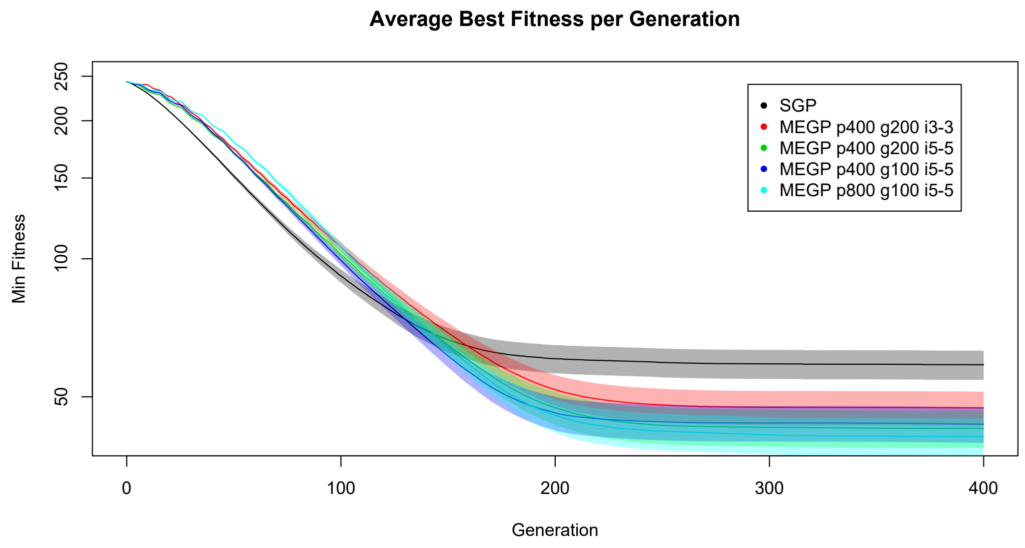

Figure 5.27: Fitness over generation comparing SGP and MEGP for the OT 8 problem with varying evolvability parameters. We exceed the current best MEGP performer with a greater p value and a lower g value. SEGP did not benefit from using evolvability ceasing generations when cycles are included, but MEGP benefits.

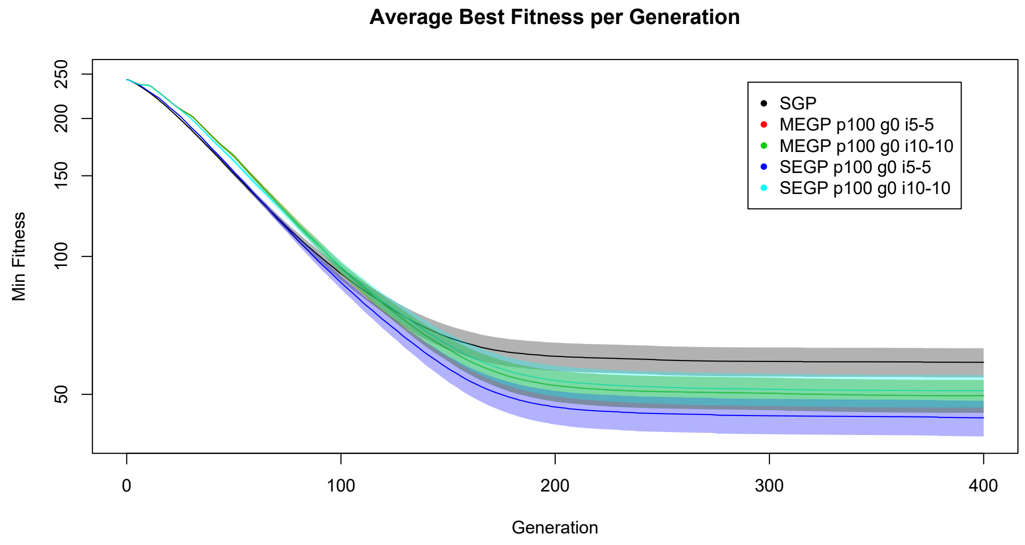

Figure 5.28: Fitness over generation comparing SGP, SEGP and MEGP for the OT 8 problem. MEGP is able to achieve comparable fitness performance to SEGP, and both surpass SGP.

Figure 5.26. We compare SEGP and MEGP together in Figure 5.28 and find comparable results, though the performance of the models suffers slightly. We observe the effects of using decay with modelled evolvability in Figure 5.27. We again find that strong selection works best, and decay is not necessary if we are interleaving evolvability selection with standard fitness.

Up until now, dynamic evolvability models are generated by considering the first several generations as training generations. Since we are sampling for evolvability as evolution occurs, in order to have labelled training instances for our ANN, we use the sampled evolvability to influence selection for those first few generations. Once we have sufficient training data, which under these conditions is about 5 generations, we switch over to using the evolvability model to influence selection and cease sampling evolvability. With interleaved use of evolvability, we are concerned that the model will degenerate more quickly, since the population transitions from the selection cy-

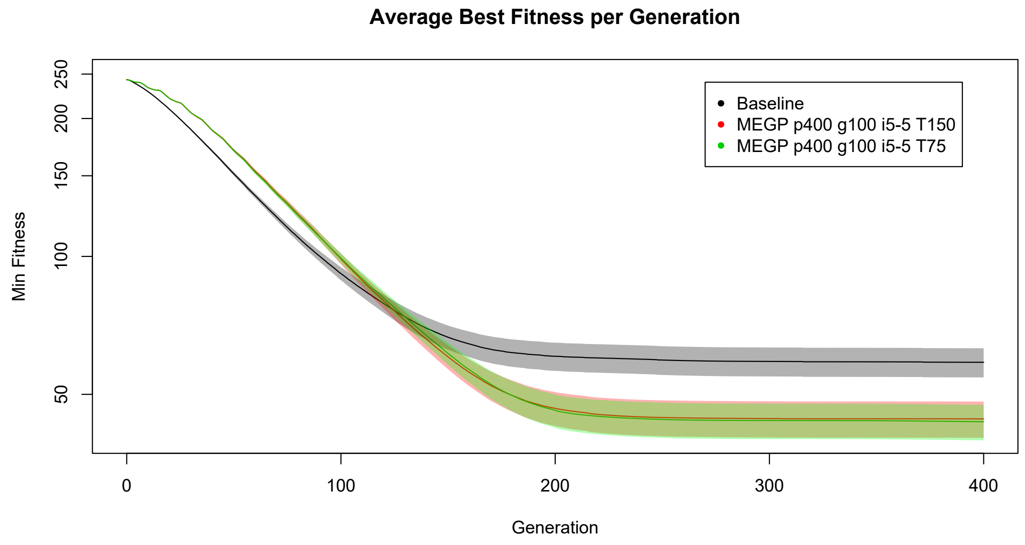

Figure 5.29: Fitness over generation comparing SGP and MEGP for OT 8 problem using varied difference-in-selection threshold values. Threshold is specified as the number following the "T". Using threshold values does not cause much difference. MEGP maintains a sufficiently good model without needing further training instances in future generations.

cling so rapidly that the model might not be as representative. This may be further exacerbated when there is no decay. Now we must consider whether our models are still representative of the data, considering the interleaved use of evolvability. We have noted there is a slight degradation in performance between sampled and modelled evolvability, as should be expected, since the models should not be as accurate as sampling, though we have computational gains. To that end, decay may still be useful, even with interleaved evolvability, as halting the use of evolvability will allow further computational gains, which may be merited, since fitness performance does not suffer significantly with its inclusion.

We also consider an additional metric to control the effectiveness of the model: the difference-in-selection threshold. We resample for evolvability every 20 generations and evaluate the individual selection difference between modelled and sampled

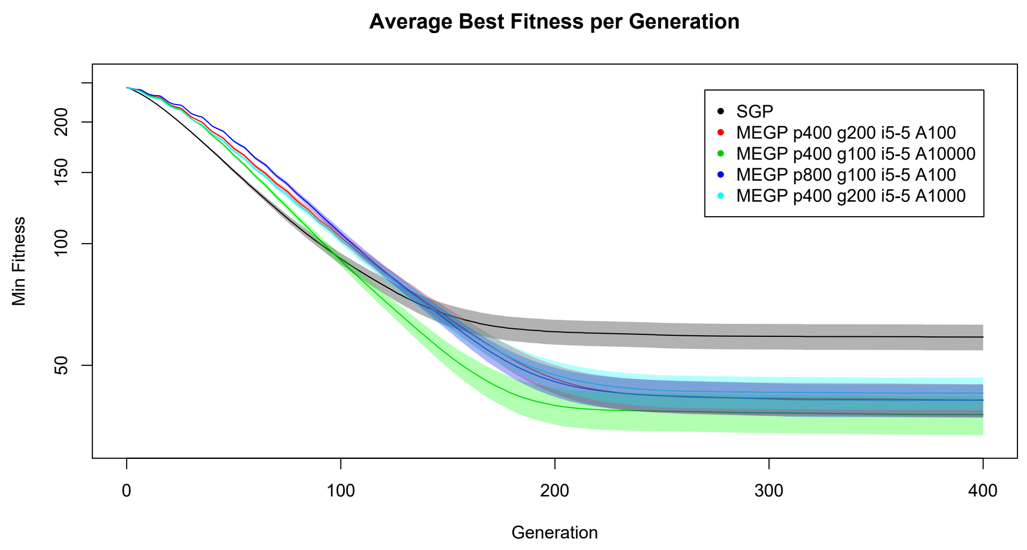

Figure 5.30: Fitness over generation comparing SGP and MEGP for OT 8 problem using varied ANN maximum training epochs. Allowing a larger number of ANN training epochs generates improved results. While other evolvability parameters vary within these results, they are comparable to generation 100, where allowing 10000 training epochs yields the best result.

evolvability. As the selection process is the only way we can consider the effectiveness of the model, these selection errors indicate that our model is not accurate enough and requires retraining with new instances. We consider the threshold value in Figure 5.29, and find that the values we are using do not make much difference, in terms of fitness performance. We consider altering some of the ANN parameters to encourage faster model development or more accurate model development. Varying the maximum number of ANN epochs at each training step, we see in Figure 5.30 that allowing longer training times for ANNs does indeed improve the fitness results. We must determine if this is worth the computational cost. We also note that strong interleaved evolvability selection can cause a wavy effect in the average best fitness that corresponds to the fitness selection cycle. We will return to the issue of model accuracy later when we look more closely at computational performance tradeoffs.

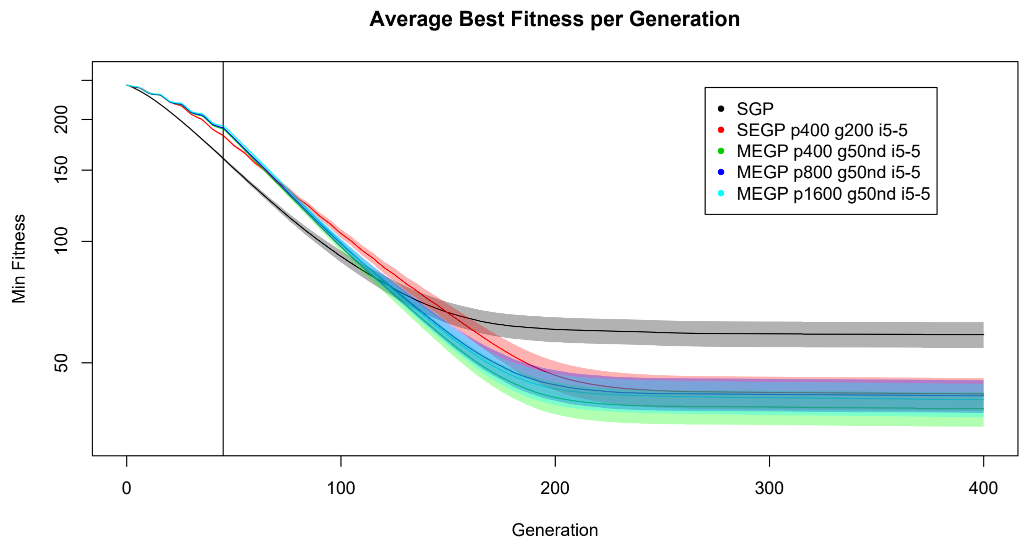

Figure 5.31: Fitness over generation comparing SGP and MEGP for OT 8 problem using various *p* values, without decaying evolvability pressure. As evolvability use is still stopped at a specified generation, we continue to use the *g* label, but we add "nd" to indicate that no gradual decay of evolvability pressure occurs. Using a sudden stop of evolvability pressure produces better results earlier, though fitness performance is eventually matched by later, gradual stops. Note the steeper slope of *nd* models, beginning at generation 45 (the last evolvability-selection generation because of cycling).

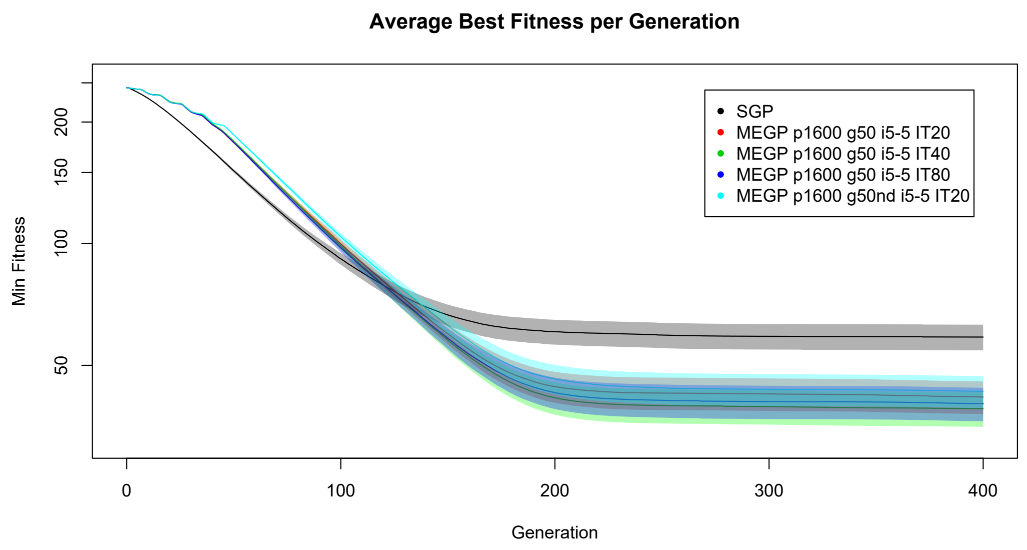

Figure 5.32: Fitness over generation comparing SGP and MEGP for OT 8 problem using various fitness-improvement thresholds. Fitness-improvement thresholds are designated by "IT". Forcing evolvability selection to be used when fitness has stagnates does not yield significantly improved results.

Considering again the effect of large selection pressure balanced by interleaving and potentially decaying evolvability pressure, we conduct an experiment to removes the traditional gradual decay of evolvability and instead introduce a hard stop point where evolvability is cut off. This allows computational gains by cutting off evolvability at a certain point while forcing higher pressure for the earlier generations. The results shown in Figure 5.31 demonstrate that the fitness performance does not suffer when using a rapid cutoff point for evolvability selection compared with a gradually decaying evolvability selection pressure.

We further consider another new method of dealing with potential selection errors, which is the number of generations that have occurred without improvement in the best fitness individual. A stagnant population means a low evolvability population. We introduce extra evolvability selection that occurs if the population's best fitness has not improved after a certain number of generations, with results shown in

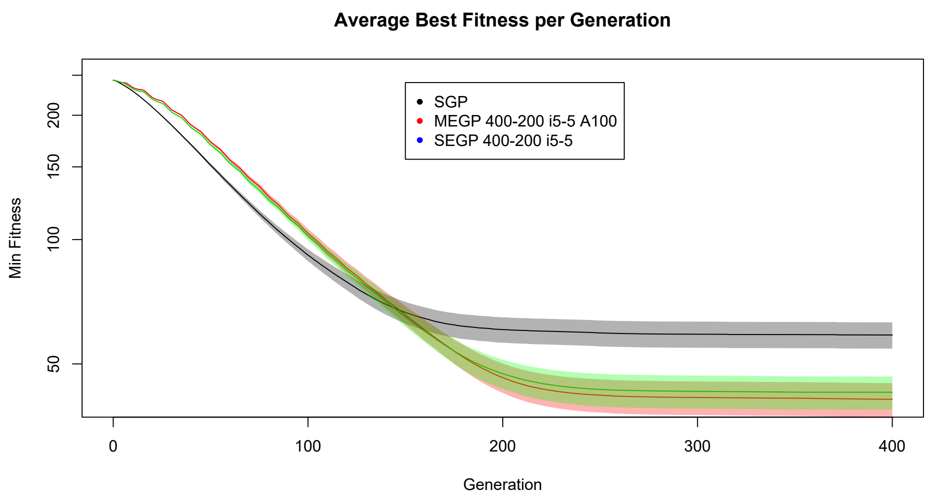

Figure 5.33: Fitness over generation comparing the best performers in the OT 8 problem. SEGP and MEGP both outperform SGP significantly. SEGP and MEGP perform comparably.

Figure 5.32. The results are not much improved.

The best performers are summarized in Figure 5.33 and their computational performance is summarized in Figure 5.34. Ultimately, the impetus of MEGP is improving upon the computational time required to use SEGP. SEGP requires onerous time because of the increased number of fitness evaluations, which is typically the main unit of work in GP. However, fitness evaluation in OT problems is simple and computationally inexpensive. The number of fitness evaluations is not as significant in determining the required computational time as other operations. As such, MEGP does not yield large decreases in computational time over SEGP. However, both require a fairly low amount of extra computational time compared to SGP relative to the increased number of fitness evaluations. Fitness performances for the best of SEGP and MEGP are comparable. Using MEGP does not lead to a significant decline in fitness performance. We are able to model evolvability effectively enough that doing so does not hinder the objective of increased fitness performance. These

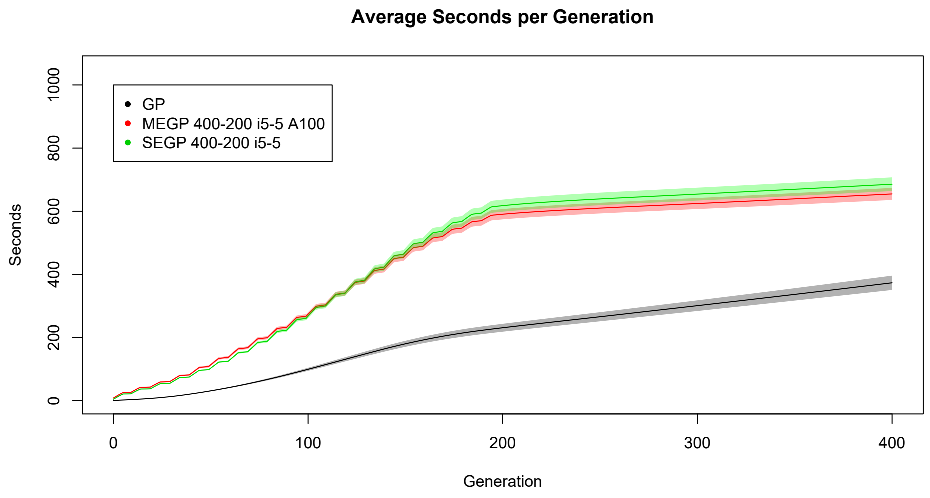

Figure 5.34: Time over generation comparing the best performers in the OT 8 problem. The best performing MEGP is only marginally faster than SEGP. OT is a unique problem in that fitness calculation is not nearly as onerous as other problems, so the gains made by reducing that number of fitness evaluations occur is marginal. Even with hundreds of samples per program per generation required for SEGP, SEGP only requires about 3 times the computational time.

experiments find conditions under which MEGP may perform as well as SEGP, and under which both outperform SGP. A natural extension of allowing more generations to SGP, to compete under similar computational time, would find its fitness performance plateaued and unable to reach the plateaus of SEGP and MEGP. As such, SEGP and MEGP both outperform SGP.

Chapter 6

Electricity Domain Experiments

In this chapter, we describe experiments and results regarding the New South Wales electricity demand domain (NSWED) [33]. These experiments explore a commonly used streaming data domain [8, 18, 84, 86, 100], examining the benefits evolvability modelling provides to the shifting problem environments.

The electricity domain consists of time, price, and consumption information, with class labels indicating if the price of electricity has increased or decreased relative to the previous time period. This dataset demonstrates concept drift: a change in the underlying patterns which create the data, which means a useful model trained on earlier time periods may not function effectively on later time periods. We conjecture that a model which may adapt more easily to changing paradigms should be effective in compensating for concept drift. These properties make the NSWED domain a good choice for evaluating the effectiveness of the evolvability modelling system.

First, we review the general qualities streaming problems possess, and other required background information for evaluating streaming problems. Then, we discuss experimental designs and their results.

6.1 Background

Streaming data is characterized by its real-time generation [8]. In static problems, fitness is defined *a priori*, but in dynamic problems fitness may change over time [17]. Streaming data is becoming ubiquitous with the rise of the Internet and increased data processing speeds. Traditional data processing methods can no longer cope with the volume of data that is now being produced in various enterprises [12, 68]. The growth in volume generation is outpacing the growth in processing power. Traditional machine learning algorithms may have made assumptions about loading entire datasets into memory, which may not be possible now that datasets have grown so vast. Streaming environments are a key problem in Big Data analysis. Big Data is characterized by the 3Vs or 4Vs, which are *velocity*, *volume*, *variety* [53], and either *value*, *variability*, *virtual*, or *veracity* [45, 99]. In streaming environments, the most relevant characterization is *velocity.* Instances may only be processed a few number of times, or even once before they are discarded, as new data streams in rapidly. In time-sensitive domains, old data quickly becomes less valuable, as it loses predictive power when the underlying data distribution changes in fundamental ways. For example, consider electricity demand: sudden events (such as a heat wave, which requires more power to run air conditioning units), not otherwise captured by data (perhaps temperature is not explicitly recorded in training instances), may change the results such that previous models are now inaccurate. These are some fundamental problems when using streaming data, that must be considered. It may be useful to have a highly evolvable population that could shift models quickly, to react to a changing problem. We describe previous work with streaming problems, describe the electricity domain in more detail, and describe important concepts that are used in the experiments, in the following subsections.

6.1.1 GP and Streaming Problems

Streaming data are a sequence of boundless, time-ordered instances [70]. The nature of data streams poses computational problems: it is difficult to even store data arriving at such a high speed, much less process and analyze [22]. Instances may arrive at varying rates and are unknown *a priori*. Instances may have a short shelf-life if the means to store them is limited. Even if storage is not so limited, the instances may have a declining predictive value for future instances, as the underlying distribution may change gradually over time, or even suddenly. Instances must be processed quickly, so that responsive action may be taken quickly. Knowledge of class labels may be limited or delayed. These characteristics pose significantly more challenges than static domains, but also an opportunity. Genetic programming (GP) may have advantages over other machine learning paradigms, considering a population's ability to adapt to new instances with less regard to older instances. Previous instances may have been used to select a population up to the current generation, but they may be quickly abandoned from fitness considerations. This may indicate that a more evolvable population will be able to adapt to the changing circumstances, presented by streaming data problems, more quickly than other machine learning paradigms. Nevertheless, GP does not have a great amount of research to date, pertaining to streaming problems.

Chongstitvatana conducts experiments using GP in a dynamic environment, which were simulated robot traversal environments [13]. There were difficulties in transferring success of GP in robot learning problems from simulation to the real world. Robot movement speeds were too low to conduct training in the real world, and slight differences in the simulation and real world would often have the simulated GP solution break down when exposed to reality. In order to compensate for slight environmental changes, Chongstitvatana uses perturbations in the simulated environment, which improves the robustness of the solutions. Previous work with GP focuses on static environments, as dynamic environments posed additional challenges to deal with. However, Chongstitvatana exploits the dynamic nature of the environment in order to improve the solutions.

Yan and Clack conduct an early attempt at using GP for a continuous learning environment: hedge fund portfolio optimization [95]. They build upon previous efforts in GP to improve the diversity of the population, with the intent of increasing adaptability. The diversity of standard fitness and of phenotypic behaviour is encouraged. The diversity of standard fitness is easily encouraged by partitioning fitnesses into segments of similar fitness. Phenotypic diversity is more difficult to measure, but for the particular domain of hedge fund portfolios, correlation and contracorrelation of individuals can be calculated by considering the Return of Investment of individuals at particular points in time. That is, phenotypic diversity can be preserved by preserving individuals whose Return on Investment falls when others rise. Contracorrelated Return on Investment pairs may have similar overall fitness, but their phenotypic differences are still easily measured in this fashion.

Riekert *et al.* [72] look more directly at dynamic environments. They propose a slight alteration to GP, dubbed adaptive genetic programming (AGP), and compare its performance with GP and gradient descent on artificial dynamic domains. AGP differs from standard genetic programming (SGP) by using adaptive control patterns and adaptive elitism. Adaptive control patterns send the population into hibernation, meaning no more genetic operations when some specified fitness is reached. Then, only the fittest of the population is evaluated on new data until its performance has deteriorated some specific amount, when evolution resumes. A specific amount of the best individuals in the population are saved when using elitism. The percentage of the population saved by adaptive elitism varies as the result of fitness improvement; if best fitness improves in a generation, more elitism is encouraged, saving more elite individuals, and if best fitness deteriorates, then elitism is reduced, allowing for more exploration. Crossover rates are adjusted similarly. Mutation rates are cyclic, adjusted when an environmental change is detected. Culling of the worst performers is also used. The dynamic domains investigated were artificially generated. Points in each domain space were generated according to particular patterns, and then those patterns were redefined after a set number of points were generated, then those same points are reclassified according to the new patterns. Sliding time windows could then be used to simulate either sudden concept shift or gradual concept drift, important concepts in dynamic domains that will be explored in a proceeding subsection.

Smith [78] conducts some experiments with GP and streaming data, showing that GP can adapt to concept drift. Synthetic and real class domains are tested under varied circumstances, to answer how GP should be initialized for streaming data, what parameters are ideal for evolution, when GP populations should be reset in response to concept drift, and whether evolution should continue when no drift is occurring. Large populations had faster adaptation. Crossover, mutation and elitism rates required a balance to achieve the best results and were domain dependent. Resetting the population when concept drift occurs was detrimental since the existing population would adapt faster than a new population would to new data. Continuous evolution outperformed evolution that was only triggered when concept drift occurred. These results provide useful experimental parameters for streaming problems using GP.

Heywood [35] provides a survey of both evolutionary and non-evolutionary model developments for streaming data classification tasks. A series of works by Vahdat *et al.* [84, 85, 86] use GP to classify streaming data with a variety of improvements to

Table 6.1: Summary of properties of the NSWED domain.

Size	45312
Input Attributes	8
Classes	2
Class Balance	42.5%:57.5%
No-Change Classifier Accuracy	85.3%
Sliding Window Size	100
Sliding Window Overlap	20

SGP which address problems that occur when dealing with streaming data. Further developments of those improvements were introduced by Khanchi *et al.* [46, 47]. These works address some core problems posed by streaming data. Label budgeting is a particular focus. In streaming problems, acquiring accurate labels for an instance may impose significant costs, so reducing those costs or otherwise only paying them for the most helpful instances can improve the efficacy of modelling the stream. There is further attention on which instances should be archived for future use, and how to manage data imbalances considering label budgets and archiving. Further, the works make use of Symbiotic Bid-Based GP [55], which functions by maintaining two populations: symbiont and host. A symbiont individual acts as a program which produces a scalar, and is assigned an action, which for classification problems, takes the form of a class. Each host individual indexes a subset of the symbionts. To evaluate an instance for a host, each program of its associated symbionts is evaluated, which will produce bids for each different class. The highest bid is the predicted class. These works provide the most rigorous examination of GP applied to streaming problems under both synthetic and real domains, considering their complexities and experimental results.

6.1.2 Data Description

The NSWED domain [33] consists of 45312 instances representing two years of data. Properties of the NSWED domain are summarized in Table 6.1 The NSWED datasets include 8 attributes and two classes. The attributes are date, day, time period, New South Wales price, New South Wales demand, Victoria price, Victoria demand, and the transfer rate between those Australian states. The class label indicates if the change in price from the previous instance is greater or less than the moving average of the last 24 hour time period. The private market prices are set every 5 minutes by matching the demand for electricity with the cheapest possible combination of electricity from the available power stations, which have their own individual price schedules. Different magnitudes of electricity production cost varying amounts at each station. Prices are determined by supply and demand. Primary factors determining electricity demand in New South Wales are season, weather, and time of day. The principle factor determining electricity supply is the number of on-line generators. Generally, electricity prices are subject to recurring variation in the form of seasons and day of the week, and short-term variation due to events such as weather fluctuations. The dataset begins when the market was first privatized, so it is expected that the suppliers become more sophisticated as they learn the intricacies of the market. Furthermore, on a specific date, the New South Wales market is linked to a neighbouring Australian state, which moderated supply. Another unexpected disruption occurred when that link was suddenly severed for a short time. All of these details indicate that the electricity domain is a useful streaming domain, where new data is constantly generated, and the underlying distribution affecting prices changes both gradually and swiftly over time.

Though the NSWED dataset is commonly used in streaming data domains, it may have some problems [8, 100]. Particularly, it is possible to construct a naïve classifier that classifies an instance according to the previous instance, which performs well. This is called a No-Change classifier. If price is falling right now, it is likely it will be falling in the next half hour. If price is rising right now, it is likely it will be rising in the next half hour. If the data was distributed uniformly (without consideration to time), the No-Change classifier would achieve an accuracy of about 51%. As the data stands, it achieves an accuracy of 85%. If a classifier is unable to achieve an accuracy of at least 85%, it should not be considered competitive. Further, if it can only achieve comparable accuracy, the classifier may simply be modelling temporal dependence. Classifiers should be compared to how well they perform on a per-instance basis with the No-Change classifier, to see if their behaviour is significantly different than such a naïve classifier.

6.1.3 Concept Drift

Concept drift is when changes in context induce changes in the target concept [94]. Considering electricity price datasets, concept drift occurs when the price of electricity changes due to some external factor. Concept drift is what makes streaming problems difficult; if the underlying data model never changes, we could just train a model based on all our existing data, and it would predict the outcomes of unseen data well enough. However, unpredictable changes in context may occur which will alter the price, in ways existing data could not be used to predict. An effective learner needs to be able to distinguish when concept drift is occurring. Furthermore, some concept drift may actually occur with regularity, such as seasonal demands on the electrical grid. An ideal learner should be able to adapt quickly, be resistant to noise, and treat reoccurring contexts [83].

Changes in data context can occur in different forms, relating to the rate of change in the underlying distribution [27]. Sudden concept shift is essentially an abrupt shift from one context to another [86, 98]. In the NSWED dataset, that may be represented by the connection of the Victoria electrical grid to the New South Wales grid. Incremental drift describes a slow change in context, a small amount in one direction at each time interval. For electrical grids, that may represent a change in weather. Gradual concept drift described a more oscillating change in context, where the next context is phased in a more discrete pattern than incremental drift. Reoccurring concept drift described a pattern of context shifting back and forth. Finally, all these types of concept drift must separate from outliers, where the underlying distribution of the data has not changed, but may appear as if it has.

There are many mechanisms to help compensate for concept drift, and Gama *et al.* [27] propose a thorough taxonomy based on memory, change detection, learning methods, and loss estimation. Briefly, memory is the consideration of which data instances to save and which to forget, change detection is how we notice when concept drift is occurring, learning methods involve how we may best generalize the data considering concept drift, and loss estimation is how to accurately define how well the model is performing considering concept drift. Older instances may be unhelpful if they no longer represent the new distribution of data, though if concept drift is recurring, some should be preserved. Identifying what type of concept drift is occurring, if any, will inform decisions in the other taxonomies. Learning methods consider how to adapt the model when facing concept drift and maintain multiple models to make a combined prediction (ensemble learning). Loss estimation is altered when concept drift occurs, since the environmental feedback may no longer be valid (if we are unable to use true data labels).

Tsymbal [83] proposes there are three fundamental approaches to deal with concept drift: instance weighting, instance selection, and ensemble learning. In instance weighting, older instances are given more or less significance according to their age, favouring newer instances. In GP, this can be done by according more fitness to newer instances. Instances may also be weighed according to their estimated predictive value of the current concept if concept drift is explicitly detected. In instance selection, only relevant instances are used in training, where the methods are designed to identify relevant instances. Ensemble learners maintain multiple concept descriptions through multiple models, using weighted voting to determine which concept is most relevant.

6.1.4 Prequential Accuracy

There is an issue in determining the accuracy of a model using streaming data, in comparison to static data. We demand that the model predict incoming, unseen instances. In static domains, we can simply remove some portion of the data, using that removed portion for testing (unseen), with the rest used for training (seen). Since all data comes from one underlying distribution, if we simply remove instances in a uniformly random way, both training and testing data will represent approximately the same distribution. The testing error is an unbiased estimator of the true error. In streaming domains, we cannot partition the data in this way, as the underlying distribution of new instances may differ from prior instances, due to concept drift. The testing error in streaming domains using this method, the holdout method, is a biased estimator [27]. In streaming domains, we are primarily interested in predicting the labels of new instances. As such, it would be useful to define accuracy in terms of performance of the model on only the newest instances. Predictive sequential (or prequential) accuracy [15] is a widely used performance metric for streaming domains [86]. Each instance is first used in testing (besides some initial training instances), which evaluates the effectiveness of the model, and then is incorporated into the training data. Data is evaluated as it is collected. However, keeping all previous data available as training data might blunt the properties of newer instances. If concept drift has occurred, are not newer instances preferable for training? The sheer volume of prior instances means that newer instances have less bearing on the developed model, and this only increases as more instances are collected. Fading factors and forgetting are advantageous, as they increase the impact of newer instances [26, 51]. One such method, sliding windows, is discussed in Subsection 6.1.6. Prequential error is simply one minus the prequential accuracy.

6.1.5 Kappa Plus Statistic

Streaming data can have high temporal dependence such that a good predictor of an instance's label is the label of the previous instance. This is true in the NSWED domain; this No-Change in label predictor has 85.3% accuracy. A model that performs worse than a No-Change classifier is not very interesting. A useful way to compare models with the No-Change classifier is the Kappa Plus statistic [8]. The Kappa statistic [14] is a useful way of comparing models that have imbalanced classes, in that it compares the accuracy of a model to one that assigns classes randomly according to the probability of any one instance belonging to a class. It is easy to build a model that has a high accuracy when your dataset has a severe class imbalance; just always predict the class that occurs the most, ignoring all other information. A classifier that predicts class according to the frequency which it appears in the dataset is called a Chance classifier. The Kappa statistic can demonstrate that high accuracy is misleading. The Kappa statistic is given by:

$$\kappa = \frac{a_0 - a_c}{1 - a_c}, \tag{6.1}$$

where a_0 is the model's accuracy, and a_c is the Chance classifier's accuracy. This is adapted to the Kappa Plus statistic as follows:

$$\kappa^+ = \frac{a_0 - a_{nc}}{1 - a_{nc}}, \tag{6.2}$$

where a_0 is the model's accuracy, and a_{nc} is the No-Change classifier's accuracy. A model that only performs as well as the No-Change classifier will have a κ^+ value of 0. One that performs worse will have a κ^+ value of less than 0, and one that is better will have a κ^+ value greater than 0. A perfect classifier will have a κ^+ value of 1.

6.1.6 Sliding Windows

Sliding windows is a data management technique used for streaming data. Simulating streaming is necessary for the electricity dataset; the entire dataset is provided at once, though in practice, instances are only generated every 30 minutes. We could arbitrarily train a model on the whole dataset, and treat the domain as if it were static. This would not generate a useful model, and the model methodology could not be generalized to practical streaming problems. Instead, we treat the data as if it is incoming through a stream, small portions at a time. Sliding windows denote which data instances are visible. At initialization, only the first few instances are used to train a model. The window of available instances "slides" along future data, allowing new data to be used in training, while older instances "slide" out of consideration. In GP, the window may slide each generation, or be allowed multiple generations to evolve before sliding. The sliding window may overlap with previous windows or contain only new instances. They may be of static size, or adaptive sizes [7]. They may be combined with archiving policies, to track older instances that have been deemed useful, though they may no longer belong in the sliding window. Larger sliding windows require more computation and may lose sensitivity to recognize concept drift. Too many older instances blunt the predictive power of fewer newer instances. Smaller sliding windows may be too sensitive, attempting to adapt to a new concept where no concept change has occurred, just noise or outliers.

6.1.7 Label Costs

Labelling instances can have cost. For static domains, all the necessary labelling may have done already, though strategies exist to make the most out of existing labels. For streaming domains, it may take extra computational (or human) resources to label fresh instances. Active learning systems attempt to overcome label costs by working directly with human operators to label as few instances as possible [75]. Vahdat *et al.* [86] explicitly address the label budget issue as it pertains to streaming problems, using a stochastic querying scheme. A subset of each sliding window is sampled and labelled, which replace old instances according to an archiving policy. Archiving keeps a limited number of older instances according to diversity and age. Label costs for the NSWED domain are small, as human intervention is not required to calculate average electrical grid prices. Further, it could be calculated inexpensively, at the end of each demand period.

6.1.8 Electricity Domain Related Results

Bifet *et. al* conducted a survey of the literature, and found that only six of the sixteen reported accuracies were greater than that of the No-Change classifier. None of those used GP. The best performer using the entire dataset at that time was 88.5%, by an ensemble classifier called Learn++.CDS [18]. Vahdat [86] use of a GP variant with label budgets and achieve accuracy of about 60%. Castelli *et al.* [11] use of GP to forecast electricity demand. Instead of the NSWED domain, they used a comparable domain, electricity demand in Italy during summer months. It performed better than artificial neural networks (ANNs) or a comparable ensemble classifier.

6.2 Experiments

Streaming domains bring their own challenges, but also an opportunity. Maintaining an evolvable population might improve the populations' ability to adapt to concept drift. We have demonstrated in Chapter 5 that both sample-evolvability genetic programming (SEGP) and model-evolvability genetic programming (MEGP) can be effective in a static domain. Accuracy was increased with only modest increases in time complexity. We conduct these experiments to show that MEGP can be effective in a real-world domain. Evolvability lends itself to concept drift; a more adaptable population should be able to reduce error that is incurred when concept drift occurs. However, streaming domains are necessarily more dependent on low computational complexity; the speed at which data arrives may be so swift that there are not enough computational cycles to devote to evolvability sampling. Modelling becomes more important, to realize time gains. First, we must show that SEGP can be effective in the electricity domain, even if it requires too much extra time to realize the accuracy gains. Previous experiments, in the OrderTree (OT) domain, identified some suggested evolutionary parameters. The nature of streaming data requires some additional analysis, however. The most appropriate evolvability selection parameters indicated for the OT domain were very strong evolvability selection initially, interleaved, and gradually decaying evolvability selection. Interleaved and decaying evolvability may not be appropriate when concept drift occurs. Maintaining an evolvable population, that may adapt quickly, should be preferable. The first experiments here investigate under what evolvability pressures can sampling evolvability thrive. Then, we show that MEGP can be comparably effective, but with a considerable reduction in computational complexity. The experiments, their supporting figures and results are summarized in Table 6.2.

6.2.1 Best Use of Previous Parameters

First, we examine the effectiveness of the previous best evolvability selection parameters, as determined by top performers in the OT domain. We define some basic GP parameters in Table 6.3. The increased complexity of the Electricity domain compared to higher OT problems translates to some higher-demand parameters: population size, tournament size, and maximum depth are all increased to ensure that a solution can be evolved, though it adds to computational complexity.

We conduct 100 runs of 200 generations, where each run uses its own random seed. We use an overlapping window size of 100, sliding 20 instances each generation. The baseline consists of SGP, with no evolvability selection, sampling, or prediction thereof. The top-performing sampling selection uses mutation positive-probability evolvability, with an evolvability weight of 400 decaying to 0 by the 200^{th} generation, with interleaved evolvability selection for 5 generations after 5 generations without evolvability selection. The top-performing model selection uses the same, with the ANN training epochs limited to 100, with a cut-off error of 0.01. 100 evolvability samples are used.

We compare results using fitness, prequential accuracy, computational time, and

Table 6.2: Summary of NSWED experimental results.

Figures	**Parameters**	**Purpose**	**Results**
6.1-6.4	Evolvability weight, evolvability cycling generations	Evaluate SEGP and MEGP using the previous best performing parameters	The previous best performing parameters are outperformed by SGP. Evolvability cycling is inappropriate for a dynamic problem.
6.5-6.7	Evolvability weight, magnitude evolvability, crossover evolvability	Evaluate different evolvability metrics with SEGP, evaluate required number of samples	Outgoing-crossover magnitude evolvability is able to produce better results than SGP.
6.8-6.12	Evolvability weight, magnitude evolvability, crossover evolvability	Evaluate if evolvability selection is better than noisy selection.	SEGP and SGP both outperform noise. SEGP consistently scores higher than a 0 Kappa+ where SGP fails.
6.13, 6.14	Evolvability weight, magnitude evolvability, crossover evolvability	Evaluate required number of samples to build an effective model and required ANN training parameters	MEGP benefits from more evolvability samples, ANNs do not require much time to build a good model.
6.15-6.18	Evolvability weight, magnitude evolvability, crossover evolvability	Evaluate SEGP and MEGP with the best known performers.	MEGP is still able to produce results comparable to SEGP, requiring only a fixed time increase to build a good initial model.

Table 6.3: GP evolutionary parameters for the electricity domain.

Parameter	**Value**
Population Size	120
Crossover Probability	0.9
Tournament Size	6
Probability of Non-Terminal Crossover	0.9
Min Initial Depth	3
Standard Mutation Probability	0.1
Max Initial Depth	6
Mutation Max Regen Depth	2
Max Depth	10
Swap Mutation Probability	0.1
Initial Grow Probability	0.5
Probability to Mutate a Function Node	0.5

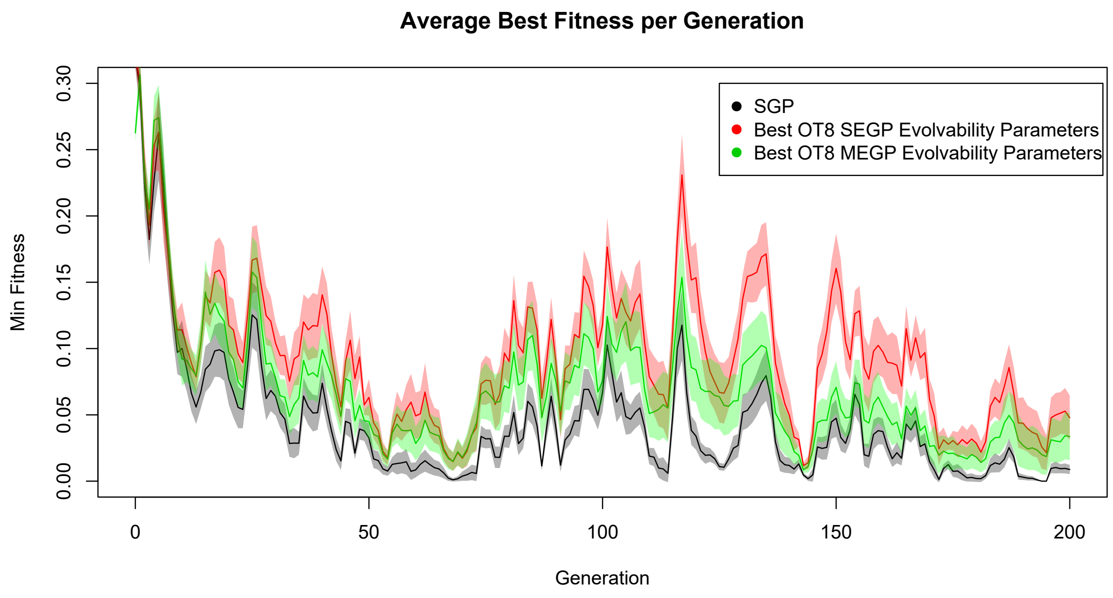

Figure 6.1: Fitness over generation comparing SGP, SEGP, and MEGP on the Electricity problem using the best evolvability selection parameter performers of the OT problem. Neither perform as well as SGP. MEGP causes less difference than SEGP. Confidence intervals of 95% as determined by the Student's t-distribution are shown.

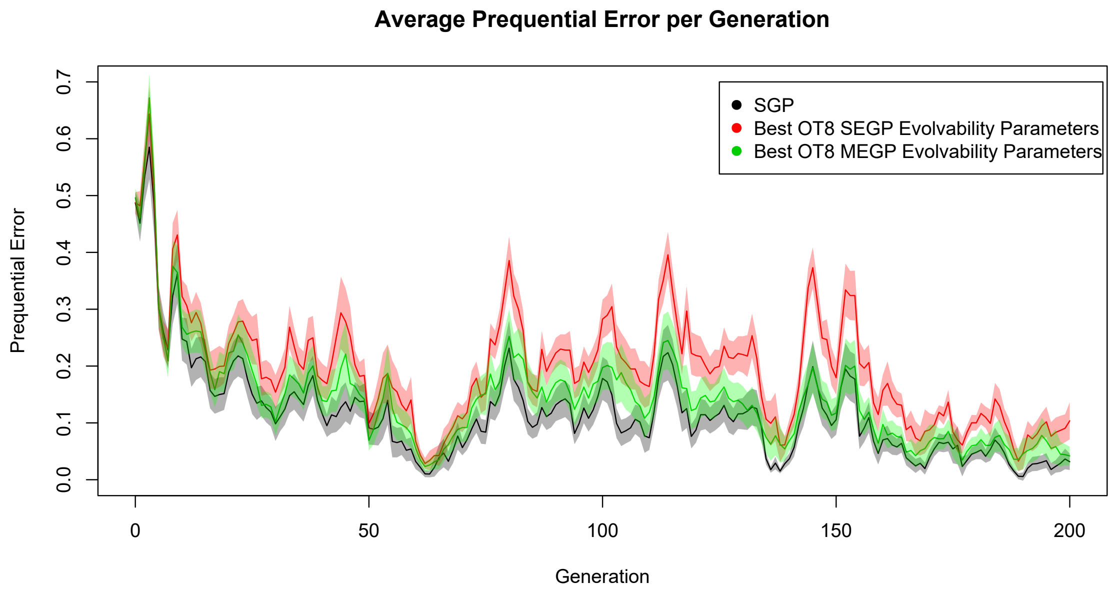

Figure 6.2: Prequential error over generation comparing SGP, SEGP, and MEGP on the Electricity problem using the best evolvability selection parameter performers of the OT problem. Prequential error follows the pattern of fitness for SGP, SEGP, and MEGP.

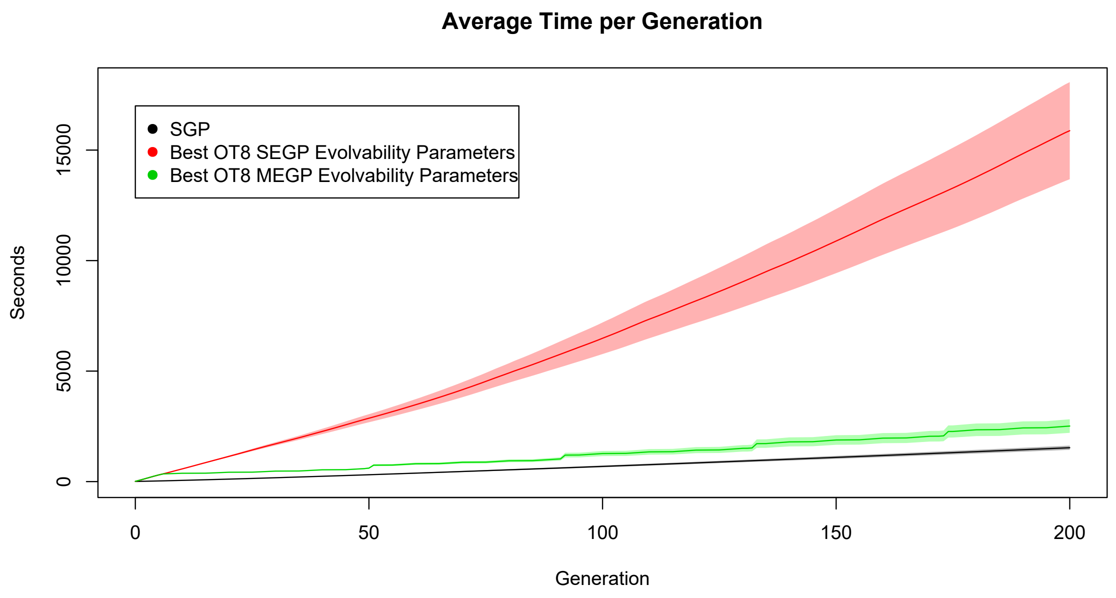

Figure 6.3: Cumulative time over generation comparing SGP, SEGP, and MEGP on the Electricity problem using the best evolvability selection parameter performers of the OT problem. SEGP requires considerably more time to complete than either SGP or MEGP. MEGP's additional time mostly comes from the first few generations, where sampling occurs to generate labelled training instances. Additional time is required on retraining periods. MEGP has periods of increased time usage on retraining periods.

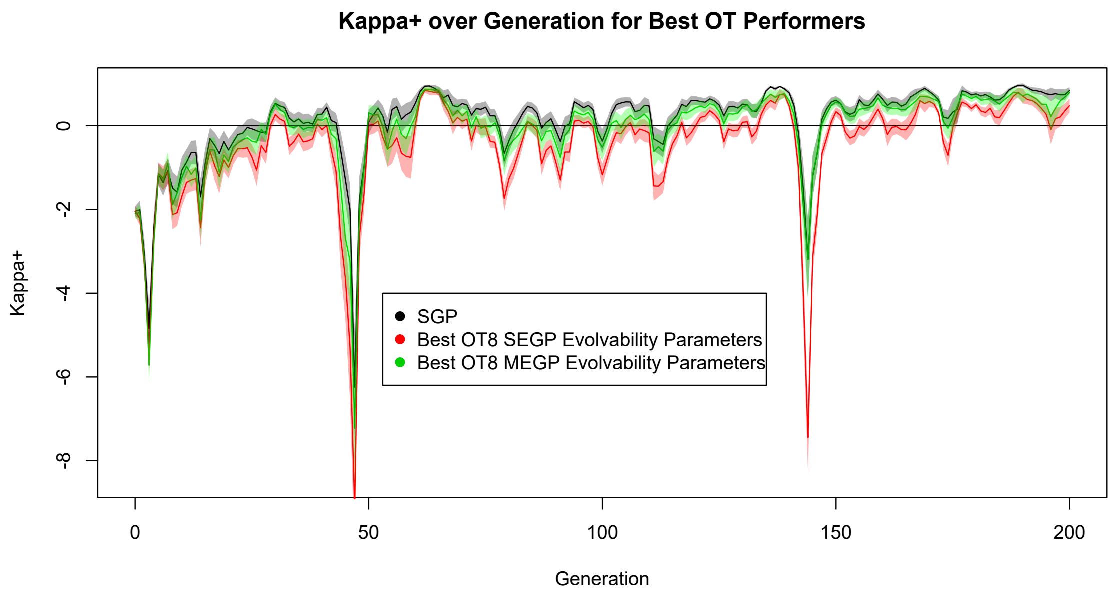

Figure 6.4: Kappa+ over generation comparing SGP, SEGP, and MEGP on the Electricity problem using the best evolvability selection parameter performers of the OT problem. A Kappa+ of 0 denotes the performance of the naïve No-Change classifier. SGP does not outperform the NCC. SEGP and MEGP do not outperform the NCC with these evolvability selection parameters.

Kappa+. We see in the fitness comparison, shown in Figure 6.1, that the top performers in the OT domain are not adequate for a dynamic domain. They cannot maintain fitness as well as the baseline and do not gradually overcome it by having a more evolvable population. Efforts as in the OT domain are insufficient to maintain the benefits of evolvability selection when facing a moving fitness target and concept drift. The best performers in static domains tended to focus on heavy diversity and evolvability in the first few generations. Ensuring an evolvable, diverse early population had long-term consequences, eventually allowing more those populations to outperform more fitness-focused populations. Though some interleaved evolvability selection was also helpful, dynamic domains should require more constant pressure to deal with the constantly shifting fitness targets. The model outperforms the sampling, as for the current settings sampling itself is detrimental, and modelling is simply less so.

Comparing prequential error in Figure 6.2, we see that prequential error fairly closely follows fitness trends. Evolvability selection on its own was not enough to improve prequential accuracy. We may have hoped that even without ideal evolvability selection, that prequential accuracy would improve, as evolvability selection should induce a more generalizable population, which may perform better on unseen instances. That does not seem to be the case. This indicates that a further pivot is necessary to ensure that a more evolvable population is maintained. Prequential error is the error on the successive generation's sliding window.

We see in Figure 6.3 that under the new domain, there are considerable time gains to be made by using modelling instead of sampling. OT problems still required many more fitness evaluations the more sampling occurred, but fitness evaluation was not especially calculation intensive compared to all the other operations that occur, and especially did not suffer too significantly compared to modelling due to the additional overhead required to build and maintain evolvability models. The results highlight the value in MEGP over SEGP.

We see in Figure 6.4 how well the baseline and best previous performers compare to the No-Change classifier, using prequential accuracy. Recall that a method is better than the No-Change classifier when *Kappa+* is greater than 0. We see that even SGP can perform better than that naïve classifier under most generations. It does not perform very well early on, and still has trouble dealing with some generations when concept drift occurs. However, its performance suggests GP is capable of handling the electricity domain.

6.2.2 Sampling

We have determined that changes in the use of evolvability selection are necessary for the streaming domain. Considering the challenges posed by concept drift, we must use evolvability in different ways in order to produce useful results. Many different methods of selecting for evolvability, measuring evolvability, and altering training methods were employed to search for better results. We shall briefly describe the failures, then describe the successful methods in more detail. Many alterations of evolvability selection, including weights, decay, and evolvability selection cut-off generations were employed, with no success. Rates of mutation were increased, hoping the population would adapt more quickly, and higher mutation evolvability would be more significant. Alternative methods of selecting the fittest individual, for the purposes of selecting one for prequential accuracy, were proposed. These considered that the most generalizable individuals were not accounted for with fitness, and different evolvability rates could predict an individual that would perform better on unseen future instances. We also attempted increasing the number of evolvability samples, to ensure no uncertainty could be responsible for the worse performance. A method of using evolvability for selection a random amount of the time was introduced, hoping for a more even mixture of fitness-only and evolvability selection, without needing interleaving generations, which would be problematic with streaming problems, also met no success. Particular problems seem to be centred about the sliding window; these early unsuccessful results were attempted with variously sized sliding windows, but none of them ever used overlapping instances. Each sliding window had entirely unseen instances, and older instances had no value. This proved to be too unstable for GP, even when evolvability is used. Success was found only once more generous overlapping of sliding window instances was allowed.

Success in selecting for evolvability in the electricity domain is possible. For the following experiments, we use the GP evolutionary parameters as specified in Table 6.3. Further, we conduct 100 runs of 200 generations, where each run uses its own random seed. We use an overlapping window size of 100, sliding 20 instances each generation. The baseline consists of SGP. To achieve more balanced results in streaming domains, interleaved evolvability selection is dropped in favour of more consistent pressure. Greater differentiation between individuals' evolvability score is achieved by switching from positive-probability evolvability (PPE) to magnitude evolvability (ME). Mutation evolvability is switched to outgoing-crossover evolvability. An individual that causes high magnitude fitness improvements in others in the population is favoured. Cross-over operations occur more frequently than mutation operations and are dependent upon the rest of the population to calculate. Crossing into high-fitness individuals would more likely result in a worse outgoing evolvability score, meaning it may be difficult to detect highly evolvable individuals when the population has converged to a high fitness.

Firstly, we compare PPE with ME, using 100 samples and an evolvability weight

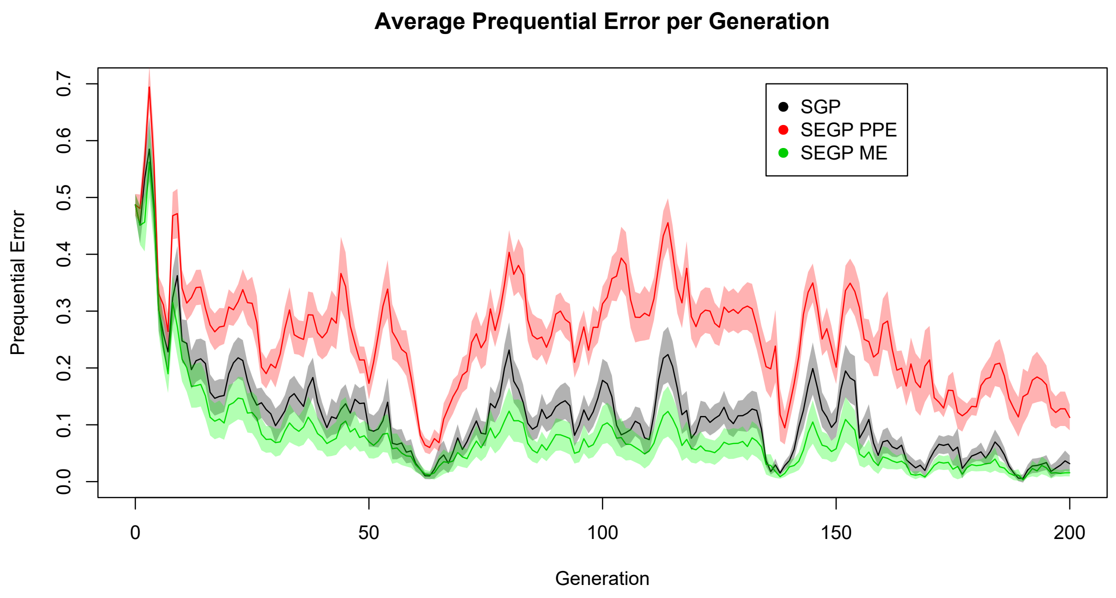

Figure 6.5: Prequential error over generation comparing SGP and SEGP on the Electricity problem using ME or PPE. PPE causes an increase in prequential error. ME causes a decrease in prequential error and fairly consistently outperforms SGP. ME does particularly well during generations where a large increase in error occurs.

of 10, with no other additional selection methods. As we see by their prequential error in Figure 6.5, the PPE method performs more poorly than SGP. However, using ME to influence selection produces results that are better than SGP in most generations. Note that SEGP using PPE performs better than SGP by about a set amount, following the general shape of the prequential error curves. However, it adds significantly more stability for the poorly performing generations. Where SGP has error rates that form mountains, ME selection forms hills. ME actually performs worse on generations where SGP performs exceedingly well. Ultimately, this is indicative of the greater stability that using evolvability offers. The opposite occurs when PPE is used. It is not producing a more evolvable population, despite selecting for a form of evolvability. The population is less tolerant of concept drift than SGP. Simply identifying when evolvability increases does not carry enough information, especially when individuals are most likely to change for the worse. Measuring the magnitude can capture that information; though it may be more difficult to measure accurately, it is a marked improvement in these circumstances.

Next, using strictly ME, we compare the effects of increasing the evolvability weight. In Figure 6.6, we see that at low magnitudes, the weight does not make much difference. Even at low magnitudes, prequential error is stabilized. In Figure 6.7, increasing the evolvability weight further, we see that the best performer is an evolvability weight of 100. It is not much better than the others, but there is a definite pattern of increasing results up to an evolvability weight 100 and then declining thereafter. That the results are so similar may suggest that evolvability magnitudes differences between individuals dwarf fitness differences.

The large number of samples required in these experiments has added considerable computational time, relative to the OT domain. We test if we can maintain these results with fewer samples, with a constant evolvability weight of 1000, to magnify any

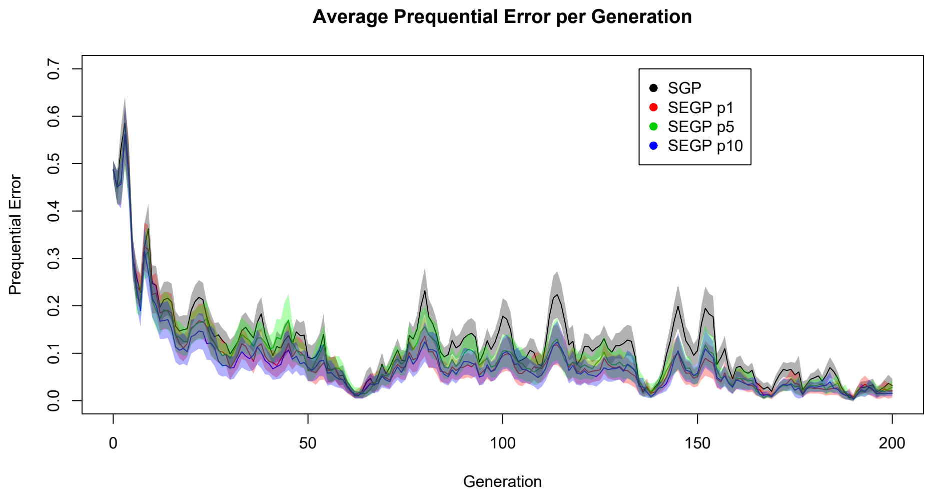

Figure 6.6: Prequential error over generation comparing SGP and SEGP on the Electricity problem, using various evolvability weights. The number following "p" indicates the value of p. For streaming problems, we do not use an evolvability ceasing generation parameter, so "g" is omitted. We use ME and outgoing crossover evolvability. Each SEGP is able to outperform SGP. At low values, the evolvability weight does not cause much difference.

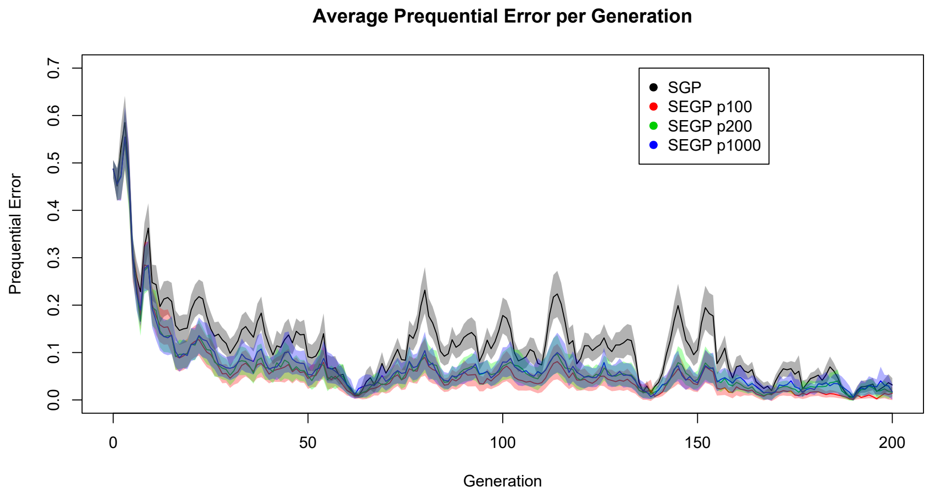

Figure 6.7: Prequential error over generation comparing SGP and SEGP on the Electricity problem, using various evolvability weights. A p value of 100 provides the best performance. The larger values are all able to outperform SGP, and the lower p values shown in Figure 6.6.

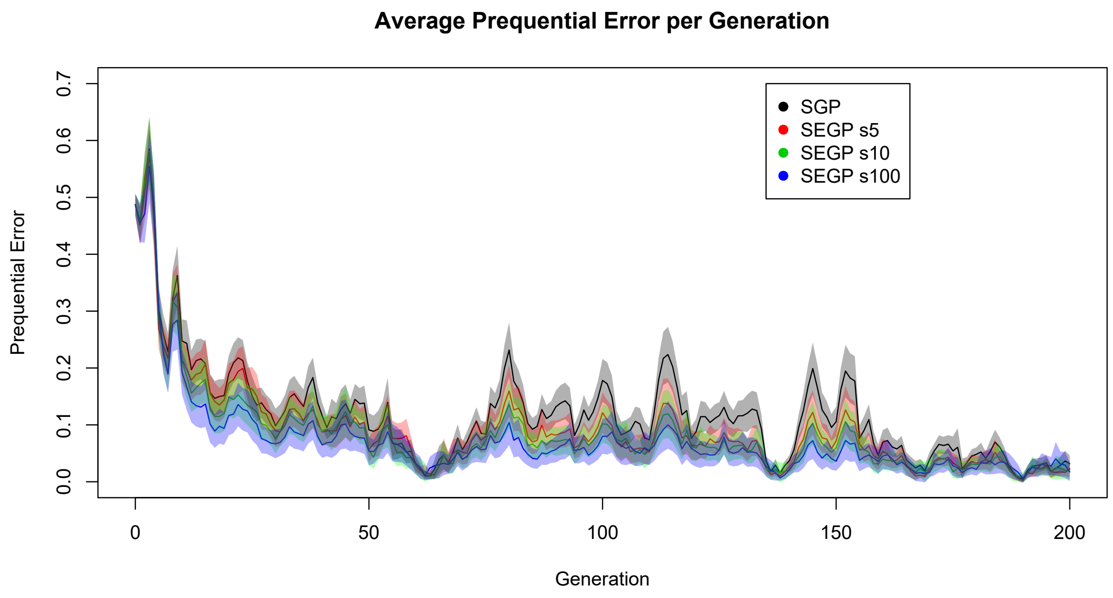

Figure 6.8: Prequential error over generation comparing SGP and SEGP on the Electricity problem, using various numbers of evolvability samples. There is not much difference with more samples beyond 10. Five samples is not different enough from SGP; in particular, it does not resist intervals of higher error as well as more samples.

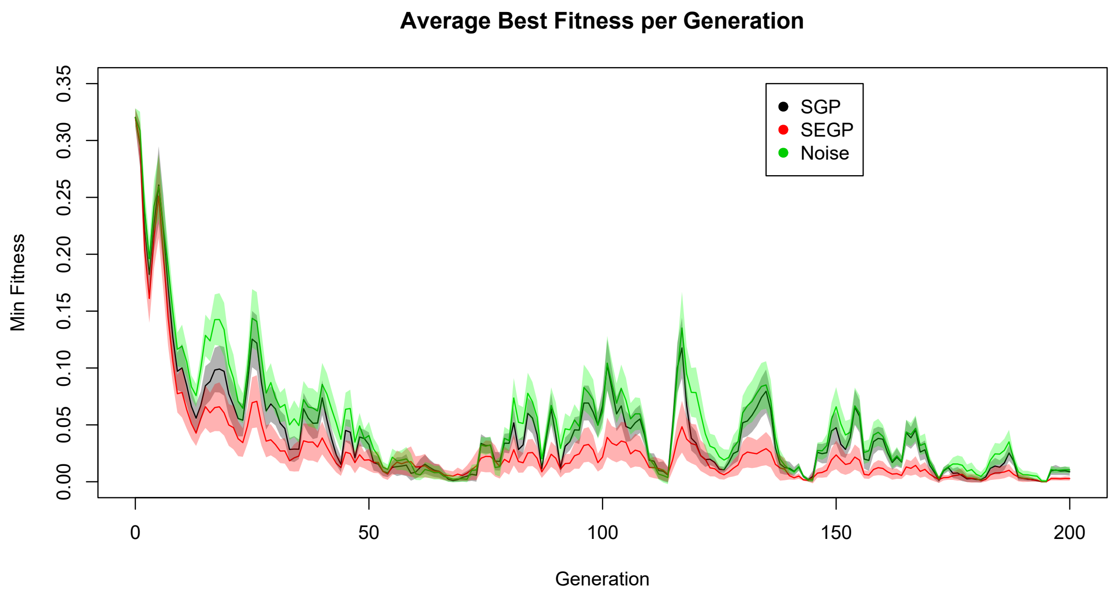

Figure 6.9: Fitness over generation comparing SGP and SEGP on the Electricity problem, using the best performing SEGP parameters and its noise counterpart. Noise is implemented by using the same parameters as SEGP, but instead of sampling for evolvability, we select a uniformly random value for evolvability. Noise performs slightly worse than SGP, and SEGP outperforms both for fitness.

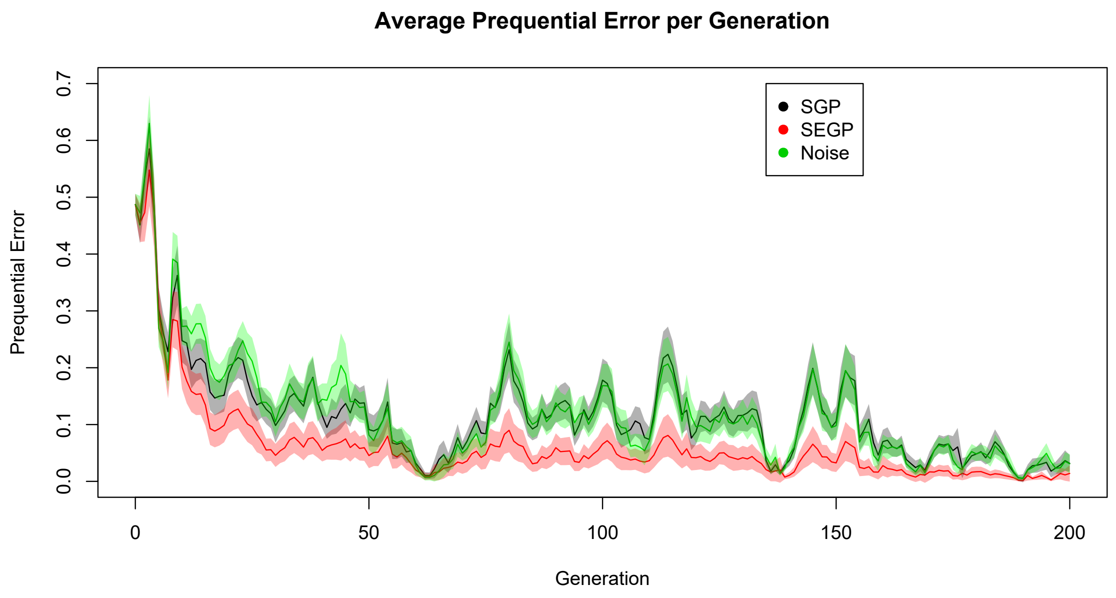

Figure 6.10: Prequential error over generation comparing SGP and SEGP on the Electricity problem, using the best performing SEGP parameters and its noise counterpart. SEGP outperforms both for prequential accuracy.

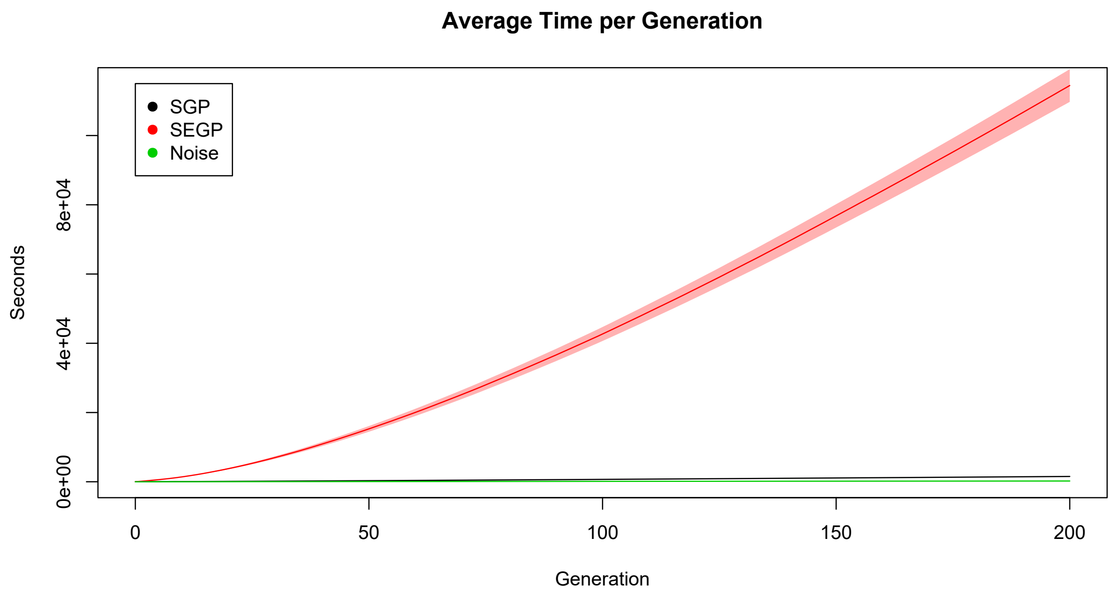

Figure 6.11: Cumulative time over generation comparing SGP and SEGP on the Electricity problem, using the best performing SEGP parameters and its noise counterpart. SGP and noise are indistinguishable. The best prequential performer of SEGP requires about 65.6 times the amount of computational time that SGP does, though the effect is not quite linear.

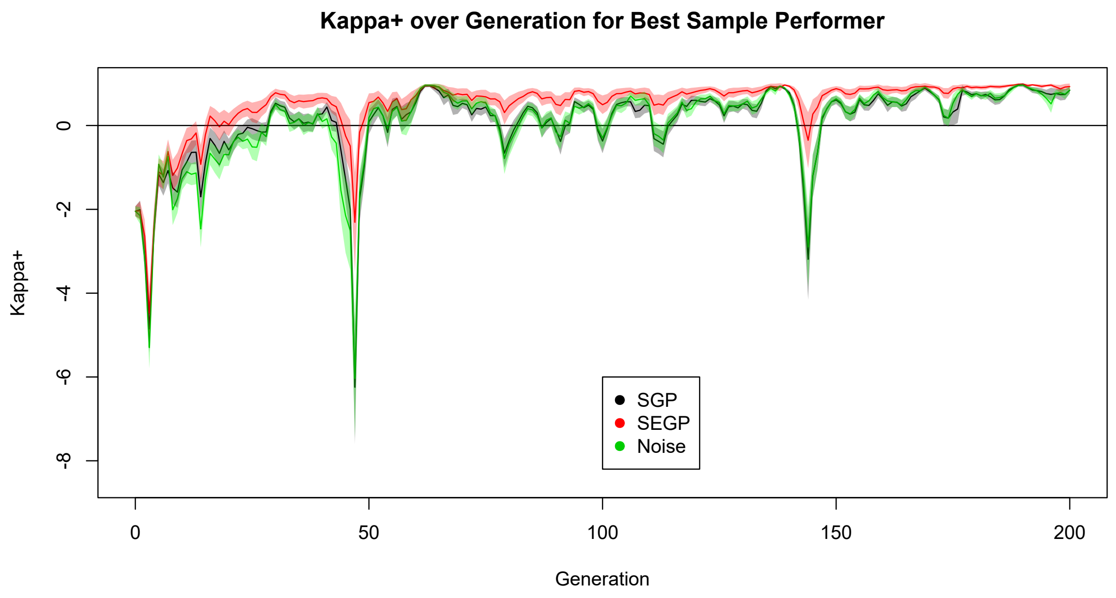

Figure 6.12: Kappa+ over generation comparing SGP and SEGP on the Electricity problem, using the best performing SEGP parameters and its noise counterpart. SEGP is much more consistently able to overcome the No-Change classifier than SGP. Past the initial evolution and training generations, SEGP only fails in two intervals, opposed to SGP, which fails in seven.

differences in SEGP. In Figure 6.8, we see that more samples does indeed produce better results. However, the extra computational time required becomes onerous, compared to the results.

The best SEGP result is obtained using 100 samples, outgoing-crossover-magnitude evolvability, weighted constantly against fitness by 100 times its natural amount. For further comparison, we introduce a noise method. The noise method functions use evolvability selection as the proposed system does; however, whenever evolvability is used in the selection process, a uniformly random value for it is generated in the $[0, 1]$ range. This comparison shows that merely introducing noise to selection does not have the beneficial effect that sampling for evolvability does. We see in Figure 6.9 that its fitness is better than SGP on average. It achieves similar poor results on the first few generations and then weathers changes in concept better than SGP. This demonstrates that the noise method performs worse in fitness than SGP.

In Figure 6.10 we see that prequential error once more performs quite similarly to fitness. The noise method is little steadier with SGP. Use of evolvability is lending an general improvement in prequential accuracy, and particularly, is providing a stable foundation for when concept drift occurs.

In Figure 6.11, we see the detriments of sampling evolvability. It requires considerably more time in order to generate the superior model by selecting for evolvability. SGP is about 65.6 times faster than sampling for evolvability 100 times. We have seen in the number of samples experiments that a trade-off exists, where even limited sampling can produce better results than SGP. Based on this ratio of sampling to performance, we would expect SGP to be about 3.28 faster than 5 sampling evolvability selection. We seek to improve this aspect and reduce the trade-off, in the proceeding subsection, when MEGP is introduced.

In Figure 6.12, we see that evolvability selection is able to improve upon SGP, and achieve much more stable gains over the No-Change classifier, only performing worse for 2 concept changes after the initial evolutionary period. This is opposed to SGP, which underperforms the No-Change classifier an additional 5 periods. This clearly indicates the viability of evolvability selection in streaming domains. The increased stability is evident in the gentler slopes of the curve. The comparison with a noise method shows that this is due to evolvability selection. Exceeding the No-Change classifier shows that it performs comparably well with the state-of-the-art machine learning methods in the NSWED domain.

6.2.3 Modelling

We determined sufficiently helpful evolvability selection settings in the preceding section, considering SEGP. We now conduct experiments to improve upon those results, in the manner of computational time relative to SGP. We move to replace SEGP with MEGP, as much as possible, to realize time gains while keeping as much as necessary to keep accuracy gains. First, we examine how many samples are necessary for these conditions to maintain an effective model.

We see in Figure 6.13 that we can generate models that outperform SGP fairly easily, even trying to calculate outgoing-crossover-magnitude evolvability. ME poses additional challenges relative to PPE, given that outliers have a more significant effect on magnitude. Crossover also poses additional challenges, since it necessarily depends on the condition of the rest of the population, which are not considered in the evolvability ANN model. These factors increase the difficulty of modelling evolvability and thus increase the error. We further see that additional samples do not increase performance greatly relative to the additional computations that such sampling requires. However, the additional sampling required to generate a good

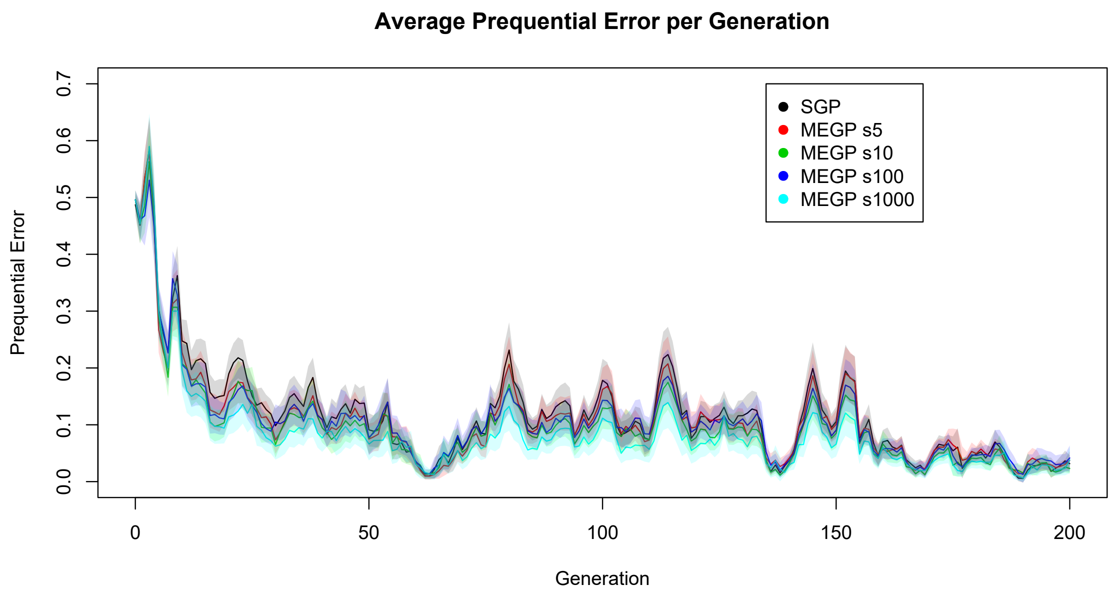

Figure 6.13: Prequential error over generation comparing SGP and MEGP on the Electricity problem, using various numbers of evolvability samples. More samples yields better performance. The effect is not linear, as many more samples are required to achieve incremental gains.

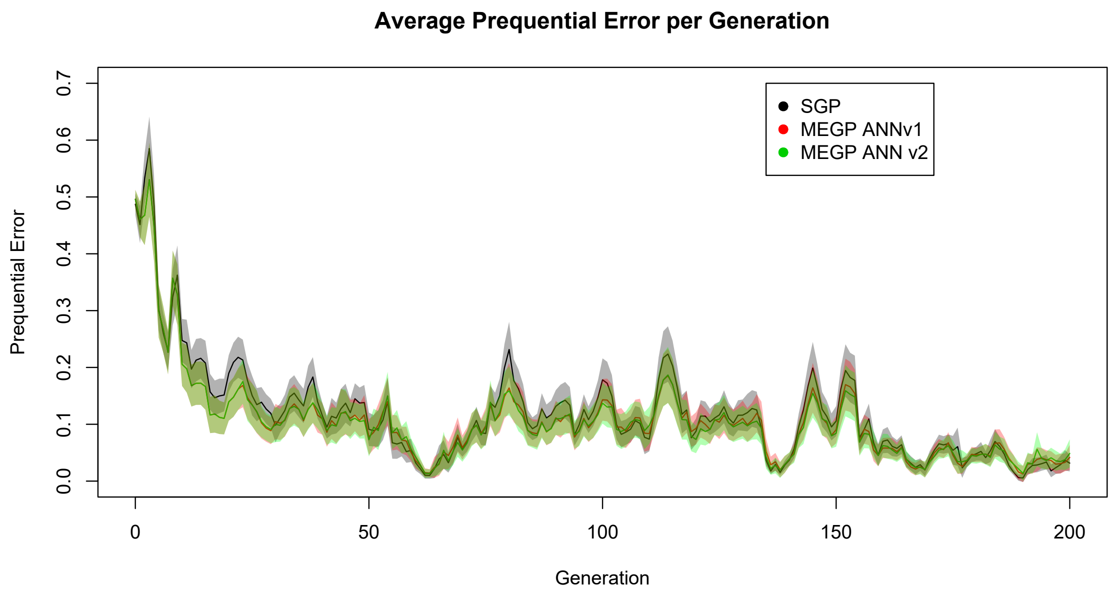

Figure 6.14: Prequential error over generation comparing SGP and MEGP on the Electricity problem, using various ANN parameters, with modelling. Allowing more training epochs does not make the ANN perform better enough such that it decreases prequential error.

model is an early fixed time cost. In a streaming problem, our evolvability model will persist, and as long as it performs adequately, it do not require much additional time to use. Devoting additional resources to train a model with some set instances is reasonable, as the recurring costs while new streaming instances are generated is very low compared to sampling all new instances. It may still be valuable to devote extra time to have more accurate sampling if this leads to a better model. While each performs quite similarly, there is a trend that more samples lead to more concept drift resistance.

We attempt fine-tuning some ANN parameters, with little effective difference. Our original ANN used a maximum of 1000 epochs and an early stopping error of 0.001. Changing these values by orders of magnitude did not alter results significantly.

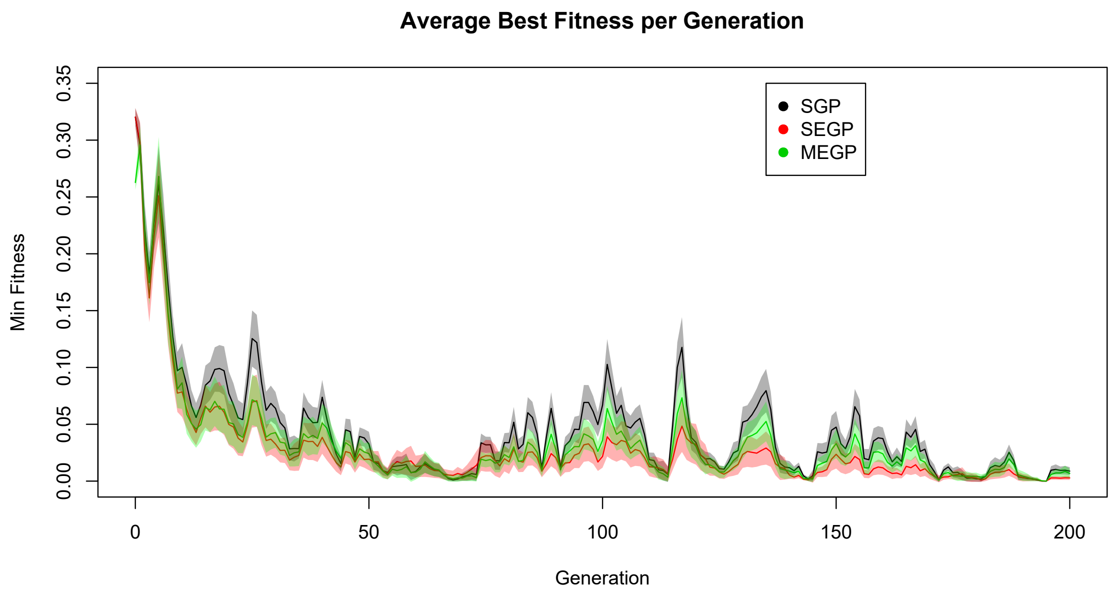

Figure 6.15: Fitness over generation comparing SGP, SEGP, and MEGP on the Electricity problem using the best performing evolvability parameters. SEGP performs the best, MEGP slighty worse, but both outperform SGP.

The ANN predictive results for evolvability did not improve noticeably, nor did the difference in computational time required to compute more or fewer epochs impact the total time required by the entire system onerously. In Figure 6.14, we see that ANN *v2* does not perform better than *v1*, despite allowing 10 times the training epochs.

The best performing MEGP is similar to SEGP, only it uses 1000 samples to train. It only trains in the first 20 generations and uses the model to predict evolvability for selection purposes from then on. We see in Figure 6.15 that it performs about as well as sampling when concept drift is weaker. However, it is much less able to resist concept drift, working about half as well at resisting concept drift as ongoing sampling does. This is also evident in Figure 6.16. However, the benefits are an ongoing reduction in computational complexity. In Figure 6.17, we see that

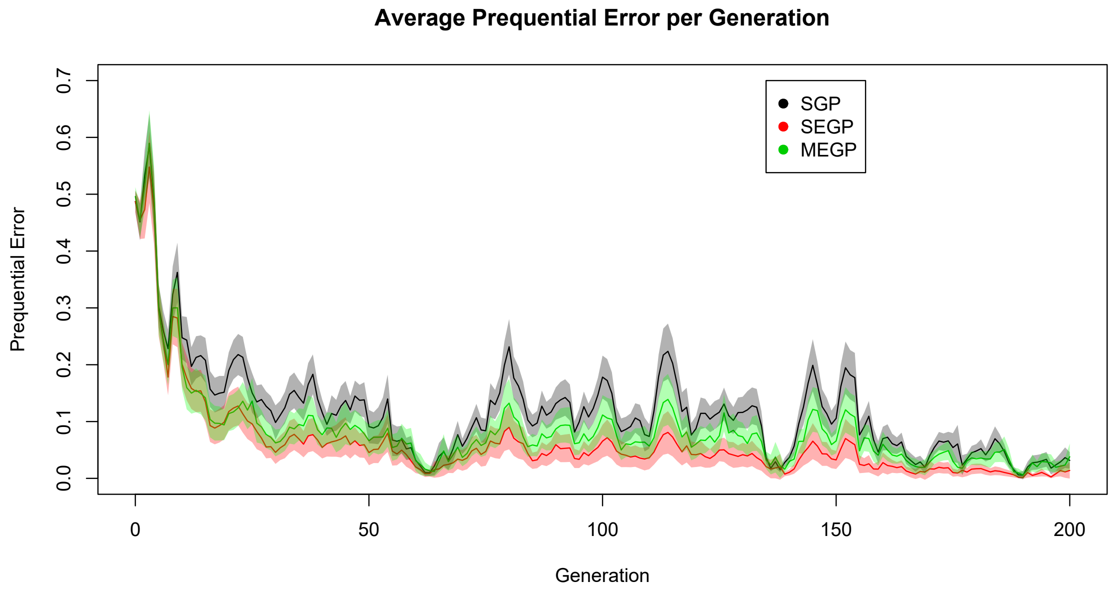

Figure 6.16: Prequential error over generation comparing SGP, SEGP, and MEGP on the Electricity problem using the best performing evolvability parameters. SEGP performs the best, MEGP slighty worse, but both outperform SGP.

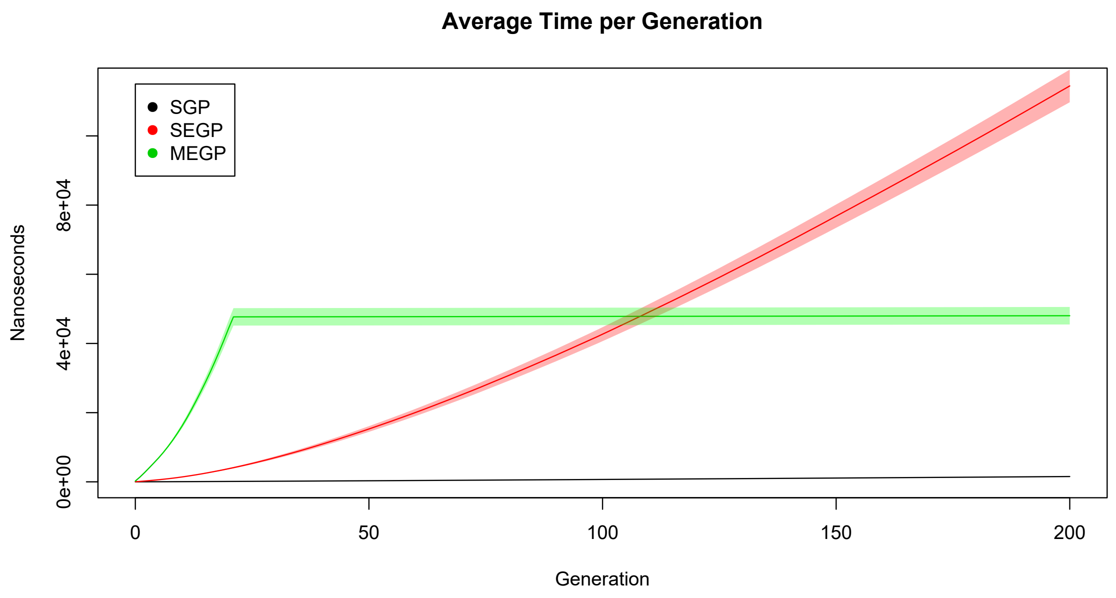

Figure 6.17: Cumulative time over generation comparing SGP, SEGP, and MEGP on the Electricity problem using the best performing evolvability parameters. MEGP requires the most initial time, to generate good training instances. Once the evolvability model is built, it suddenly plateaus, only requiring as much time as SGP. SEGP requires additional time throughout. If the initial training generations are treated as a fixed time cost, as we would in a long-running streaming problem, MEGP is nearly equivalent to SGP.

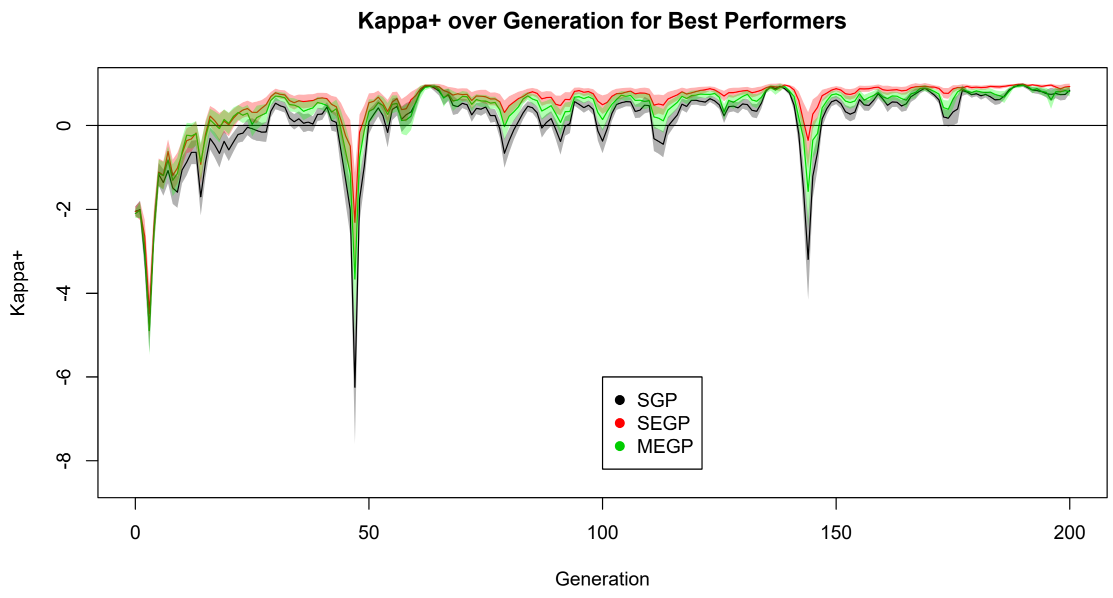

Figure 6.18: Kappa+ over generation comparing SGP, SEGP, and MEGP on the Electricity problem using the best performing evolvability parameters. MEGP is much more consistently able to overcome the No-Change classifier than SGP. Past the initial evolution and training generations, MEGP only fails in two intervals, opposed to SGP, which fails in seven. This is similar to the SEGP performance.

the additional initial samples that build a good model requires a steep time cost, far outpacing sampling alone. However, once training ends, modelling method time quickly plateaus, becoming similar to SGP. The benefits over sampling become more significant the more generations we allow. In streaming problems, the generations should continue indefinitely. The initial training period becomes more of a fixed cost. Furthermore, resources may be considered less strained in the initial training period. We require more judicious use of ongoing resources. Finally, in Figure 6.18 we see that MEGP is still able to surpass the No-Change classifier in all but the most serious incidents of concept drift. MEGP's benefits are made clear. MEGP sacrifices accuracy, compared to SEGP, but at considerable gains in computational complexity, that is, there is additional initial expense in computational complexity, in return for a massive reduction in post-training computational complexity.

Chapter 7

Conclusion

In conclusion, we designed three genetic programming (GP) systems that each exploit evolvability to produce more accurate results than standard genetic programming (SGP) alone. These systems culminated in Model-Evolvability Genetic Programming (MEGP), which is able to produce more accurate results than SGP without onerous additional computation time. We empirically investigated the conditions required for evolvability selection to be useful, and we demonstrated the superiority of evolvability selection in artificial, real, static, and dynamic domains.

7.1 *ap*EGP

a priori Evolvability Genetic Programming is demonstrated primarily in Chapter 4. *ap*EGP is shown to have benefits in parity and regression domains. *ap*EGP is designed primarily to show that evolvability in GP can be modelled effectively with easily generated evolutionary statistics. It is not practical to generate evolvability models for a problem *a priori*, as the potential statistical space is large and requires too many evolvability samples to chart. It presents idealized modelling conditions, which demonstrate that modelling evolvability is possible. The experiments further show that introducing modelled evolvability within the selection process can lead to improved fitness results. Ultimately, this demonstrates that modelling evolvability has merits.

7.2 SEGP

Sample-Evolvability Genetic Programming (SEGP) is demonstrated in Chapter 5 and Chapter 6. SEGP is compared with MEGP to demonstrate how much efficacy is lost when using modelled evolvability. Since some accuracy is lost in the modelling process, we first demonstrate that sampled evolvability can be used to produce fitter results. SEGP provides more idealized conditions than MEGP, to observe selection conditions without considering model accuracy losses. We use SEGP to consider multiple forms of evolvability and multiple forms of evolvability selection, in search of fitness gains. SEGP is able to produce fitter results when using the appropriate amount of evolvability selection. The appropriate amount varies from domain to domain. Strong but interleaved mutation evolvability selection is most successful in the static domains of Chapter 4 and Chapter 5, and moderate, constant, outgoing-crossover evolvability selection is most successful in the dynamic domain of Chapter 6. However, SEGP requires constant additional computation time to sample for evolvability as long as evolvability is used in selection. This may be mitigated by only using evolvability in selection a portion of the time, but the additional time required to evaluate fitness so many additional times is onerous, especially in dynamic domains.

7.3 MEGP

The efforts of this thesis culminate in MEGP. With modelling evolvability shown to be possible, and selection of evolvability shown to be useful, MEGP is deployed to make evolvability selection practical. MEGP does not require the constant onerous evolvability sampling that SEGP requires. Instead, the extra computation is mostly front-loaded, to acquire accurate labels for training instances in the first few generations. Once the evolvability model is made, it requires little extra computation, relative to SGP, to predict evolvability. Furthermore, it does not suffer onerously in fitness performance relative to SEGP. MEGP is able to achieve fitness performance gains relative to SGP for a modest increase in computational time, which diminishes with increasing generations. Thus, MEGP is particularly well suited to streaming problems, such as those of Chapter 6.

7.4 Limitations and Future Work

*ap*EGP, SEGP, and MEGP all rely on sampling evolvability to obtain an estimate for true evolvability. They will not function if a fairly accurate estimate cannot be obtained. The number of samples required to achieve a fair evolvability estimate for the examined problems is low. Problems with a higher fitness variance require more samples to achieve fair magnitude evolvability (ME) approximation. Positive-probability evolvability (PPE) can be similarly vulnerable to fitness variance. More samples translate to more computational complexity. Modelling evolvability, in particular, requires good evolvability estimates.

We have demonstrated that evolvability can be modelled with easily generated evolutionary statistics on the examined problems. However, it is possible that these statistics are insufficient to model evolvability in all problems. There may be domains where generation, tree size, function frequency, non-input terminal frequency, input frequency, dormancy ratio, previous fitness, and fitness do not predict future fitness reliably. We have restricted ourselves to easily generated statistics, but it may be that more computationally expensive statistics could be useful enough in evolvability modelling that it would be worth spending the extra time to generate them. The generational age of the best performing program may be useful as it indicates how long the population has been unable to improve upon fitness.

We have restricted ourselves to tree-based GP. It is possible that evolvability is not easily modelled in other paradigms. We have further restricted ourselves to mutation and crossover evolvabilities. Crossover evolvability is population dependent, but evolutionary statistics regarding the population are not used to model evolvability, only individual statistics are. Modelling crossover evolvability is incomplete without considering population dynamics, though we have observed that it may be modelled sufficiently well as to produce fitness improvements.

We have empirically shown that MEGP can produce fitness performance gains, but we have not described the problem conditions necessary for it to do so. We conjecture that since the fitness landscape of OrderTree (OT) is structured such that there are many local minima which are quite disconnected from the global minimum, evolvability is useful there. We conjecture that since concept shift demands a more evolvable population to ensure good ongoing fitness that evolvability is useful there. We have not defined how rugged a fitness landscape must be, nor defined how much a fitness landscape much shift, for evolvability to be useful. The impact of diversity and evolvability along fitness landscapes could be used to characterize problems where evolvability is useful. There are many potential dynamic domains where evolvability could be useful, due to the ubiquity of streaming data.

SEGP and MEGP are able to achieve efficacy gains relative to SGP but at a cost of computational time. MEGP reduces the computational cost considerably, relative to SEGP. If additional computational time is not problematic, and efficacy is favoured, SEGP is favoured. However, computational time may be limited. Particularly, if a rapid response to new data is required, MEGP offers benefits for minimal loss in efficacy. Problems, where computational limits or accuracy demands are known, could benefit from a more flexible system, one that could adapt to sample evolvability more or less, as required. Such a system would be useful in a long-running dynamic domain. The balance between efficacy and efficiency are problem dependent, such that it may be difficult to decide a general balance between the two.

Simulated annealing schedule optimization is well researched and is analogous to varying the strength of evolvability selection. Further investigation into simulated annealing schedule optimization may yield insights into the ideal timing of evolvability selection. This work has mainly focused on stronger evolvability pressure in earlier generations, or consistent pressure for dynamic domains. The optimal evolvability pressure may be more complex. Pareto selection indicates there may be more complexity to optimal evolvability selection. Optimal selection may involve non-linear combinations of fitness and evolvability.

A more evolvable population may have benefits when searching for solutions for similar tasks. Evolvable populations may be useful as initial populations, replacing random initial populations. We have focused on using evolvability to improve fitness within single tasks, but there may be fitness gains in using evolvable populations for similar tasks. Evolvability selection is beneficial for dynamic domains which indicates that evolvability selection helps a population adjust to new problem distributions.

Models were developed both *a priori* and parallel to evolution. The success of *ap*EGP indicates that evolvability models are portable within a single task. It is possible that evolvability models may be portable between similar or even dissimilar tasks. Reusing evolvability models would produce computational complexity gains.

7.5 Concluding Remarks

In conclusion, we have demonstrated the necessary amount of evolvability sampling to generate a sufficiently accurate calculation of evolvability. We have further demonstrated how evolvability may be used to modify the standard fitness function, in order to encourage the selection of evolvable individuals, and how this may generate an overall increase in standard fitness. We have demonstrated how many instances and samples are required to build a sufficiently accurate model of evolvability, in order to predict it to guide selection. Finally, we have shown that modelling evolvability and using it in selection allows for a similar improvement in overall fitness than simply sampling for evolvability. The additional number of evaluations required to sample evolvability to guide selection is prohibitive. The extra computational time required to predict evolvability using an external program is prohibitively computationally expensive, as well. Furthermore, its use is limited in these experiments by gathering training instances *a priori*. However, the results demonstrate that predicting evolvability from relatively few training instances with a relatively small number of samples still leads to improved fitness. This indicates the viability of modelling evolvability in order to improve GP.

Bibliography

[1] L. Altenberg. Advances in Genetic Programming. Chapter 3, The evolution of evolvability in genetic programming. pages 47–74. MIT Press, 1994.

[2] L. Altenberg. Evolvability and robustness in artificial evolving systems: Three perturbations. *Genetic Programming and Evolvable Machines*, 15(3):275–280, 2014.

[3] W. Banzhaf. Genetic Programming and Emergence. *Genetic Programming and Evolvable Machines*, 15(1):63–73, 2013.

[4] W. Banzhaf, G. Beslon, S. Christensen, J. A. Foster, F. Képès, V. Lefort, J. F. Miller, M. Radman, and J. J. Ramsden. Guidelines: From artificial evolution to computational evolution: A research agenda. *Nature Reviews Genetics*, 7(9):729, 2006.

[5] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann San Francisco, 1998.

[6] J. K. Bassett, M. Coletti, and K. A. De Jong. The relationship between evolvability and bloat. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 1899–1900. ACM, 2009.

[7] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 443–448. SIAM, 2007.

[8] A. Bifet, J. Read, I. Žliobaitė, B. Pfahringer, and G. Holmes. Pitfalls in benchmarking data stream classification and how to avoid them. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 465–479. Springer, 2013.

[9] E. K. Burke, S. Gustafson, and G. Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62, 2004.

[10] E. Byvatov, U. Fechner, J. Sadowski, and G. Schneider. Comparison of support vector machine and artificial neural network systems for drug/nondrug classification. *Journal of Chemical Information and Computer Sciences*, 43(6):1882–1889, 2003.

[11] M. Castelli, M. De Felice, L. Manzoni, and L. Vanneschi. Electricity demand modelling with genetic programming. In *Proceedings of the Portuguese Conference on Artificial Intelligence*, pages 213–225. Springer, 2015.

[12] C. P. Chen and C.-Y. Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314–347, 2014.

[13] P. Chongstitvatana. Improving robustness of robot programs generated by genetic programming for dynamic environments. In *Proceedings of the 1998 IEEE Asia-Pacific Conference on Circuits and Systems*, pages 523–526. IEEE, 1998.

[14] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[15] A. P. Dawid. Present position and potential developments: Some personal views: Statistical theory: The prequential approach. *Journal of the Royal Statistical Society. Series A (General)*, 147(2):278–292, 1984.

[16] E. D. De Jong and J. B. Pollack. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines*, 4(3):211–233, 2003.

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

[18] G. Ditzler and R. Polikar. Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2283–2301, 2013.

[19] T. Flatt. The evolutionary genetics of canalization. *The Quarterly Review of Biology*, 80(3):287–316, 2005.

[20] B. Fowler and W. Banzhaf. Modelling evolvability in genetic programming. In *European Conference on Genetic Programming*, pages 215–229. Springer, 2016.

[21] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[22] M. M. Gaber. Advances in data stream mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):79–85, 2012.

[23] C. Gagné and M. Parizeau. Genericity in evolutionary computation software tools: Principles and case study. *International Journal on Artificial Intelligence Tools*, 15(2):173–194, 2006.

[24] E. Galván-López, J. McDermott, M. ONeill, and A. Brabazon. Defining locality as a problem difficulty measure in genetic programming. *Genetic Programming and Evolvable Machines*, 12(4):365–401, 2011.

[25] E. Galván-López, R. Poli, A. Kattan, M. ONeill, and A. Brabazon. Neutrality in evolutionary algorithms What do we know? *Evolving Systems*, 2(3):145–163, 2011.

[26] J. Gama, R. Sebastião, and P. P. Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.

[27] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

[28] I. Gonçalves and S. Silva. Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In *Proceedings of European Conference on Genetic Programming*, pages 73–84. Springer, 2013.

[29] D. Hadka and P. Reed. Borg: An auto-adaptive many-objective evolutionary computing framework. *Evolutionary Computation*, 21(2):231–259, 2013.

[30] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[31] M. A. Hall. Correlation-based feature selection of discrete and numeric class machine learning. *In Proceedings of 17th International Conference on Machine Learning*, pages 359–366, 2000.

[32] S. Harding, J. F. Miller, and W. Banzhaf. Evolution, development and learning using self-modifying cartesian genetic programming. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 699–706. ACM, 2009.

[33] M. Harries. Splice-2 comparative evaluation: Electricity pricing. Technical report, University of New South Wales, 1999.

[34] J. Heaton. Encog: Library of interchangeable machine learning models for Java and C#. *Journal of Machine Learning Research*, 16:1243–1247, 2015.

[35] M. I. Heywood. Evolutionary model building under streaming data for classification tasks: opportunities and challenges. *Genetic Programming and Evolvable Machines*, 16(3):283–326, 2015.

[36] T.-H. Hoang, N. X. Hoai, N. T. Hien, R. I. McKay, and D. Essam. ORDER-TREE: a new test problem for genetic programming. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 807–814. ACM, 2006.

[37] T. Hu and W. Banzhaf. Evolvability and speed of evolutionary algorithms in light of recent developments in biology. *Journal of Artificial Evolution and Applications*, 2010:1, 2010.

[38] T. Hu, W. Banzhaf, and J. H. Moore. Robustness and evolvability of recombination in linear genetic programming. In *Proceedings of European Conference on Genetic Programming*, pages 97–108. Springer, 2013.

[39] T. Hu, J. L. Payne, W. Banzhaf, and J. H. Moore. Evolutionary dynamics on multiple scales: A quantitative analysis of the interplay between genotype, phenotype, and fitness in linear genetic programming. *Genetic Programming and Evolvable Machines*, 13(3):305–337, 2012.

[40] G. Iuhasz, V. I. Munteanu, and V. Negru. Data mining considerations for knowledge acquisition in real time strategy games. In *Proceedings of the 2013 IEEE 11th International Symposium on Intelligent Systems and Informatics (SISY)*, pages 331–336. IEEE, 2013.

[41] D. Jackson. The identification and exploitation of dormancy in genetic programming. *Genetic Programming and Evolvable Machines*, 11(1):89–121, 2009.

[42] H. Jeff. Programming neural networks with Encog3 in Java. Heaton Research, 2011.

[43] T. Jones. *Evolutionary algorithms, fitness landscapes and search.* PhD thesis, The University of New Mexico, 1995.

[44] A. Kattan and Y.-S. Ong. Bayesian inference to sustain evolvability in genetic programming. In *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems*, volume 1, pages 75–87. Springer, 2015.

[45] J. Kepner, V. Gadepally, P. Michaleas, N. Schear, M. Varia, A. Yerukhimovich, and R. K. Cunningham. Computing on masked data: A high performance method for improving big data veracity. In *High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2014.

[46] S. Khanchi, M. I. Heywood, and A. N. Zincir-Heywood. Properties of a GP active learning framework for streaming data with class imbalance. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 945–952. ACM, 2017.

[47] S. Khanchi, A. Vahdat, M. I. Heywood, and A. N. Zincir-Heywood. On botnet detection with genetic programming under streaming data label budgets and class imbalance. *Swarm and Evolutionary Computation*, 39:123–140, 2017.

[48] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

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

[50] J. R. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284, 2010.

[51] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak. Ensemble learning for data stream analysis: a survey. *Information Fusion*, 37:132–156, 2017.

[52] I. Kushchu. Learning, evolution and generalisation. In *Proceedings of the 2003 Congress on Evolutionary Computation*, volume 4, pages 2441–2448. IEEE, 2003.

[53] D. Laney. 3D data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6:70, 2001.

[54] K. Li, S. Kwong, J. Cao, M. Li, J. Zheng, and R. Shen. Achieving balance between proximity and diversity in multi-objective evolutionary algorithm. *Information Sciences*, 182(1):220–242, 2012.

[55] P. Lichodzijewski and M. I. Heywood. Managing team-based problem solving with symbiotic bid-based genetic programming. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pages 363–370. ACM, 2008.

[56] K. M. Malan and A. P. Engelbrecht. A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163, 2013.

[57] O. Matviykiv and O. Faitas. Data classification of spectrum analysis using neural network. *Lviv Polytechnic National University*, Technical Report, 2012.

[58] J. F. Miller. Cartesian Genetic Programming, Chapter 2, Cartesian genetic programming. pages 17–34. Springer, 2011.

[59] J. F. Miller and S. L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.

[60] P. Nordin, F. Francone, and W. Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. *Advances in Genetic Programming.* pages 111–134. MIT Press, 1996.

[61] M. ONeill, L. Vanneschi, S. Gustafson, and W. Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):339–363, 2010.

[62] F. Otero, T. Castle, and C. Johnson. Epochx: Genetic programming in Java with statistics and event monitoring. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 93–100. ACM, 2012.

[63] F. E. Otero and C. G. Johnson. Automated problem decomposition for the Boolean domain with genetic programming. In *Proceedings of the European Conference on Genetic Programming*, pages 169–180. Springer, 2013.

[64] C. Öztürkeri and C. G. Johnson. Self-repair ability of evolved self-assembling systems in cellular automata. *Genetic Programming and Evolvable Machines*, 15(3):313–341, 2014.

[65] M. Pigliucci. Is evolvability evolvable? *Nature Reviews Genetics*, 9(1):75, 2008.

[66] R. Poli, W. Langdon, N. McPhee, and J. Koza. A field guide to genetic programming. *Genetic Programming and Evolvable Machines*, 10(2):229–230, 2009.

[67] R. Poli, L. Vanneschi, W. B. Langdon, and N. F. McPhee. Theoretical results in genetic programming: The next ten years? *Genetic Programming and Evolvable Machines*, 11(3-4):285–320, 2010.

[68] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):67, 2016.

[69] R. Ramos-Pollán, M. Á. Guevara-López, and E. Oliveira. A software framework for building biomedical machine learning classifiers through grid computing resources. *Journal of Medical Systems*, 36(4):2245–2257, 2012.

[70] S. Ramrez-Gallego, B. Krawczyk, S. Garca, M. Woniak, and F. Herrera. A survey on data preprocessing for data stream mining. *Neurocomputing*, 239(C):39–57, 2017.

[71] M. Riedmiller and H. Braun. RPROP-a fast adaptive learning algorithm. In *Proceedings of the International Symposium on Computer and Information Sciences VII*, pages 32–44. Springer, 1992.

[72] M. Riekert, K. Malan, and A. Engelbrect. Adaptive genetic programming for dynamic classification problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 674–681. IEEE, 2009.

[73] E. Schulte, Z. P. Fry, E. Fast, W. Weimer, and S. Forrest. Software mutational robustness. *Genetic Programming and Evolvable Machines*, 15(3):281–312, 2014.

[74] K. Seo and C. Pang. Tree-structure-aware genetic operators in genetic programming. *Journal of Electrical Engineering and Technology*, 9(2):749–754, 2014.

[75] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[76] S. Silva, S. Dignum, and L. Vanneschi. Operator equalisation for bloat free genetic programming and a survey of bloat control methods. *Genetic Programming and Evolvable Machines*, 13(2):197–238, 2012.

[77] K. Sindhya, K. Miettinen, and K. Deb. A hybrid framework for evolutionary multi-objective optimization. *IEEE Transactions on Evolutionary Computation*, 17(4):495–511, 2013.

[78] M. Smith and V. Ciesielski. Adapting to concept drift with genetic programming for classifying streaming data. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 5026–5033, 2016.

[79] T. Smith, P. Husbands, and M. O'Shea. Fitness landscapes and evolvability. *Evolutionary Computation*, 10(1):1–34, 2002.

[80] T. Taheri. Benchmarking and Comparing Encog, Neuroph and JOONE Neural Networks, 2014.

[81] U. Tangen. On evolvability and robustness in the matrix-GRT model. *Genetic Programming and Evolvable Machines*, 15(3):343–374, 2014.

[82] D. Tarapore and J.-B. Mouret. Evolvability signatures of generative encodings: beyond standard performance benchmarks. *Information Sciences*, 313(1):43–61, 2015.

[83] A. Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2), Technical Report, 2004.

[84] A. Vahdat, A. Atwater, A. R. McIntyre, and M. I. Heywood. On the application of GP to streaming data classification tasks with label budgets. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 1287–1294. ACM, 2014.

[85] A. Vahdat, J. Morgan, A. R. McIntyre, M. I. Heywood, and A. N. Zincir-Heywood. Tapped delay lines for GP streaming data classification with label budgets. In *Proceedings of the European Conference on Genetic Programming*, pages 126–138. Springer, 2015.

[86] A. Vahdat, J. Morgan, A. R. McIntyre, M. I. Heywood, and N. Zincir-Heywood. Evolving GP classifiers for streaming data tasks with concept change and label budgets: a benchmarking study. In *Handbook of Genetic Programming Applications*, pages 451–480. Springer, 2015.

[87] L. Vanneschi, M. Castelli, and S. Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 877–884. ACM, 2010.

[88] L. Vanneschi and S. Silva. Using operator equalisation for prediction of drug toxicity with genetic programming. In *Proceedings of the Portuguese Conference on Artificial Intelligence*, pages 65–76. Springer, 2009.

[89] A. Wagner. Robustness and evolvability: A paradox resolved. *In Proceedings of the Royal Society of London B: Biological Sciences*, 275(1630):91–100, 2008.

[90] Y. Wang and M. Wineberg. Estimation of evolvability genetic algorithm and dynamic environments. *Genetic Programming and Evolvable Machines*, 7(4):355–382, 2006.

[91] A. M. Webb. *On Selection for Evolvability.* PhD thesis, University of Manchester, 2016.

[92] A. M. Webb, J. Handl, and J. Knowles. How much should you select for evolvability? In *Proceedings of the 2015 European Conference on Artificial Life*, pages 487–494. MIT Press, 2015.

[93] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaśkowski, U. M. O'Reilly, and S. Luke. Better GP benchmarks: Community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29, 2013.

[94] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[95] W. Yan and C. D. Clack. Behavioural GP diversity for dynamic environments: an application in hedge fund investment. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 1817–1824. ACM, 2006.

[96] X.-H. Yu, G.-A. Chen, and S.-X. Cheng. Dynamic learning rate optimization of the backpropagation algorithm. *IEEE Transactions on Neural Networks*, 6(3):669–677, 1995.

[97] B.-T. Zhang and H. Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

[98] X. Zhu, P. Zhang, X. Lin, and Y. Shi. Active learning from stream data using optimal weight classifier ensemble. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40(6):1607–1621, 2010.

[99] P. C. Zikopoulos and C. Eaton. *Understanding big data: Analytics for enterprise class hadoop and streaming data.* McGraw-Hill Osborne Media, 2011.

[100] I. Zliobaite. How good is the electricity benchmark for evaluating concept drift adaptation. *arXiv preprint arXiv:1301.3524*, 2013.

Appendix A

Software

A.1 Open BEAGLE

In an earlier system, working with parity, regression, and OrderTree (OT) domains, we used a modified minimalist version of Open BEAGLE, referred to as BEAGLE Puppy [23]. Open BEAGLE is an evolutionary computation framework, developed in C++. BEAGLE Puppy utilizes the core GP algorithms of Open BEAGLE, but is simpler to modify for the purposes of modelling and exploiting evolvability, since it is minimalist. It contains a tree-based GP implementation of simple parity and regression problems. Our methodology requires editing the selection process, additional tracking of various statistics (such as dormancy), and sampling for evolvability. These steps are all required to dynamically model evolvability. These steps are easier to implement by editing core GP algorithms. Furthermore, we are afforded more flexibility by working with a lesser amount of code. Another advantage is working with an efficient object-oriented language. To sample for evolvability and robustness, more fitness cases need to be evaluated, which is already the most computationally intensive part of GP. Object-oriented code allows for more easily reusable code; as we expand into more difficult problem domains, we can reuse our efforts building simpler ones. Eventually, BEAGLE Puppy proved too limiting for some major implementation changes that were necessary, and we moved on to another framework, EpochX.

A.2 EpochX

EpochX is an open sourced genetic programming (GP) framework, written in Java [62]. It naturally supports three different GP representations: strongly-typed trees, context-free grammar, and grammatical evolution. It is highly modular, as it was designed to be used primarily by researchers, who would add their own functionality. This work is no exception; EpochX is modified to achieve a system capable of measuring evolvability and using it in selection. EpochX tracks a wide variety of statistical data about each run, which can be used to both help model evolvability, and more easily understand the consequences of doing so. EpochX has a common core package which describes how the evolutionary algorithm functions, regardless of representation. This is further split into several managerial classes, which serve as the core of different evolutionary functions. This means that some functions, such as selection, may be altered quite independently from other functions. Thus, EpochX serves as a useful framework for this work.

EpochX is designed with an event framework, based on the Java event handling model. Events can take forms such as the end of population initialization, start or end of a training generation, or even be tied to statistics and performance of genetic operators. Using events, we can easily describe what we want the system to do, at different magnitudes of the best fitness or best evolvability in the population.

Problem domains in EpochX are specified as models. The model defines the fitness function as well as representational parameters, such as node definitions for tree-based GP. Additionally, how to calculate evolvability is defined here. These aspects all vary depending on the problem domain, and are programmer supplied. Representation-based parameters, such as maximum tree depth, may also be specified as a property of the model. The traits allow new problem domains to be introduced rapidly, since little is required to specify a problem. Models are fed into a run manager, which contain a initialization and generational manager. The generation manager, in turn, contains elitism, selection, and genetic operation managers. Of chief concern of this work are the generation and selection managers.

Generational listeners are added which specify the conditions under which evolvability must be measured or modelled. On certain generations, evolvability is not required for selection or training at all, so it would waste computational resources to calculate it. Generational listeners allow us to trigger the measurement or modelling of evolvability as desired. These circumstances include, but are not limited to, initial evolvability model building, model inaccuracy, or reaching defined generation numbers. Statistical listeners may indicate if the evolvability model is not performing well, and thus indicate more training time or instances are needed.

The selection manager must be altered to include evolvability, and how evolvability influences selection. Since the impact of evolvability on selection will vary, the selection manager is edited to allow varying selection pressures, even after its initialization. Further, in order to evaluate the effectiveness of the evolvability model, it is necessary to track the selection of individuals based on their predicted evolvability, and the measured evolvability; this is accomplished in the selection manager.

Of further interest to this work is statistical information. Programmers may add their own statistic definitions, and they may be treated as all the innately defined statistics. This allows the specification of multiple evolvability statistics, tied to different genetic operators, different base metrics (such as magnitude of improvement versus probability of improvement), and whether it was predicted by a model or sampled for. As statistics are not calculated unless required, specifying an evolvability statistic does not necessarily mean it will always be calculated, thus performance of standard genetic programming (SGP) on the system will not suffer. Further, dormancy statistics are added, since they are used to build evolvability models. Statistics may be calculated using other statistics, forming dependencies, which may not be circular. Once calculated, statistics may be cached, so they will not need to be recalculated, unless an event occurs which indicates that those statistics are no longer valid, such as a new generation beginning.

EpochX naturally supports several GP paradigms, of which tree-based GP is one. Additional functionality, unique to tree-based GP, is included in the framework. This functionality includes statistics unique to tree-based GP (*e.g.* tree size and depth), implementation for genetic operators of mutation and crossover, as well as tree initialization and evaluation. As this work focuses on tree-based GP, the separation of GP paradigms is useful, as it allows easier modification and expansion of the framework, to consider the tree representation specifically, instead of more general purpose modifications that must function with all forms of GP representation.

Problem domains themselves have their own class definitions, extended from the representation models. Programmers need to specify how fitness is evaluated, and describe domain-specific information, such as the function and terminal set for tree-based GP. These are representation-specific as well. If functions or terminals are needed that are not already described in the standard EpochX library, then they must be implemented in their own class, as well. Nodes need to have a return type, an identifier, and a method of evaluation. Functions also need specifications for child nodes. Describing fitness evaluation and the nodes are all that is required for new tree-based GP problems to be specified in EpochX.

A.3 WEKA

Modelling evolvability and robustness requires faster machine learning algorithms than evolutionary computation provides. These may be provided by a machine learning suite, the Waikato Environment for Knowledge Analysis (WEKA) [30]. WEKA provides several interfaces which are useful for this methodology; a command line interface, and an exploration interface. The exploration interface provides data manipulation, visualization, and statistical queries. The significance and relatedness of attributes is easily compared. Furthermore, it is a convenient interface to run many different machine learning algorithm tests on, allowing changes to learning parameters, which data attributes are included, and which algorithms to use, while yielding easily comparable results. Any models that are generated may be saved, and used to predict output for new data. We can experiment on the WEKA explorer until we find a model which is deemed sufficiently accurate. The command line interface allows the execution of any models that are generated by WEKA to evaluate new data, which can easily be executed from the modified BEAGLE Puppy system. The predicted values can then be used to modify the selection process in the modified BEAGLE Puppy system. WEKA is implemented in Java, but there is no need to modify any WEKA code, or bridge the code to C++; the command line interface is sufficient for this methodology.

This implementation can be expanded to allow models to be generated by WEKA as evolution occurs. The accuracy of these models can be monitored until they are sufficiently accurate, in order to stop sampling evolvability and robustness, and instead, predict them. Sampling may still be interleaved to ensure the models remain accurate.

WEKA allows rapid prototyping of different machine learning methods, but does not offer as much flexibility as frameworks based upon a single, or even a few machine learning methods. Once artificial neural networks (ANNs) were identified as good predictors of evolvability, a more specialized framework for ANNs, Encog, replaced it.

A.4 Encog

Encog is a machine learning framework for Java and C# [34]. It is available as a JAR, DLL package, and source code library. It provides machine learning algorithms for classification, clustering, and regression. Its machine learning paradigms include ANNs, Bayesian networks, support vector machines, simple recurrent networks, even tree-based GP, among others. However, its speciality is ANNs. Encog models are implemented with a strong consideration for efficiency; it outperforms many other machine learning Java and C# libraries [40, 57, 80, 69]. Efficiency is critical for the ANN component of Model-Evolvability Genetic Programming (MEGP). We are interested in the gains that can come from modelling evolvability using ANNs, as an improvement over SGP. The underlying GP algorithm affects the SGP results, as well as the evolvability systems. Any gains brought about by changes in the evolvability systems would affect one using a different framework for GP equally. Furthermore, no modifications to the underlying ANN algorithm is necessary, so a rigorous, complex source code, that may not be easily modified, is of no consequence. Neural networks may be saved and used as initial positions for future training. Thus, Encog meets the needs of the evolvability systems.

Unlike EpochX, which required changes to the underlying source code in order to support the use of evolvability most effectively, the source code of Encog has no required changes. Training data for Encog is generated through the statistical faculties of EpochX, which may then be stored as a text file (for analysis) or as arrays (for speed). The data must be converted to a usable form for Encog, the MLDataSet. MLDataSets may be generated from text files, if they adhere to a certain formatting. Alternatively, they may be generated from input-output pairs, generated manually from EpochX statistics. Neural network architecture may be specified, and altered, according to how well the ANN is performing, as a measure of how accurately it may predict evolvability.

The training algorithm used for ANN fitting in the *ap*EGP and MEGP systems is resilient backpropagation [71]. Resilient backpropagation is similar to standard backpropagation, but it requires no set learning parameters, such as learning rate and momentum. Instead, learning rates are maintained separately for each weight, and each are adapted during training. Instead of using the magnitude of the gradient, combined with the learning rate and momentum, to adjust each weight, only the sign of the gradient is used, along with adapted values replacing the use of the learning rate and momentum. This usually translates to a more efficient training algorithm than backpropagation, at the cost of increased implementation complexity. Encog's full user manual encourages the use of resilient backpropagation as a general purpose training algorithm [42]. Different networks are interchangeable in Encog, so other training algorithms may be tested easily, if any show promise as being more efficient than resilient backpropagation. The structure of the network (number of hidden nodes) must still be specified, and affect the training time and accuracy of the model. Training algorithms can even be changed between generations. The most significant parameters affecting training time that may be adjusted are the terminating conditions, which are generally when the model is generating a small error rate or when a specified number of training generations for the ANNs (epochs, in the Encog parlance) have occurred. These may be adjusted dynamically in the evolvability systems, by considering how many epochs are acceptable for the increase in accuracy we may benefit from. We may measure the error rate of unseen, newer generation individuals compared to the error rate of the previously seen individuals, to determine if the model is no longer performing as well as necessary.

A.5 Integration

The EpochX GP framework serves as the foundation for the Sample-Evolvability Genetic Programming (SEGP) and MEGP systems. BEAGLE Puppy is appropriate for *a priori* Evolvability Genetic Programming (*ap*EGP), but more intrinsic GP changes are required to handle the more complex systems. Problems are defined in EpochX, and the existing code can be used as an SGP baseline, to compare with the evolvability systems. We determine if Encog training will occur for each new generation, and whether or not evolvability needs to be calculated or predicted with the existing model. If training occurs, we process the generational data so that Encog may use it, and use the existing network (or if there has not been any training yet, a random network) as the initial network. We then train the network with the new generational data, and any previous labelled generational data. If the network is accurate enough, successive generations may use the network to predict evolvability, instead of calculating it. Intermittently, evolvability may still be calculated and used to test the accuracy of the existing network, to determine if more training is needed with newer generations, in order to maintain its accuracy. This describes the integration of the EpochX and Encog frameworks in the evolvability systems. More specific implementation details regarding the use of evolvability is discussed in Chapter 5.

Statistics used in evolvability training, as well as time elapsed, fitness, and run number, are saved to a comma separated value file. These are loaded as data frames in an R interpreter. Data is aggregated by mean based on generation, which is then plotted. The plots describe the desired statistics (such as fitness or evolvability) under various conditions, such as the weight of evolvability selection. This is used to demonstrate the effectiveness of the system against SGP, and other changing-run conditions against one another.

A.6 Hardware

Experiments were conducted on hardware provided by ACENET, in partnership with Compute Canada, a high performance computing (HPC) consortium for universities based in the Atlantic Provinces of Canada. ACENET is funded by the Atlantic Provinces Opportunities Agency, the Canada Foundation for Innovation, and the provincial governments of Nova Scotia, New Brunswick, and Newfoundland and Labrador.

ACENET offers four HPC clusters: Mahone (based at St. Mary's University), Placentia (Memorial University of Newfoundland), Fundy (University of New Brunswick), and Glooscap (Dalhousie University). They have 532, 3756, 636, and 2196 cores, and RAM ranges per core of 4-16 Gb, 2-16 Gb, 4-8 Gb, and 1-8 Gb, respectively. Each uses Red Hat Enterprise Linux 6 as the operating system.

Clusters were accessed using WinSCP and PuTTY. Experiments were run as array jobs, which allow arbitrary numbers of tasks to be scheduled independently on a particular cluster, with the only difference between them being a task number, which may be used as a random number seed, and to track the run number. This allows many experiments to be run at once, allowing statistically significant results to be achieved in a much shorter period of time than running jobs serially on local hardware.

Appendix B

About the Author

The author is a PhD Candidate with a Master's degree in Computer Science, with theses based on machine learning. He has a largely academic background with experience in building machine learning systems with various frameworks and languages, with particular specialization in artificial neural networks and genetic programming. His Master's thesis work in artificial neural networks features multiple task learning and consolidating domain knowledge.

B.1 Selected Publications

B. Fowler and W. Banzhaf. Modelling Evolvability in Genetic Programming. In *Proceedings of the European Conference on Genetic Programming*, pages 215-229. Springer International Publishing, 2016.

V. Veloso de Melo, B. Fowler, W. Banzhaf. Evaluating methods for constant optimization of symbolic regression benchmark problems. In *Proceedings of the 2015 Brazilian Conference on Intelligent Systems*, pages 25-30. IEEE, 2015.

R.A. Enguehard, B. Fowler, O. Hoeber, R. Devillers, W. Banzhaf. Integrating human knowledge within a hybrid clustering-classification scheme for detecting patterns within large movement data sets. In *Proceedings of the Workshop on Complex Data Mining in a Geospatial Context*, pp. 18-21, 2012.

B. Fowler. *Context-Sensitive Multiple Task Learning with Consolidated Domain Knowledge.* Master's Thesis. Acadia University, 2010.

L. Tu, B. Fowler, D.L. Silver. CsMTL MLP For WEKA: Neural Network Learning with Inductive Transfer. In *Proceedings of the International Florida Artificial Intelligence Research Society Conference.* AAAI Press, 2010.

B.2 Education

Sept 2011-Present	In Progress: Doctor of Philosophy in Computer Science at **Memorial University of Newfoundland**, St. John's, NL Thesis: Modelling Evolvability in Genetic Programming Advisor: Dr. Wolfgang Banzhaf
Sept 2008-Dec 2010	Master of Science in Computer Science at **Acadia University**, Wolfville, NS Thesis: Context-Sensitive Multiple Task Learning with Consolidated Domain Knowledge Advisor: Dr. Daniel Silver
Sept 2004-May 2008	Bachelor of Science with Distinction in Mathematics at **Mount Allison University**, Sackville, NB