NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

# A Proposal for Generating GSGP-Hard Datasets

## An introductory research towards GSGP hardness

Cornelis Marnik Verschoor

Dissertation presented as partial requirement for obtaining the Master's degree in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

**NOVA Information Management School**

**Instituto Superior de Estatística e Gestão de Informação**

Universidade Nova de Lisboa

# A PROPOSAL FOR GENERATING GSGP HARD DATASETS

by

Cornelis Marnik Verschoor

Dissertation report presented as partial requirement for obtaining the Master's degree in Advanced Analytics

**Advisor:** Leonardo Vanneschi

September 2018

## A PROPOSAL FOR GENERATING GSGP HARD DATASETS

# Acknowledgements

As to be expected, writing this thesis did not come without its obstacles, robbing me of at least some of my motivation at numerous occasions. Luckily, I've found myself surrounded with people who could aid me with their expertise or simply with their presence and support, keeping me on track towards finishing this thesis.

I would first and foremost like to thank professor Leonardo Vanneschi of the Information Management School at Universidade Nova de Lisboa. His expertise on the subject of this thesis has been invaluable to me during numerous stages into this research. His enthusiasm on the subject gave me a motivational boost after each meeting we had and his willingness to help is admirable and appreciated. Despite his busy schedule, he was always readily available for me whenever I had any questions. This thesis would not have been what it is now without his help.

I would furthermore like to thank my family and friends. Their presence has been of great importance to me throughout my life and during this research. They provided me with constant support, even in the last months when this thesis kept me from giving them the attention they deserve. Besides their support, they also provide me with much needed distraction.

In particular I would like to mention my parents. To thank them for all the things they have done for me in my life so far would require a thesis on itself, therefore I'd like to concretely express my gratitude for the last 2 years. Without them I would not have been able to move to Lisbon, where I attended my master's degree and was privileged to live in one of Europe's finest cities (in my humble opinion). Their support and encouragement enabled me to complete this thesis and with it my master's degree. Maybe even more so, they enabled me to grow as a person.

Thank you

Marnik Verschoor

# Abstract

Since its introduction, GSGP has been very successful in optimizing symbolic regression problems. GSGP has shown great convergence ability, as well as generalization ability. Almost without exception, GSGP outperforms GP on real life applications. These results give the impression that almost all symbolic regression problems are easy for GSGP to solve. However, little to no research has been done towards GSGP hardness, while GP hardness has been researched quite extensively already. The lack of real life applications on which GSGP has more difficulty converging than GP creates a paradox: if there are no, or very little, problems that are GSGP hard, no research can be done towards it. In this paper we propose an algorithm that generates and evolves datasets on which GP outperforms GSGP, under the condition that the GP model remains as accurate as possible. The algorithm can also be altered so that it produces datasets in which GSGP outperforms GP. This allows for comparing GP hard datasets with GSGP hard datasets. The algorithm has shown to be able to produce favorable datasets for both GP and GSGP using multiple settings for the number of instances and the number of variables. Therefore, the algorithm proposed in this paper breaks the earlier mentioned paradox by producing GSGP hard datasets, thus allowing GSGP hardness to be effectively researched for the first time. Furthermore, tuning the algorithm led to some early observations about the relation between dataset composition and GP/GSGP performance. GSGP has difficulty converging when using only 1 dependent variable and 1 independent variable, while it is easy to produce datasets on which GP heavily outperforms GSGP with the same settings. GSGP performs better when more independent variables are added. Furthermore, a big range of the dataset has been shown to be beneficial for GP convergence, while a small range is beneficial for GSGP convergence.

**Keywords:** Geometric Programming; Geometric Semantic Genetic Programming; Particle Swarm Optimizaion; Multi Objective Optimization; Multi Objective Particle Swarm Optimizaion; GP hardness; GSGP hardness

# Contents

# List of Figures

# List of Tables

# Listings

# Acronyms

**GBest** Global Best. A concept of Particle Swarm Optimization. The best solution found by all particles through all epochs (see Glossary).

**GP** Genetic Programming. A heuristic based optimization technique that uses tree structured genes to encode computer programs.

**GSGP** Geometric Semantic Semantic Genetic Programming. An altered implementation of the Genetic Programming algorithm that uses geometric operators to search semantic space.

**LBest** Local Best. A concept of Particle Swarm Optimization. The personal best solution found by a particle (see Glossary).

**MOOP** Multi Objective Optimization Problem. An optimization problem that has more than one objective function that needs to be maximized/minimized.

**MOPSO** Multi Objective Particle Swarm Optimization. A particle swarm optimization algorithm that uses more than 1 objective function.

**NRMSE** Normalized Root Mean Square Error. Root Mean Squared Error divided by the standard deviation of the dependent variable.

**PSO** Particle Swarm Optimization. A heuristic based optimization problem algorithm based on animal flock behavior.

**RMSE** Root Mean Square Error. A measure that indicates the accuracy of a model.

**VMax** Velocity Maximum. A concept of Particle Swarm Optimization. The maximum value for velocity (see Glossary).

**VMin** Velocity Minimum. A concept of Particle Swarm Optimization. The minimum value for velocity (see Glossary).

# Terminology

Because of the use of multiple algorithms throughout this thesis there are a lot of different concepts and definitions that will be mentioned interchangeably throughout this paper. Some of these concepts could cause confusion because they are quite similar. For example, both Particle Swarm Optimization and Genetic Programming (algorithms used to optimize our problem, as discussed later) evolve their individual solutions by going through multiple iterations. If the word iteration would be used, it would not be clear for the reader whether the iteration of a GP/GSGP run or a PSO run is meant. It is important to have a clear distinction between concepts such as these so that there will be no confusion when reading this paper. It is also simply useful to have common terminology for often occurring definitions. For these purposes, below is a small list of definitions with their respective meanings.

**Definitions for Genotype of the solutions**

- **Data Point:** A point in n dimensional space, containing values of all dependent variables and the independent variable of one instance.
- **Dependent variable:** The variable that needs to be predicted by the symbolic regression engine, also known as the target variable.
- **Independent variable:** Variable that is used to build a model to predict the dependent variable.
- **Instances:** Observations in the dataset.

**Definitions for Particle Swarm Optimization**

- **Cognitive Learning Factor:** Factor that determines how much a particle is steered towards its own best found solution.
- **Epoch:** One iteration in the PSO algorithm.
- **GBest:** The best solution found by all particles through all epochs. The whole algorithm has only one GBest value.
- **LBest:** The best solution found by an individual particle through all executed epochs.
- **Particle:** An individual member of the swarm.

- **PBest:** The personal best solution found by a particle. Every particle has their own PBest value.
- **Position:** The genotype of the solution, which is a dataset containing of $m$ instances and $n$ dimensions.
- **Social Learning Factor:** Factor that determines how much a particle is steered towards the global best found solution.
- **Swarm:** The entire population, which is made up of all the particles.
- **Velocity:** The direction and speed in which a particle "flies" through the search space.
- **VMax/VMin:** The maximum and minimum values for velocity.

**Definitions for (Geometric Semantic) Genetic Programming**

- **Error:** The difference to the global optima, or the fitness of an individual
- **Generation:** One iteration of the GP or GSGP algorithm.
- **Individual:** An individual member of the GP/GSGP population.
- **Population:** All solutions of the GP and GSGP algorithms, consisting of all the individuals

**Definitions for Multi Objective Particle Swarm Optimization**

- **Local Pareto Front:** The multi-objective equivalent of a local optimum: a Pareto front outputted by the algorithm that is not the true/global Pareto front.
- **Pareto Front:** The hyperplane in objective space on which the individuals lie with the best possible tradeoff between different objective functions.
- **Pareto Particle:** A particle of which it's position is a member of the non-dominated set.
- **Non-Dominated Set:** A set of particles of which it's position is not dominated by any other particle's position found so far.

# 1 Introduction

In the field of symbolic regression, Genetic Programming (GP)[1] has been a popular algorithm to solve optimization problems. Geometric Semantic Genetic Programming (GSGP) [2] is an alteration of GP using geometric crossovers, which has been shown to outperform GP in real life applications [3] [4] [5]. As this introduction will further point out, numerous studies have been conducted to GP hardness but little to no research has been done towards GSGP hardness. Research towards GSGP hardness has so far been impossible due to the excellent performance of GSGP, leaving no to very little datasets which hare GSGP hard. The objective of this paper is to address this lack of GSGP hard datasets by designing and implementing an algorithm that generates datasets that are harder to solve for GSGP than for GP. This allows for effectively analyzing GSGP hardness for the first time.

Since the introduction of GP in 1992 [1] it has been widely studied and used in many fields, like climatology [6], cryptanalysis [7] and finance [8]. In 2011, M. Korns [9] researched accuracy in state-of-the-art symbolic regression. In said research, GP was used for the experiments and it was concluded that a serious issue exists for symbolic regression engines; even though a solution is returned with good fitness, this doesn't mean that the correct formula is returned. In fact, even for very simple test problems consisting of one basis function with no noise and only a few features, GP is very often not able to return the correct formula to the user. It is stated that even simple test problems can have very large search spaces, hence the poor performance in accuracy.

To overcome this problem and to improve GP in general, different alternative implementations of the GP algorithm have been proposed throughout the years. One of those alternatives is Geometric Genetic Semantic Programming (GSGP) [2]. GSGP uses geometric operators that allow the algorithm to search in semantic space, thus being able to transform a solution's genotype based on an alteration in its phenotype. The main advantage of this approach is that this creates a unimodal fitness landscape for any supervised learning problem, turning even complex fitness landscapes into simple fitness landscapes. The test results from Moraglio et al. [2] showed that this new implementation heavily outperformed GP.

The biggest limitation of GSGP on its introduction however, was the size of the solution. The geometric operators caused the solutions to grow exponentially, rendering the program useless even within just a few generations because of the extensive computational power required to build and compute an individual. Vanneschi et al. [3] proposed a solution for this problem by using references (or memory pointers) instead of actually building every individual with every generation. This approach requires storing only the original population and the random trees (used by crossover and mutation) that are still used by any individual in the current generation. A third reference table is then used that holds the current solutions by "pointing" to the parent(s), the operator used and the random tree(s) used during the operation. This way, the individual is never actually build and only the semantics need to be computed.

This still left researchers with the problem that upon finishing the algorithm the solution is still too big to be constructed and thus to be used externally or logically interpreted (which has been recently addressed by J.F. Martins et al [10]), but it did allow for GSGP to be used on real life data. Upon using GSGP on real life data for the first time, it was shown that GSGP outperforms GP on training data and also, surprisingly at first, in terms of generalization ability [3] [4] [5]. After that, GSGP has successfully been applied to many different real life problems, like predicting concrete strength [11] [12], predicting the severity of Parkinson's disease symptoms [13] and sentiment analysis [14].

GSGP's performance on training data decreased by a little bit when a small alteration was made to the mutation [3] than originally proposed when GSGP was first introduced [2] by adding a logistic function to the random trees that are generated. This limits the output of the random trees in the interval [0,1]. Gonçalves et al. [15] researched the implications of this small alteration and found that the training error got higher with the new mutation implemented by Vanneschi et al. [3], but it performed significantly better in terms of generalization ability. The mutation was named bounded (BM) mutation [15] and became the standard for GSGP. Even though the training error of GSGP with BM on was higher than GSGP without BM, it still outperformed GP on all but one dataset in said research and continues to outperform GP in other applications.

## 1.1 GP Hardness vs GSGP Hardness

It is known that there are a lot of problems where GP has difficulties converging to a desirable solution. Several researches have been performed towards GP hardness. M. Korns [9] researched several simple benchmark problems and found that some were hard for GP to solve. K.E. Kinnear [16] researched GP problem difficulty early after GP's inception by examining the structure of the fitness landscapes, which showed good correlation with problem difficulty. J.Daida et al. [17] investigated the role of structure on GP hardness. Their hypothesis was that structural mechanisms existed

[18] [19]. This implies that all possible tree structures can be divided into different categories based on a scale of hardness. The research states that most structures are difficult to create for GP, which might contribute to GP hardness.

M. Clerque et al. [20] researched if the fitness distance correlation (fdc), used earlier for measuring Genetic Algorithm difficulty [21], could also be used to measure GP problem difficulty and concluded that it is indeed a suitable problem difficulty metric. Vanneschi et al. [22] continued the research towards fdc as a GP difficulty metric and confirmed that fdc is a suitable metric for this purpose but with some limitations, e.g. the existence of counterexamples (functions that are easy for GP to optimize while fdc says the opposite) and the global optima has to be known a priori. Therefore, they proposed another measure for GP problem difficulty that overcomes these limitations; negative slope coefficient (nsc). Vanneschi et al. [23] expanded on this research by pointing out that the nsc is not a reliable measure if a fitness cloud [24] is partitioned into arbitrary segments and proposed a more suitable way of partitioning the fitness cloud.

The research that has been conducted on GP problem difficulty is extensive and doesn't limit itself to the researches mentioned above. While GP hardness has been researched quite well, there is no research to be found on GSGP hardness. In fact, the impression that the good performance of GSGP has raised seems to be that virtually any problem is easy to solve for GSGP. Not only is this contradictory to the No Free Lunch theorem [25], it is also unsubstantial to make such a statement without adequate research available towards GSGP problem difficulty. In this research we attempt to address the lack of research to GSGP hardness. Concretely, we propose an algorithm that generates datasets on which GP outperforms GSGP. Alternatively, the algorithm can be tuned to generate datasets in which GSGP performs better than GP, so that the results of both can be compared. A lot of different parameters come into play when designing such an algorithm. This paper presents the algorithm, its different parameters, the effect that different parameter values have on the outcome, and finally reports and discusses the findings that were derived from the results.

# Background 2

This section explains the theory of the different algorithms and concepts that are used throughout this research: the Particle Swarm Optimization algorithm, the concepts of multi-objective optimization and Multi Objective Particle Swarm Optimization.

## 2.1 Particle Swarm Optimization

The original algorithm is a basic Particle Swarm Optimization (PSO) [26]. PSO is a heuristic based optimization technique which is based on animal flock behavior using swarm intelligence. One of the key features of PSO is that besides that solutions learn from their own search history, it is also social, i.e. the different solutions within the algorithm don't just act independently from each other but they share information so that they can learn from each other. A couple of characteristics of the PSO algorithm made it a suitable choice for this research. Firstly, it is designed for continuous optimization problems, which is exactly the kind of problem we are dealing with in this paper. Secondly, considering the many different parameters already involved in this algorithm, PSO requires just a few parameters to adjust which reduces the complexity of an already complex algorithm. Thirdly, PSO is known to require low computational cost. This is a plus considering the heavy fitness function that is used for the algorithm (as will be presented later).

The entire population in PSO is called a **swarm**, which consists of **particles**. Each particle is an individual member of the swarm. **GBest** stands for **global best** and denotes the best solution found by all particles over all executed iterations. An iteration in PSO is called an **epoch**. **PBest** stands for **personal best** and denotes the personal best found by a particle over all executed epochs, which means there are as many PBest values as there are particles. **Velocity** is the speed in which a particle "flies" through the search space. **VMax** and **VMin** are the maximum and minimum boundaries for the velocity value. The **cognitive learning factor** is a factor that determines how much a particle is steered towards its PBest value and the **social learning factor** is a factor that determines how much a particle is steered towards GBest. Every epoch the velocity is

calculated for every particle, which is then used to update the position of every particle, where the position is the genotype of a particle. Equation 2.1 is used to update the velocity for a particle, after which 2.2 is used to update the position of a particle.

$$V_i(t+1) = V_i(t) + c_1 r_{i1} \times (pbest_i(t) - X_i(t)) + c_2 r_{i2} \times (gbest(t) - X_i(t)) \tag{2.1}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{2.2}$$

Where i=1,2,3...,n and n is the number of particles in a swarm. $X_i(t)$ is the position of the particle $i$ at epoch t and $V_i(t)$ is the velocity of particle $i$ at epoch $t$. $pbest_i(t)$ denotes the $i^th$ particle personal best found up and until the $t^th$ epoch and $gbest(t)$ denotes the global best found up and until the $t^th$ epoch. $c1$ is the cognitive learning factor and $c2$ is the social learning factor. $r1$ and $r2$ are both random real value numbers between the interval of $[0,1]$, which means they are a factor for the proportion in which a particle is steered towards the global best and how much it is steered towards their local best. The pseudocode of a basic PSO is given below:

**Listing 2.1:** Pseudocode for a basic PSO algorithm

```
Randomly initialize swarm
Evaluate each particle
    While termination condition is not met
        For every particle
            Compute velocity according to Equation 1
            If velocity > VMax
                Velocity = VMax
            If velocity < VMin
                Velocity = VMin
        Update position according to Equation 2
    For every particle
        Evaluate Particle
        If PBest is worse than current fitness
            Current position is set to be new PBest
        If GBest is worse than PBest
    PBest is set to be new GBest
Until termination condition is met
```

## 2.2 Multi Objective Optimization

Multi Objective Particle Swarm Optimization (MOPSO) is a PSO that is adapted to work with multi-objective optimization problems (MOOP) [27]. MOOP has 2 or more objectives that have to be optimized, where all different objectives are optimized simultaneously. A solution to a multi-objective optimization problem is the best possible tradeoff between all given objectives. Formally, a MOOP can be represented as:

$$\begin{aligned} &\text{Minimize/Maximize } f_i(x), i = 1, 2, \ldots, I \\ &\text{Subject to } g_j(x) \geq 0, h = 1, 2, \ldots, J \\ &\qquad h_k(x) = 0, h = 1, 2, \ldots, K \\ &\qquad x_m^L \leq xi \leq x_m^L, m = 1, 2, \ldots, M \end{aligned} \tag{2.3}$$

Where $x$ is the decision vector and $f_i$ is the $i^th$ objective function. $g_j(x)$ and $h_k(x)$ are inequality and equality constraints respectively. The last constraint, $x_m^L \leq x_i \leq x_m^L$, constrains the variable $x_i$ to be within the lower bound $x_m^L$ and the upper bound $x_m^L$. A solution is said to be feasible if all constraints and variable limits are met. If it doesn't, the solution is called infeasible. All feasible solutions together are called the feasible region. Comparing different solutions with a MOOP is more complex than with a single objective optimization problem since the results of more than 1 objective function have to be compared with each other. A way of comparing solutions in a MOOP is based on Pareto dominance [28] [29]. Assuming a maximization problem on all objective functions, the concept of dominance states that a solution x dominates a solution y (written as $x \prec y$) when two conditions are met:

1. *x is no worse than y for all objective values:*

   $f_i(x_i) \geq f_i(x_k); \forall k = 1, 2, 3, \ldots, N$

2. *x is strictly better than y in at least one objective function:*

   $f_i(x_i) > f_i(x_k)$; for at least one $k \in 1, 2, 3, \ldots, N$

Pareto dominance uses the concept of dominance to rank solutions of a MOOP. Pareto dominance uses this ranking system to return a set of possible solutions. The ranking system is explained by a number of definitions [30] [31]:

**Definition 1 (Non Dominated Set)**
*A vector of decision variables $x \in S \subset \mathbb{R}^n$ is said to be non-dominated if there exists no other $x' \in S$ such that $f(x^i) \prec f(x)$.*

**Definition 2 (Pareto Optimality)**
*A vector of decision variables $x \in F \in \mathbb{R}$ is said to be Pareto optimal if it is non-dominated with respect to the feasible region $F$.*

**Definition 3 (Pareto Optimal Set)**
*The Pareto optimal set contains all Pareto optimal solutions. Formally, the Pareto optimal set $P^*$ is defined as $P^* = \{x \in F | x \text{ is Pareto optimal}\}$*

**Definition 4 (Pareto Front)**
*The Pareto front $PF*$ are all the Pareto optimal solutions positions in the objective space. The Pareto front $PF^*$ is formally defined as $PF^* = \{f(x) \in \mathbb{R}^k | f(x) \in P^*\}$*

A MOOP based on pareto dominance thus tries to determine the pareto optimal set from the set $F$, which we call a globally Pareto front. Just like finding the global best in a single objective optimization problem is not a certainty, finding the global pareto optimal set is also not always achievable. A Pareto front that is produced by an algorithm that is not the true Pareto front is called a local Pareto front.

## 2.3 Multi Objective Particle Swarm Optimization

A number of changes have to be made to the original PSO algorithm so that it can optimize multiple objective functions based on pareto dominance, which is called Multi Objective Particle Swarm Optimization (MOPSO) [32]. Firstly, the algorithm needs to store the non-dominated particles. Note that when we say non dominated particles, we refer to the non-dominated positions of the particles. These non-dominated particles are called leaders. Any of the leaders can be used to guide the search of a particle and every particle can choose a different leader at epoch t. This leads to a slightly altered velocity function as can be seen in equation 2.4.

$$V_i(t+1) = V_i(t) + c_1 r_{i1} \times (pbest_i(t) - X_i(t)) + c_2 r_{i2} \times (P_h - X_i(t)) \tag{2.4}$$

Where $P$ is the external archive containing the non-dominated particles, $P_h$ is a single leader that is selected from it and h is selected randomly from the non-dominated set. The external archive is updated with each epoch, at which all the currently non dominated solutions are entered into the archive and any solution in the archive that is now dominated is removed from the archive. Finally, PBest is chosen using Pareto dominance. If PBest and the new position both don't dominate each other, one of them is randomly selected. The procedure of MOPSO is given in listing 2.2.

**Listing 2.2:** Pseudocode for a basic MOPSO algorithm

```
Randomly initialize swarm
Evaluate each particle in the swarm
Setup non dominant set S
While termination condition is not met
    For every particle
        Compute velocity according to equation 2.4
        If velocity > VMax
            Velocity = VMax
        If velocity < VMin
            Velocity = VMin
        Update position according to equation 2.2
    For every particle
        Evaluate Particle
        If current position < PBest
            Current position is set to be new PBest
        If both current position and PBest don't dominate each other
            Randomly choose PBest
        Update non-dominated set S
Until termination condition is met
```

# Methodology & Results 3

The main goal of the experiments that follow is to design and implement an algorithm which outputs a dataset consisting of dependent and independent variables, on which GP outperforms GSGP for symbolic regression problems. These solutions would then form a solid basis towards analyzing GSGP problem hardness, of which little to nothing is known to date. We want to emphasize that the focus is on designing and implementing an algorithm that is successfully able to produce solutions that meet the stated goal. However, analyzing these results so that definitive conclusions can be made as to what characteristics of a dataset define what makes an optimization problem GSGP hard is not the main goal of this thesis. It is however desirable and the aim is to draw conclusions wherever possible, but the main goal is to provide a foundation that can be used in future research towards GSGP hardness.

The original thought behind the thesis was simple: *write an algorithm that produces data sets in which GP outperforms GSGP.* It was at first also believed that the implementation of such an algorithm would prove to be not too complex either. As so often happens, reality proved differently and this thesis and the research behind it grew a whole lot bigger than initially expected. Actually, this research consists of multiple researches. Each research building upon the results of the previous researches: the first results led to observations, which led to the conclusion that the algorithm had to be altered in a way such that more desirable results would be achieved. The alterations led to new results, which led to new observations, which led to new changes in the algorithm and so on.

This section will therefore be divided into 5 sub-sections, where each sub-section represents a research phase. Each research phase is characterized by its own implementation of the Optimal Dataset Finder (ODF) algorithm and/or its own research goals. Every research phase therefore has its own experimental setup and results. Since every subsequent research phase builds on the results of the previous research phase, the results will be presented and discussed for every research phase separately. Even within a single research phase it is possible that the interpretation of experimental results led to new experiments within the same research phase. These sub-experiments still using the same algorithm and they are run with the same goals in mind that are applicable

to that research phase. This means that it is not uncommon for small new experiments to be proposed (and executed) in the results section of a given research phase.

Phase I proposes the initial ODF algorithm and runs the first experiments with it. Phase II proposes a new version of the ODF algorithm that is able to handle multiple objective functions. Phase III focuses on solving issues with the ODF that arose during Phase II and were deemed critical to solve. Phase IV goes in depth to different parameter settings and the effect they have on single objective functions. Phase V combines all knowledge from the previous research phases to try and evolve solutions that satisfy our goals. Lastly, the final ODF algorithm is shortly reviewed.

## 3.1 Phase I

Phase 1 presents the initial design of the ODF algorithm. The purpose of the algorithm is to output a data set in which GP outperforms GSGP. This means that the genotype of every particle is a matrix of size $mn$ consisting of real numbers, where $m$ is the number of instances and $n$ is the number of dimensions. This is also the position of a particle and therefore represents the matrix $X$ in equations 2.1, 2.2 and 2.4. The variables in a symbolic regression problem consist of independent variables and dependent variables. In our case, we are always predicting 1 dependent variable. This means that there are $n-1$ independent variables. The learning process for these datasets is a PSO run as described in section 2.1 at which in every epoch the velocity and positions are computed for each particle.

The fitness function is simple in terms of definition but complex in terms of computational effort. We are looking for datasets that are more suitable for GP than for GSGP. This translates itself into a maximization problem where the fitness metric is the difference between the fitness of GSGP and the fitness of GP. This means that both GP and GSGP have to be executed on the dataset in order to compute the fitness, which is expected to be computationally extensive but a necessity none the less. Formally, the fitness function $f$ can be defined as follows:

$$f(X) = g(X) - h(X) \tag{3.1}$$

Where $g(X)$ is the fitness output of GSGP, $h(X)$ is the fitness output of GP and $f(X)$ needs to be maximized. Informally, the fitness output is the difference between the fitness of GSGP and the fitness of GP. GP and GSGP both use Root Mean Square Error (RMSE) as their fitness metric. It is formally defined in equation 3.2

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y} - y)^2}{n}} \tag{3.2}$$

Where $n$ is the number of instances, $y$ is the predicted value of the dependent variable for the $i^th$ instance and $\hat{y}$ is the actual value of the dependent variable for

the $i^th$ instance. The lower the RMSE, the better the individual. To avoid confusion between the fitness of the PSO algorithm as outputted by equation 3.1 and the fitness of GP and GSGP as outputted by equation 3.2, GP and GSGP fitness will from now on be referred to as simply GP Error and GSGP Error. The PSO fitness will be referred to as delta error. The PSO algorithm updates the velocity of a particle using equation 2.1 after which the position of the particle is updated using 2.2.

### 3.1.1 Experimental Setup

The number of variables is set to 2, so this means that we have 1 independent variable and 1 dependent variable. The reason we choose this number is for plotting purposes. Using only 2 variables makes it possible to plot the independent variable against the dependent variable on a 2D plane. This allows for visually analyzing and interpreting the outputted datasets such that we might be able to identify patterns and/or functions that determine what makes a dataset easier to solve for GP than for GSGP. The number of instances is set to 100 and the particles positions are initialized within the range [-100:100]. The swarm consists of 50 particles and the particles will evolve for 50 epochs. No other termination condition is set because we want the delta error to be as high as possible. Different values of VMax are used to see the effect it has on the delta error. Every variant of the PSO was run 10 times. The results presented in this paper are the median al of those 10 runs. Ideally, the number of runs would be higher but the heavy computational cost of the algorithm limits us in the number of runs (as will be pointed out later). For now this is deemed sufficient to get a good impression of the performance of the algorithm. Once the algorithm reaches a more final stage, solutions will be proposed to address the heuristic nature of the algorithm.

The algorithms used for the PSO fitness function are canonical GP and canonical GSGP. The function set consists of the four binary arithmetic operators +, -, * and / *protected* as used in early studies of GSGP [3] [4] [5] and as in [1]. The terminal set contains constants within the range [-1:1] with increments of 0.25, totaling to 9 constants in total. The total number of terminals will therefore be $9 + n$, where n is the number of independent variables. The number of individuals is fixed to 50 and all GP and GSGP runs iterate over 100 generations. It might be argued that these values are low but they are set low intentionally. Keeping these settings low saves a lot of computational cost in an already computational expensive program. If these settings prove to be too low after analyzing the results, the settings will be altered accordingly.

The initialization method used is Ramped Half and Half [1] with maximum initial depth set to 6 and selection is done with tournament size 2. GP uses standard crossover [1] and mutation [1], where crossover probability is set to 0.9 and mutation probability to 0.1. This means that crossover will be performed 90% of the time and individuals will be mutated 10% of the time. GP has a depth limit that is set to 17 to limit computational cost. GSGP uses a crossover rate of 0.0 and a mutation rate of

1.0, meaning that all the perturbations to individuals are done by mutation only as proposed in [15]. The only small alteration made to the canonical GSGP is that it uses bounded mutation (BM) [15]. Reason for this is because in real life applications GSGP would most likely use BM as well because of its good generalization ability. We are aware of the heuristic nature of both GP and GSGP but choose to run both algorithms only once none the less. Our first goal is to implement an algorithm that is able to produce suitable solutions. If we are able to reach this stage the heuristic nature of both GP and GSGP will be addressed accordingly. This approach saves much needed time in finding the right settings for the algorithm.

### 3.1.2 Results

The experimental results are presented through the curves of the delta error and the curve of the GP Error and describe the initial results of running the algorithm and the effect changing VMax has on the results. GP Errors are analyzed as well because it can lead to more insights about the behavior of the ODF. The plots mentioned above represent the median values for 10 PSO runs. We like the emphasize that we are aware that this is low for a heuristic based algorithm, but as we mentioned earlier this is a necessity because of the heavy fitness function. The heuristic nature will be further addressed at the more final stages of the algorithm. This is discussed more thoroughly in section 3.1.1. The median value is preferred over average values because of its robustness to outliers. The position of GBest after the last epoch is also plotted. These results are plotted against the predicted outputs of GP and GSGP.

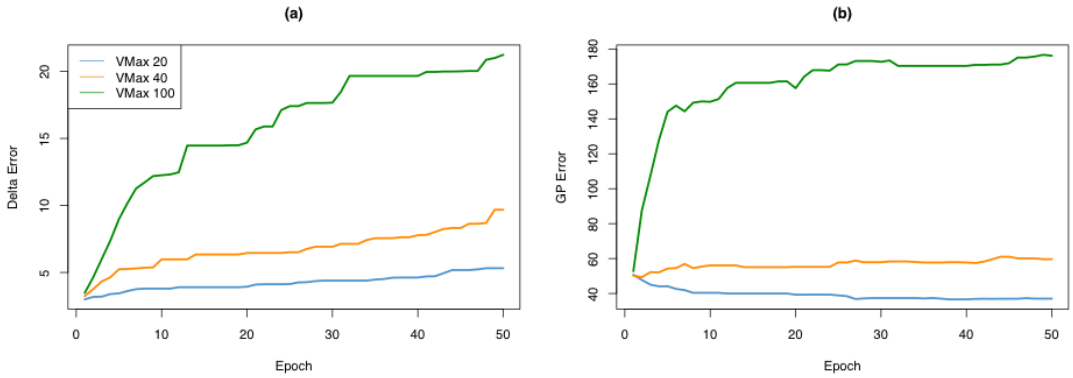


**Figure 3.1:** Results of running ODF V1 for the first time. Evolution of (a) delta error and the corresponding (b) GP error. The delta error denotes the difference between the GP error and GSGP error in favor of GP: the higher the delta error, the higher the dataset favorability is towards GP. The GP error is the RMSE value of the final solution produced by GP. Median values over 10 runs.

Plots (a) and (b) of Figure 3.1 show the evolution of the median delta error and the median GP error over all epochs. Three different values of VMax were used for this test: 20, 40 and 100. The results in plot (a) clearly show that the delta error rises

significantly when raising VMax. Even more so, the higher VMax, the longer and quicker the delta error rises. There seems to be no decline in the increase of delta error at 50 epochs when VMax is set to 100. These results would indicate that the algorithm is working well, as in it is doing what it is supposed to do: maximizing the delta error. Plot (b) shows that the GP error also grows significantly when raising VMax. In the early epochs the GP error skyrockets upwards when VMax is set to 100. After that the drastic increase in GP error makes room for a more steady, fluctuating increase during the remaining epochs. PSO with VMax set to 20 is the only run that decreases the GP error but upon finishing it has a delta error that is almost identical to the delta error at initialization.

Figure 3.2 shows the position of GBest for the median run for every different VMax value. Each data point represents the independent variable (X axis) against the dependent variable (Y axis). The predicted outputs of GP and GSGP of GBest are plotted as well. All different VMax settings have an almost perfectly linear line running through approximately the center of the data points of GBest. What is also a common occurrence is that all the positions have some extreme values which GP is able to predict pretty accurately, while GSGP is not able to predict them at all. In general the positions still look very random but GP is able to adjust to the randomness a little more than GSGP. This explains why GP is "outperforming" GSGP as per the definitions of the algorithm. The range gets bigger for both GP and GSGP but their models stay the same, almost linearly running through the mean of the data points. However, since GP is able to predict some extreme values quite accurately, it is able to "outperform" GSGP in terms of delta error, but it comes with an increase of the GP error as well.

### 3.1.3 Discussion

it is safe to conclude that both the GP and GSGP models are equally bad for all different VMax values. In the field of symbolic regression we would not be interested in a model that takes the mean of data points (assuming the data points are spread out), nor are we interested in a model that does the same except for predicting 1 or 2% of the independent variable while totally missing the others. One can argue how suitable a solution with high delta error is when the GP Error is very high as well. What good would a dataset that's beneficial for GP be, if GP itself performs very poorly too? It could informally be stated that both GP and GSGP perform badly, but GP is the algorithm that performs a little less bad. If both algorithms perform very poorly, the results might be neglectable since any algorithm with this kind of performance wouldn't stand in real life applications. Ideally we would want the difference to be as high as possible, while the GP error stays at low as possible. This indicates that we are not dealing with a single objective optimization problem but with a multi-objective optimization problem instead.

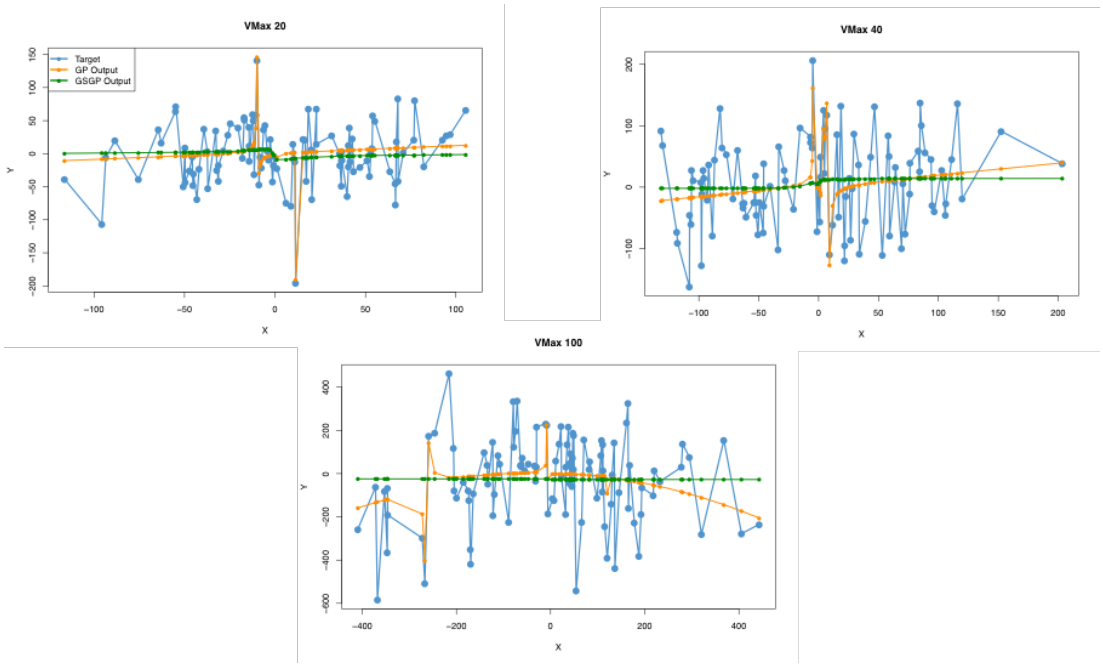


**Figure 3.2:** GBest position plotted against the predicted outputs of GP and the predicted outputs of GSGP. Positions are given for VMax values of 20, 40 and 100. Each plot is the GBest position of the median run for the respective parameters. The y axis holds the independent variable and the x axis holds the dependent variable

## 3.2 Phase II

Phase II proposes a multi objective version of the original ODF algorithm. The goal of this phase is to successfully implement a multi objective variant of the ODF and to test different parameter settings to see which set of settings result in the best solutions.

### 3.2.1 Experimental Setup

The ODF was altered into a Multi-Objective Particle Swarm Optimization (MOPSO) as described in Section 2.3 consisting of 2 objective functions. The first objective function is the delta error as computed in Equation 3.1, which will have to be maximized. This function was also used in the first experiments as the only fitness function. The second objective function is $h(x)$ from Equation 3.1, which denotes the GP error that is computed as in 3.2. The GP error is a minimization problem. In short, we are trying to maximize the delta error while simultaneously trying to minimize GP error. Since we do not know what kind of dataset patterns/functions we are looking for, there are no constraints that the solutions need to satisfy. All parameter settings were kept the same as in the previous experimental settings.

### 3.2.2 Results

Unlike the previous single objective PSO, MOPSO produces a set of possible solutions instead of just a single solution. This set of solutions will from now on be referred to as $S$. A single solution in $S$ will from now on be named a pareto particle. Having multiple solutions returned leads to a different way of presenting and interpreting the results. MOPSO is run 10 times. The median run is not used because selecting the median run is not as straight forward when there's multiple solutions with different objective functions, since this greatly limits the possibility of aggregating the results and comparing them. Instead, one run is selected manually by estimating which run produced a non-dominated set $S$ that appears to hold pareto particles with average objective function values.

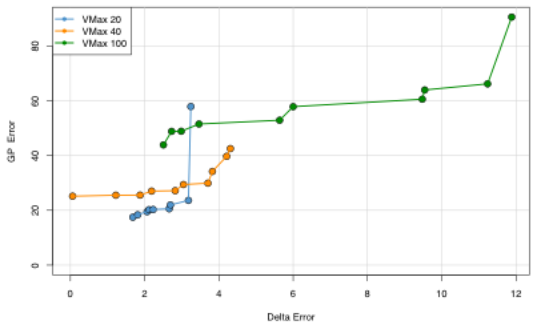


**Figure 3.3:** Local Pareto front of the median non dominant sets out of 10 MOPSO runs for VMax values of 20, 40 and 100. Each point in the plot represents a Pareto particle in the objective space.

Figure 3.3 shows the local Pareto front for S, where the local Pareto front refers to the Pareto front that is produced by the ODF but does not necessarily resemble the true/global Pareto front. VMax set to 20 produced 9 pareto particles ranging approximately between a delta error of 1.6 and 3.2 and a GP error between 19 and 60. VMax set to 40 produced 10 particles ranging approximately between a delta error of 0 and 4.3 and a GP error between 25 and 43. VMax set to 100 produced 10 particles ranging approximately between a delta error of 2.5 and 12.5 and a GP error of 41 and 90. On first sight, VMax set to 40 seems to have produced the best set of solutions since the delta error is bigger than VMax 20 but it doesn't increase the GP error as much as VMax 100 does. However, a GP error of 25+ still seems to be too high to represent even a slightly accurate model. A pattern can also be noticed in the different Pareto fronts of Figure 3.3. The higher VMax is set, the higher the GP error seems to become. To further try to explain this phenomena, Figure 3.4 plots the position of one pareto particle for each VMax setting. The Pareto particles that are chosen to plot are the Pareto particle with the lowest GP error, the median GP error and the highest GP error over all different settings.

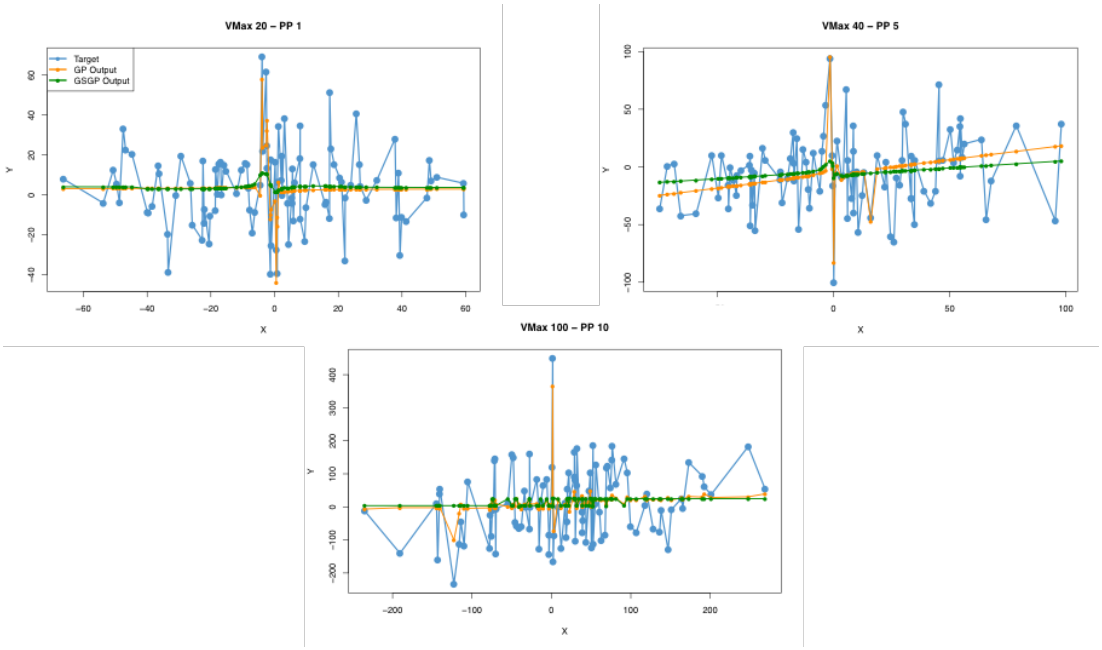


**Figure 3.4:** Positions of pareto particles from different non dominant sets, produced by different VMax settings. 'PP' in the titles stands for pareto particle. VMax 20 – PP 1 has a GP error of 19, VMax 40 – PP5 has a GP error of 28 and VMax 100 – PP 10 has a GP error of 90.

Figure 3.4 shows the position of the Pareto particles (denoted as PP in the plot titles) in increasing magnitude of GP error. VMax 20 - PP 1 has the lowest GP error, VMax 100 – PP 8 has the highest GP error and VMax 40 – PP 3 is in between. The GP and GSGP outputs for all Pareto particles are almost a linear line through the middle of the data points, just like with the previous experiments. VMax 40 and VMax 100 also show one or two extreme values that gets predicted accurately by GP but not at all by GSGP. The standard deviation of the data seems to get higher over the Pareto particles. VMAX 20 – PP 1 is clearly less spread out on the Y axis (independent variable) than VMax 40 – PP 3, which in turn is clearly less spread out than VMax 100 – PP 8. This indicates that there might be a correlation between GP error and the spread of the data.

Figure 3.5 shows data for all Pareto particles of the different median non-dominated sets that were produced during this experimental phase. Every point in the plot represents a Pareto particle's standard deviation of the dependent variable (y axis) against the Pareto particle's GP Error (y axis). The dotted line is the regression line for this data. It is evident that GP error and standard deviation are very highly correlated. The Pearson's R value is 0.995, indicating a nearly perfect positive correlation between GP Error and the standard deviation of the dependent variable.

The increase in delta error is analyzed in a similar way as the increase in GP error was. Two of the lowest delta errors were picked from all non-dominated sets and compared to the two Pareto particles with the highest delta errors from all non-dominated

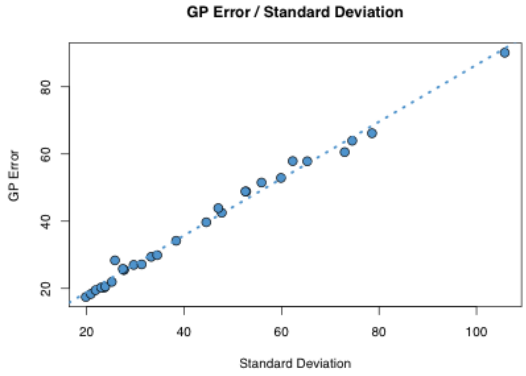


**Figure 3.5:** GP errors from all pareto particles from Figure 3 expressed against their standard deviation of the dependent variable and the regression line for this data.

sets. The positions of these Pareto particles are plotted in Figure 3.6 against their respective GP and GSGP outputs. VMax 40 – PP 1 and VMax 100 – PP1 have low delta errors, while VMax 100 – PP 9 and VMax 100 – PP 10 have high delta errors. A difference between the positions of pareto particles with high and low delta errors is an outlier that is present. VMax 100 – PP 9 and VMax 100 – PP 10 both have an extreme outlier that GP is able to predict quite accurately, while GSGP is not able to predict it at all.

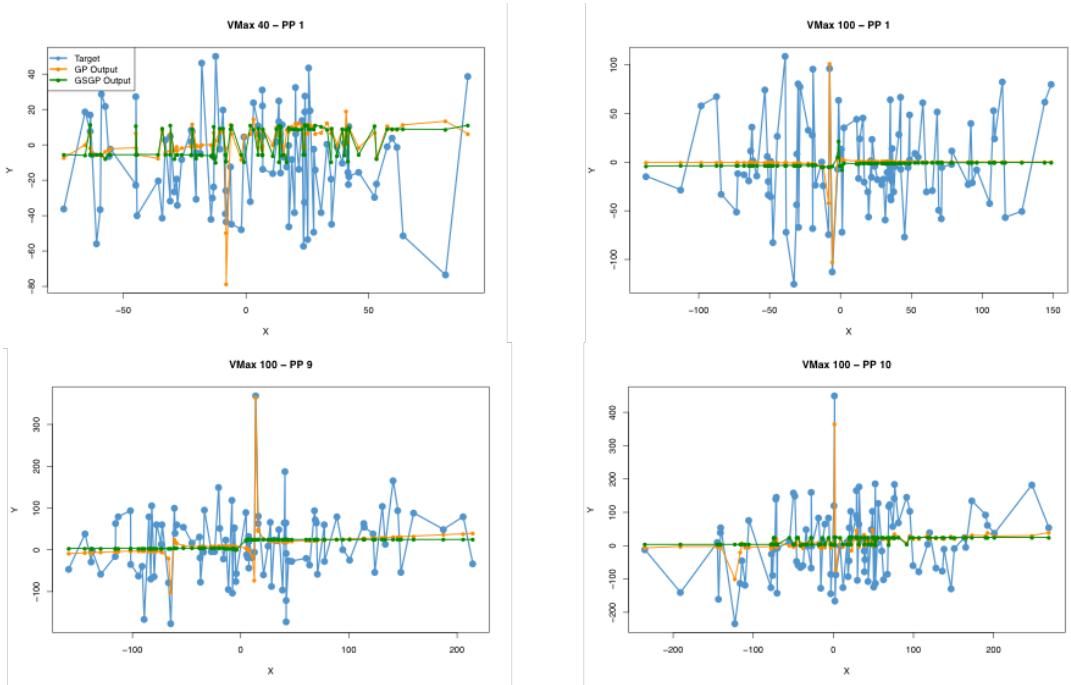


**Figure 3.6:** Positions of pareto particles from different non dominant sets, produced by different VMax settings. 'PP' in the titles stands for pareto particle. VMax 40 – PP 1 has a delta error of 0, VMax 100 – PP1 has a delta error of 2.5, VMAX 100 – PP 9 has a delta error of 12 and VMax 100 – PP 10 has a delta error of 13.

### 3.2.3 Discussion

In order to get the GP error down, the algorithm is shrinking the spread of the data instead of finding a data set that better fits a model produced by GP. Thus, the algorithm is doing what it's programmed to do (minimizing GP error) but not doing it in a way that was expected, nor in a way that satisfies our objectives. A smaller GP error at this stage simply means a smaller spread of the data as proven in Figure 3.5, not a better fitted model. Besides that, the ODF also seems to be increasing the delta error by producing an outlier. This outlier is accurately predicted by GP but not by GSGP, hence the increase in delta error. The further the outlier is removed from the rest of the data, the further the increase in delta error. After seeing these results MOPSO was run more times and this behavior of increasing the delta error was confirmed. It is very likely that this result is simply because of an overfitted model and therefore not behavior that we want to see in the ODF. Even if it wasn't due to overfitting, a model that predicts only one value accurately can't be considered a good model.

To summarize, the algorithm is successfully able to both lower the GP error and to increase the delta error, but not doing so in the way that is desired. The "shortcuts" that are being taken by the ODF probably prevents the algorithm from actually searching for datasets in which it is able to better fit a model. Instead, the ODF is searching for data that better fits a linear model through the center of the data. Firstly, an adjustment has to be made to the algorithm so that the spread of the data is not correlated anymore with the GP error. This will force the ODF to look for other ways of reducing the GP error, hopefully by producing data sets where GP is able to fit a better model. Secondly, the algorithm needs to be prevented from producing outliers to boost the performance of GP compared to GSGP, which is not able to predict these outliers but GP is. The same goes here: if this behavior is restricted, hopefully the ODF will find more suitable ways of improving the datasets.

## 3.3 Phase III

Phase 3 focuses on addressing the main issues that were discovered in the previous research phase. The goal of this phase is to resolve these issues. Alterations to the ODF are proposed to prevent the ODF from shrinking the spread in order to lower the GP error and to prevent the ODF from producing outliers. Once these issues are resolved, experiments are run to see if the updated ODF is able to evolve the datasets in a way such that the datasets actually become more favorable for GP.

### 3.3.1 Experimental Setup

To address the problem of the ODF shrinking the standard deviation of the data in order to shrink the GP error, an alteration to the fitness function is implemented. The RMSE from Equation 3.2 is divided by the standard deviation of the independent variable, which we will call Normalized Root Mean Squared Error (NRMSE). From now on, when we refer to GP error, GSGP error or delta error we are always referring to the NRMSE value. The terms GP/GSGP Error and NRMSE will be used interchangeably. A NRMSE of 0 means the model predicts all independent variables perfectly, a NRMSE of 1 means the model is simply a linear line through the center of the data points as seen in our experiments, thus solving the problem of the algorithm seeing a linear line through the center as a good solution. 3.7 plots the same Pareto particles from Figure 3.3, but this time using the new NRMSE value instead of the RMSE value. It proofs that the correlation between the GP error and standard deviation of the earlier tested data is now non-existent after applying the normalization step. An additional big benefit is that all errors are now range independent. This allows for comparing solutions with different data, e.g. a dataset that has its data points within the range of [0:1] with a NRMSE of 0.5 performs equally as good as a dataset in the range [-1,000:1,000] with a NRMSE of 0.5.

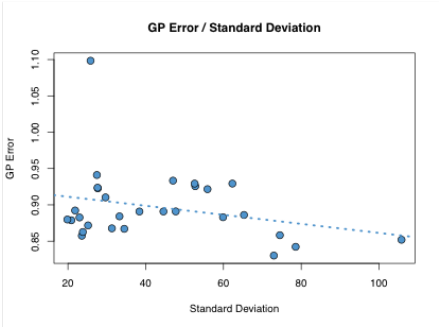


**Figure 3.7:** GP NRMSE from all pareto particles from Figure 3 expressed against their standard deviation of the dependent variable.

An additional alteration is made to the algorithm to address the "shortcut" of outlier generation in order to improve GP error, as discussed in Phase II. Before a particle is evaluated, their positions go through an outlier removal phase. The outliers that need to be removed are the outliers in the independent variable. An outlier is considered an outlier if it is below $Q1 - (1.5 \times IQR)$ or above $Q3 + (1.5 \times IQR)$, where $Q1$ is the first quartile of the particles set of independent variables, $Q3$ is the third quartile of the same set and $IQR$ is the interquartile range. If an outlier is found it is replaced by a random value that lies between the current $Q1$ and $Q3$. The ODF is run with the same settings as before but now only limiting the runs to VMax values of 20 and 100. PSO is first run with only the new NRMSE fitness measure turned on to see the effects of this by itself. After this, outlier processing is turned on as well to analyze the changes in outcome this new feature has by itself.

### 3.3.2 Results

The results are obtained in the same way as with the previous experiment/ MOPSO is run 10 times for each setting and the run that appears to have average results is hand

selected.

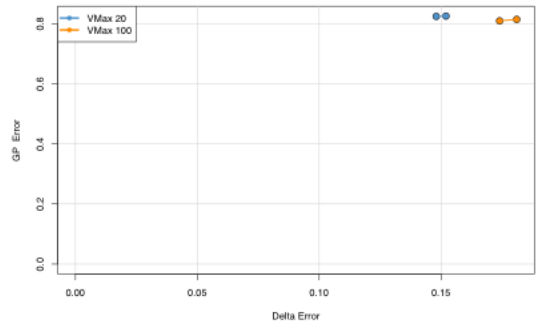


**Figure 3.8:** Positions of non-dominated particles in the objective space for the median runs with different VMax settings. The results were obtained with NRMSE turned on and outlier processing turned off.

To first see the effects of the new NRMSE fitness measures, Figure 3.8 presents the local Pareto fronts without the proposed outlier processing mechanism turned on. Where the earlier used RMSE function produced around 10 Pareto particles per non dominated set $S$, the new NRMSE function produces not more than two Pareto particles in a non-dominated set. This number was higher for some other runs but still a lot less than before. This implies that the algorithm has a harder time finding solutions now it is restricted from taking "shortcuts" in finding better solutions. The fact that the solutions are not able to go below a GP error of 0.8 strengthens this assumption. Next to the GP errors being similar for different VMax settings there also seems to be no significant difference in delta error.

The GP error values of 0.8 translate into the positions of the pareto particles as seen in Figure 3.9. Both models show an almost linear line through the center of the data, which is what is expected with a high GP error of 0.8, thus the new measure is successful in penalizing the way it was previously improving the GP error. The

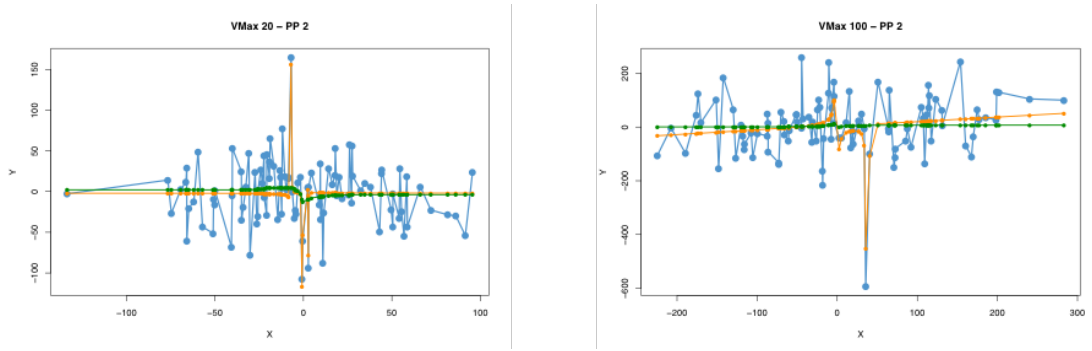


**Figure 3.9:** Positions of two pareto particles produced by the ODF with VMAX 20 and VMax 100 respectively and the new NRMSE error function.

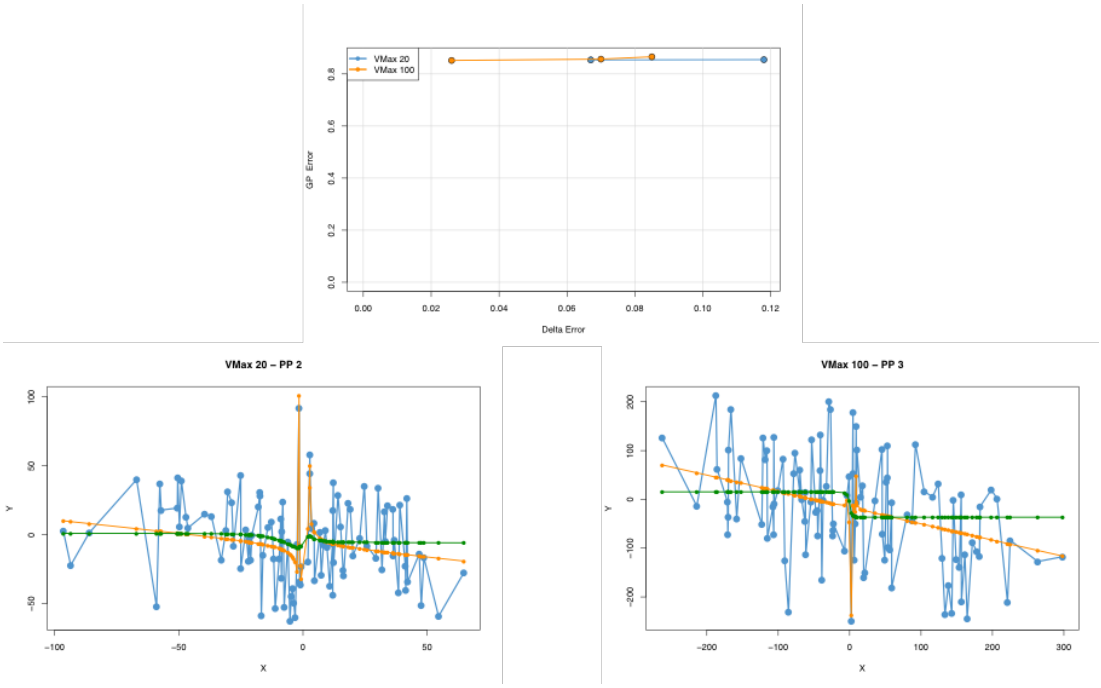


**Figure 3.10:** Local Pareto fronts (top plot) and the positions of selected particles (bottom plots) they contain. MOSPO was run with both NRMSE and outlier processing activated.

Pareto fronts in Figure 3.10 show the local Pareto fronts produced by the ODF with now also outlier processing turned on. The delta errors of these Pareto particles are noticeably lower compared to the Pareto fronts produced without outlier processing. This indicates that the ODF now has a harder time increasing the delta error since it is limited in producing outliers that are only predictable by GP. The positions of the two Pareto particles with the highest delta error confirm this assumption. VMax 20 – PP2 still has a somewhat big value but it is not nearly as extreme as the outliers produced with outlier processing turned off.

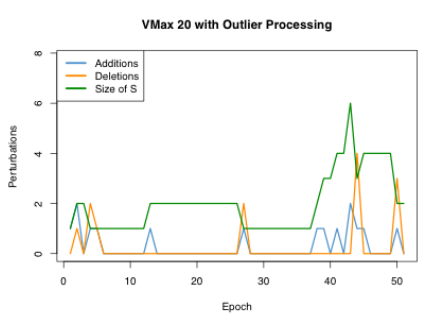


**Figure 3.11:** Perturbations of the non-dominated set.

The ODF is now not showing unwanted behavior anymore as in that it is taking "shortcuts" to falsely indicate that a solution is getting better. The solutions are, however, far from desirable since they do not represent accurate models in the slightest. It is highly probably that the solution to this problem is not to be found in a different VMax value or increasing the number of epochs as can be concluded after looking at Figure 3.11. The plot shows the size of S at every epoch and the amount of additions and deletions performed in that epoch. There are not a lot of perturbations

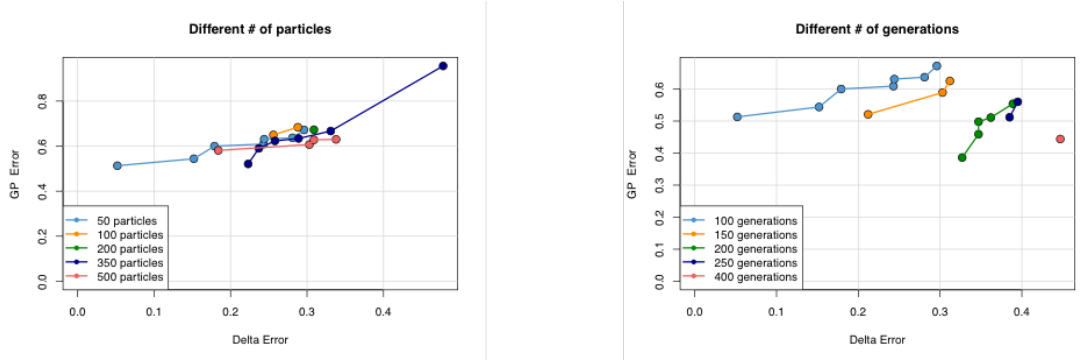


**Figure 3.12:** Comparison of local Pareto fronts produced by different particle settings and G(SG)P generation settings respectively. These results are obtained with the number of instances set to 30.

taking place throughout the entire ODF run and even if the Pareto front would continue evolving like this indefinitely, which is highly improbable, it would take months to produce an acceptable solution. Two other parameters, which have intentionally been kept low so far to save computational cost, that might improve the performance of the ODF are the number of particles and the number of generations for GP and GSGP.

To research the effect of changing the number of particles and generations, the number of instances has been lowered to 30. The reason for this being the very big increase in computational effort with the increase in both parameters (Table 1). VMax has been fixed to 50. Changing the number of particles doesn't result in any real improvement to the produced solutions as can be seen in Figure 3.12. There is a slight difference but nothing that leads to believe that increasing the number of particles has a positive effect on the ability the ODF has on finding a suitable solution.

Changing the number of generations clearly does lead to an overall improvement in the solutions found by the ODF. The local Pareto front moves beneficially with every increase of the number of generations. However, it doesn't seem like improving the generations would lead to the ODF producing desirable solutions since it doesn't seem like the GP error will converge somewhere close to 0 anytime soon. Even if it did, it would not be able to do so within reasonable limits: raising the number of generations increases the execution time of the ODF significantly, as can be seen in Table 3.1.

| | | Generations | | | |
|---|---|---|---|---|---|
| | | **100** | **200** | **300** | **400** |
| **Particles** | **50** | 0.74 | 2.36 | 4.2 | 6.17 |
| | **150** | 2.06 | 6.38 | 14.62 | 18.78 |
| | **300** | 3.42 | 12.96 | 30.25 | 37.33 |

**Table 3.1:** Execution times of the ODF in hours for different settings of GP/GSGP and number of particles.

### 3.3.3 Discussion

The implementation of the new NRMSE measure effectively solved the problem of the ODF reducing the standard deviation of the dataset in order to reduce the error of both GP and GSGP. However, the ODF doesn't come anywhere near converging to a satisfiable solution. This indicates that in this state, it's not possible for the ODF to find other ways of improving the solutions, i.e. by finding data sets that GP is actually better able to fit its models on. The same goes for the new outlier processing step. It does solve the problem it's designed to address, but it makes it even harder for the ODF to evolve the solutions. It also doesn't seem like changing VMax can have that much of an impact to get the ODF to produce suitable solutions. Instead, we assume that the solution lies in the number of GP/GSGP generations and the number of particles.

Both parameters have intentionally be kept low so far to limit computational cost. Also, it was argued that it might produce even better solutions when these parameters are kept low. The reasoning behind this was that if the ODF is able to find solutions at only 100 generations that have very low GP errors and very high delta errors, it would surely imply that a dataset is very easily solvable by GP but very difficult for GSGP. Since this assumption has not been able to be proven so far, different particle and generation settings were used. Increasing the number of particles did not noticeably improve the solutions, but improving the number of generations did. That the increase in the number of generations would lead to a decrease in GP error was expected since GP/GSGP has more time to fit its model. However, it also leads to a noticeable increase in delta error, implying that the it does more than just allowing GP/GSGP to evolve for longer. That the increase in particles did not lead to better results was not expected considering that the search phase is infinite for every instance. With a search space that big, one would expect that more particles would surely lead to better performance.

Either way, the ODF is not coming anywhere near converging to a solution that allows for analyzing GSGP hardness effectively. It also doesn't seem that any of the tried parameters will influence the performance of the ODF enough to achieving such solutions. The optimization problem proves to be too complex at this complexity level, so it is believed to be beneficial to take a step back by starting from a low complexity level and build up the complexity from there. This can be done by lowering the number

of instances even further, evolving the ODF successfully on that number of instances, and then raise the number of instances to do the same on the new complexity level. Using single objective optimization instead of multi objective optimization lowers the complexity level as well, plus that it might lead to valuable insights. Knowing what minimizes the GP/GSGP error and what maximizes the delta error separately would be valuable knowledge that allows for targeted parameter setting.

## 3.4 Phase IV

Previous experiments with MOOP have not yet been able to produce solutions that can be used to effectively analyze GSGP hardness: both the GP error and the delta error are not nearing convincing values. Therefore it is decided to focus on one of the measures first instead. Specifically we would like the GP error to be as low as possible, but so far it has not been anywhere near to closing in on zero, nor did it look like it would anytime soon. If we would know what settings lead to a low GP error, that knowledge could be integrated in MOPSO which would in turn produce better results as well. The same can be said about knowing what settings lead to a high delta error, independently from the GP/GSGP error.

Focusing on these objectives separately allows for a more scoped research to the optimization problem at hand. Thus, this phase does not focus on multi-objective optimization, but once again on single objective optimization. Unlike in the first phase however, we are not only maximizing the delta error but also minimizing the GP or GSGP error, depending on the goals of the experiment. The goal of this phase is to gain insights towards the effect of different parameter settings on GP/GSGP error, which in turn can then be applied to the multi objective version of the ODF. This is done by starting from a low complexity problem (low instances & single objective) and building up in complexity from there.

### 3.4.1 Experimental Setup

The algorithm is altered so that it only produces datasets that minimizes the GP error or maximizes the delta error, reverting it back to a single objective PSO. Firstly the focus will be on producing datasets that minimize the GP error. After this research has been concluded, attempts will be made to produce datasets that minimize the GSGP error. The starting point of this research will be a dataset with only 10 instances. If good results are obtained for this number of instances, the number of instances will be increased. This allows us to build our way up to increasing levels of complexity while reducing computational complexity. GP/GSGP generations are raised to 700 generations. This decreases the chance of a too low number of generations being the reason why the ODF is not able to converge to a solution. The number of epochs are set to 15. All other settings remain the same as with the previous experimental phase.

Note that in this phase all settings are subject to change in an attempt to find optimal settings. These changes will be presented and discussed as they come. PSO is run a total of 10 times, after which the median run is used to analyze the results.

A big alteration to the ODF is that the independent variable(s) will now be fixed upon initialization. Only the dependent variables will be evolved. This reduces the complexity of the optimization problem because it leaves the PSO with only having to evolve the solutions on one dimension. Since the target values are not frozen, it remains possible to evolve a dataset so that it fits a possible successful target function. This phase also closes in on reaching solutions with low errors, which raises the necessity of addressing the issue of the heuristic nature of both GP and GSGP. As said previously, currently both algorithms run only once which means there is a possibility of biased results, e.g. by bad initialization of GSGP but good initialization of GP. A validator is therefore implemented as a final step of the algorithm. This means that after the PSO is done executing, the best found solution will be run again for 50 times on 1000 generations. Because of this implementation we will from now on be referring to and discussing these different results as original results and validated results.

This stage is a more empirical based research through heavy trial and error than it is structured. This leads to actually multiple different experiments being performed in this phase alone. This results in a lot of information. Therefore, several observations are pointed out to collect and organize the most important information from all these results.

### 3.4.2 Results

The experimental results will be distinguished between original and validated results when necessary. All presented results are the solutions with the median fitness value over 10 PSO runs. Datasets are evolved to favor GP and GSGP interchangeably depending on the goals of the experiment. Table 3.2 presents the first results of optimizing both GP and GSGP separately with 700 generations and only 10 instances. The GP error gets really low successfully on both the original run and validator run.

| | **Original** | | **Validated** | |
|---|---|---|---|---|
| | **NRMSE** | **Delta Error** | **NRMSE** | **Delta Error** |
| **GP** | 0.029 | 0.969 | 0.018 | 0.867 |
| **GSGP** | 0.133 | 0.014 | 0.028 | -0.015 |

**Table 3.2:** Original and validated results for optimizing datasets for GP and GSGP respectively. NRMSE was used as the objective function and the dataset consists of 10 instances.

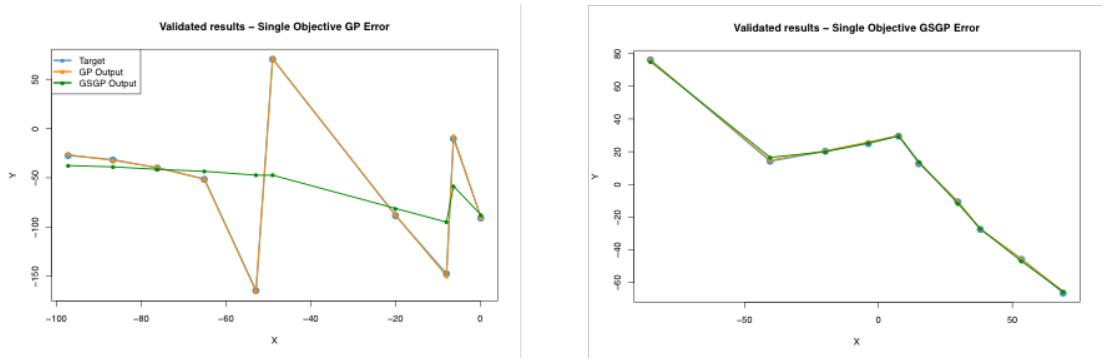


**Figure 3.13:** Comparison of local Pareto fronts produced by different particle settings and G(SG)P generation settings respectively. These results are obtained with the number of instances set to 30.

What is a very unexpected result however is the value of the delta error when optimizing GP. It would be expected that both GP and GSGP are able to easily solve a problem with only 10 instances, no matter what the settings are. Remember that we are only trying to evolve datasets that minimize the error for GP and GSGP respectively; the delta error is not an objective function. Despite this, the delta error produced when optimizing GP is very big. Very unexpectedly, GSGP has a harder time converging to a solution than GP. Even when the GSGP error is optimized, it still gets outperformed slightly by GP after validation. For the first time and without intention, the solution produced by optimizing GP error is starting to resemble a dataset that satisfies our goals (Figure 3.13).

The results of the favorable delta error for GP raises more research questions towards this subject. It is unexpected and unclear why GP is outperforming GSGP on just 10 instances and 2 dimensions. The ODF was run again to optimize datasets that for GSGP on delta error as fitness function. In other words, the ODF will try to produce a data set in which GSGP is able to outperform GP by the biggest difference possible, regardless of how high the GSGP error is. For comparison purposes, the ODF was also run to optimize delta error in favor of GP. The evolution of GBest for both settings is plotted in Figure 3.14).

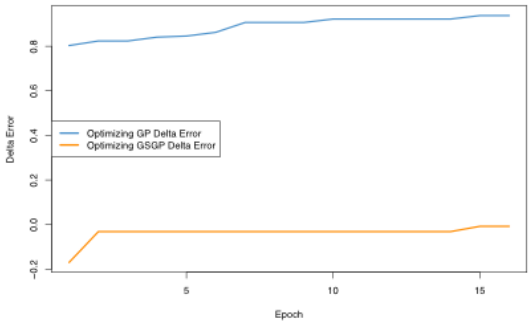


**Figure 3.14:** Validated results of the Evolution of GBest when optimizing delta errors in favor of GP and GSGP respectively.

The ODF is not able to find a solution where GSGP is outperforming GP, even when solely focusing on the delta error. It is able to converge to a delta error that is close to 0, but has not been able to find a position where it actually outperforms GP. GSGP is hardly able to evolve at all as can be seen from the graph. However, if we're optimizing GP on delta error, it is clear that it is very easy for the ODF to find better solutions from the start. Optimizing datasets in favor of GP also shows more evolution than optimizing for GSGP. The results were so remarkable that multiple tests have been run with different number of epochs and different number of VMax values, all converging to similar solutions.

GP is able to converge to a good solution almost instantly, while GSGP is not. This fact, together with the analyzed results, imply that the reason for this behavior is most likely not to be found in PSO settings but in the genotype of the solutions, i.e. the composition of the dataset. As a first way of testing this assumption, the ODF was run with a different number of variables while trying to optimize delta error in favor of GSGP. In other words, we are again looking for a data set at which GSGP outperforms GP, no matter how high or low the GSGP error is. The following number of variables are tested: 2, 3, 5, 7 and 10. Remember that there is always 1 dependent variable, making the remaining $n-1$ dimensions independent variables. The median results are presented in Table 3.3.

| # Variables | NRMSE | Delta Error |
|---|---|---|
| 2 | 0.749 | -0.016 |
| 3 | 0.036 | 0.437 |
| 5 | 0.031 | 0.663 |
| 10 | 0.005 | 1.265 |

**Table 3.3:** Original results for optimizing GSGP with delta error as the objective function, using different number of variables.

The results from the original runs are very drastic. Where the ODF is not able to find a good solution when the dataset has 2 variables, the delta error seems to take off for all other datasets which use a higher number of variables. The results are remarkable and imply that the number of variables is an important factor in GSGP performance. Specifically, GSGP doesn't seem to be able to converge when using 2 variables. Even more so, the performance of GSGP seems to increase with every increase of the number of varuavkes. The good results of more variables are not due to a better initialization. Figure 3.15 proves that the opposite is true: the higher the number of variables, the worse the initialization is. The good results are because the learning curve becomes steeper as the number of variables are raised. This is very counterintuitive because more dimensions intuitively mean a more complex optimization. The above results lead to the first observation:

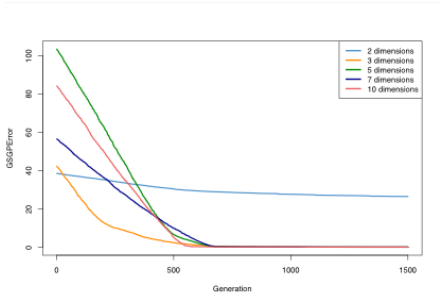


**Figure 3.15:** Validated evolution of GSGP error of GBest for different number of variables.

**Observation 1.**
*The number of variables in a dataset is an important ranking factor towards GP or GSGP favorability.*

To add do Observation 1, experiments have showed thus far that datasets with 2 variables are favorable for GP and more datasets with more variables are favorable for GSGP. However, this holds only under the current experimental context and is not a universal fact.

The initial goal of this research was to find GP and GSGP favorable datasets containing 2 variables. This would allow us to plot the solutions on 2 dimensional space so that we might be able to derive patterns or functions from the data. However, Observation 1 leads us to believe that this goal comes with an impossibility: GP converges very well on datasets with 2 variables but GSGP does not, Before it is possible to make any definitive conclusions about this, more tests have to be run. The ODF has not proven to work yet: GP did converge successfully, but it already did so on initialization, meaning that it's because of the simplicity of the data set that consists of only 10 instances. We want the ODF to actually evolve to draw further conclusions and see whether Observation 1 still holds. To proof this, the GP/GSGP error is used again as objective function. This time however, the number of instances are raised.

The ODF has been run on 20 instances while optimizing datasets on GP error and GSGP error separately. The goal is to see if we can optimize GP on 20 instances as well. Also, we want to verify if GSGP does not improve performance for 2 dimensions if the number of instances is raised. It is assumed that both errors will suffer from

raising the number of instances. Every instance that is added to the genotype means that every particle has an infinite search space to fly through. Thus, the ODF has been run on different number of particles: 20, 50, 75, 100, 200, 300, 400 and 500. Although previous attempts to raise the number of particles in Phase II were not successful, it is believed that freezing the independent variables might change this.

| | Original Results | | Validated Results | |
|---|---|---|---|---|
| **# Particles** | **Opt. for GP** | **Opt. for GSGP** | **Opt. for GP** | **Opt. for GSGP** |
| 20 | 0.422 | 0.455 | 0.269 | 0.427 |
| 50 | 0.318 | 0.275 | 0.235 | 0.252 |
| 75 | 0.328 | 0.287 | 0.316 | 0.264 |
| 100 | 0.268 | 0.275 | 0.188 | 0.257 |
| 200 | 0.248 | 0.228 | 0.189 | 0.211 |
| 300 | 0.23 | 0.193 | 0.215 | 0.18 |
| 400 | 0.214 | 0.155 | 0.199 | 0.109 |
| 500 | 0.161 | 0.147 | 0.112 | 0.126 |

**Table 3.4:** Original and validated NRMSE results for running the ODF with different number of particles while using NRMSE as objective function. The results are obtained from optimizing datasets for both GP and GSGP separately.

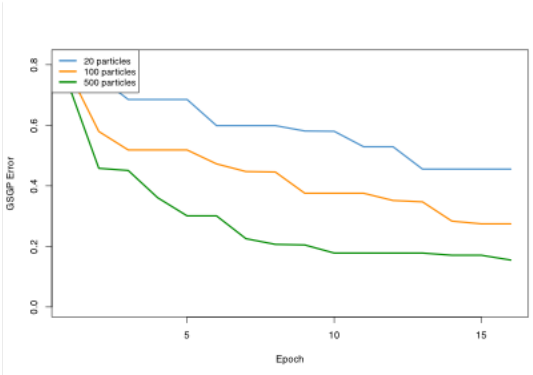


**Figure 3.16:** Evolution of GBest when optimizing for GSGP on 20, 100 and 500 particles.

The results in Table 3.4 and show that the number of particles do matter for the ODF's ability to converge. A decrease in error is present when optimizing for both GP and GSGP if the number of particles is raised. This indicates that raising the number of particles is beneficial towards converging to a better solution and also that the ODF is now actually evolving. The latter is verified by the GBest evolution of 20, 100 and 500 particles when optimizing for GSGP, plotted in Figure 3.16. For the first time, the ODF has shown that it is able to evolve solutions towards a solution. Interestingly, this time GSGP is performing better than GP, as in that it is able to converge to a solution that more accurately predicts the evolved dataset. However, it doesn't say anything about

favorability towards either algorithm. To get more insights into dataset favorability towards GP or GSGP, the ODF was run using 500 particles and 300 generations, but this time using the delta error as the objective function. In other words, the ODF is trying to find datasets on which either GP or GSGP outperforms the other algorithm, not considering the value of the NRMSE. The results are shown in Table 3.5.

| | Original Results | | Validated Results | |
|---|---|---|---|---|
| | **NRMSE** | **Delta Error** | **NRMSE** | **Delta Error** |
| **GP** | 0.959 | 42.408 | 0.51 | 6.935 |
| **GSGP** | 0.719 | 0.086 | 0.67 | -0.364 |

**Table 3.5:** Original and validated results for optimizing datasets for GP and GSGP respectively. Delta error was used as the objective function and the dataset consists of 20 instances.

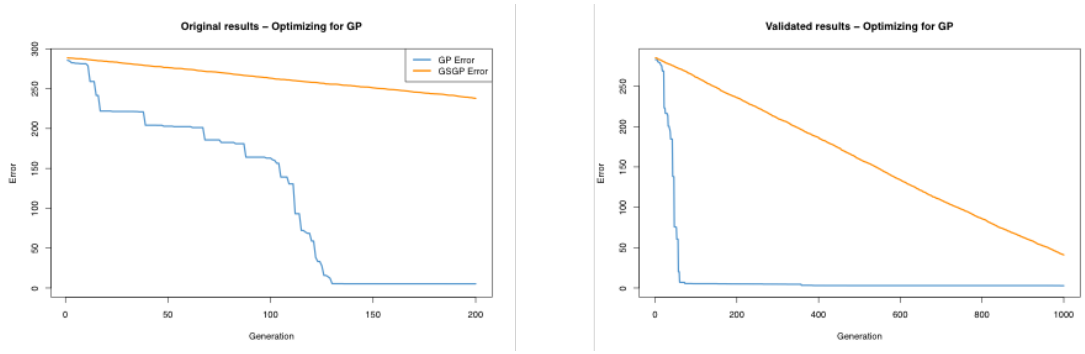


**Figure 3.17:** Original and validated evolutions of GP/GSGP errors for GBest solution when improving for GP as presented in Table 3.5.

GSGP still has a hard time converging to a solution in which it is able to outperform GP on 20 instances, just like it had on 10 instances. The produced original solution beats GP only by a small margin but it is outperformed after validation. The ODF thus falsely identifies a solution as better during the original run. However, even the falsely identified "good" run can't really be considered "good" because the delta error is very low. GP doesn't get outperformed by a convincing amount even if the value were true. Thus, findings under Observation 1 has been confirmed under a higher number of instances as well; GP easily outperforms GSGP while using 2 variables, while GSGP can't outperform GP on 2 variables.

The difference in delta error between original and validated delta error when optimizing for GSGP is not alarming since the produced original solution doesn't create a very big false image of the actual results. The same can not be said when optimizing for GP. The delta error was originally 42.408, after which it got down to 6.935. The validated value is still very big, but the difference with the original value is even bigger. Figure 3.17 explains why this is happening: GSGP just takes a longer time to converge

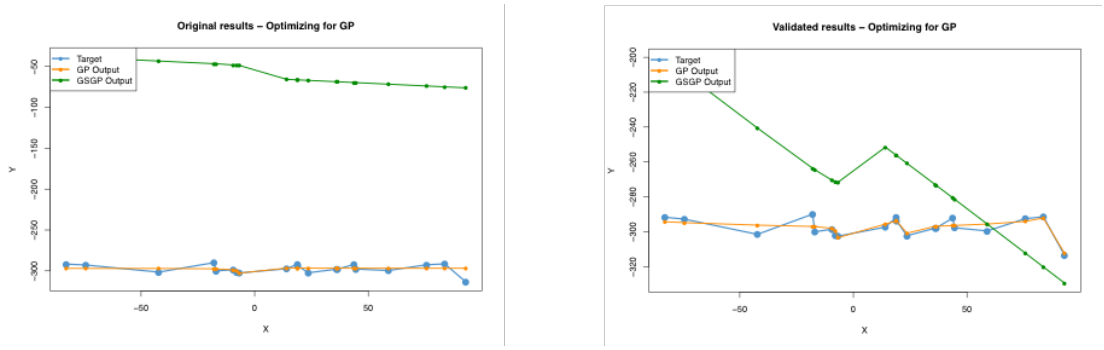


**Figure 3.18:** Original and validated positions of GBest after optimizing for GP as presented in Table 3.5 with the corresponding GP and GSGP outputs.

to a good solution. In fact, GSGP keeps on improving constantly and might have done so for longer if the validator was run for more generations.

However, raising the number of generations is not deemed necessary after seeing these results. Firstly, the ODF is not completely wrong in that GP is heavily outperforming GSGP. GP converges in around 100 generations, while GSGP is not even at that point after 1,000 generations. In a way, this also means that GP is outperforming GSGP since this dataset is favorable for GP in terms of performance. In fact, it might give more valuable insights towards GSGP hardness. This makes even more sense after looking at the position of GBest and its GP/GSGP outputs in Figure 3.18. The original GSGP outputs are not even close to the target values and in validation it's simply closing in on the target values. *Why does this dataset lead to a GSGP initialization that is so far off?* This could be an important question in analyzing GSGP hardness, as the validation has shown that this was not due to a bad initialization: the validated results started at the same error as the original results. Secondly, in the multi objective version of the ODF, this would only be one of the solutions in the non-dominated set. For these reasons, this behavior is not seen as undesirable and no steps are taken to prevent it from happening.

We can now say that Observation 1 still holds on 20 instances when the ODF actually evolves. This raises more questions about the dataset composition, which was originally out of scope; the assumption was that the ODF would be able to evolve solutions without altering the genotype. However, as this research has shown the genotype is an unneglectable ranking factor. So far the number of instances has been changed, as well as the number of dimensions. A third big factor that determines the composition of the genotype is the range of the dataset, which has so far always been in the interval [-100:100]. As said before, it is impracticable at this stage to thoroughly test different combinations of genotype settings. However, as with the number of variables, a small experiment was performed to see if it leads to any noticeable changes in the outcome of the ODF. The ODF is run with a range of [-500:0], [-1:0], [-1:1], [0:1], and [0:500] and [-250:250].

These ranges test the effect of either negative or positive ranges and the effect of smaller ranges, as well as larger ranges. The ranges have been tested when trying to optimize datasets for both GP and GSGP on both objective functions separately. In other words, the ODF is run on different ranges while optimizing four different combinations: optimizing for GP with NRMSE as objective function, optimizing for GP with delta error as objective function, optimizing for GSGP with NRMSE as objective function and optimizing for GSGP with delta error as objective function. The original and validated results are shown in Table 3.6 and Table 3.7 respectively.

| | **Minimizing NRMSE** | | **Maximizing Delta Error** | |
|---|---|---|---|---|
| **Range** | **Opt. for GP** | **Opt. for GSGP** | **Opt. for GP** | **Opt. for GSGP** |
| [-500:0] | 0.226 | 0.298 | 10.377 | 0.021 |
| [-1:0] | 0.305 | 0.158 | 0.365 | 0.634 |
| [-1:1] | 0.297 | 0.107 | 0.414 | 0.652 |
| [0:1] | 0.38 | 0.149 | 0.401 | 0.344 |
| [0:500] | 0.21 | 0.373 | 20.632 | 0.023 |
| [-250:250] | 0.408 | 0.238 | 23.511 | 0.063 |

**Table 3.6:** Original results after running the ODF with different ranges for different optimizable algorithms and different objective functions: Optimizing for GP on NRMSE, optimizing for GP on delta error, optimizing for GSGP on NRMSE and optimizing for GSGP on delta error.

| | **Minimizing NRMSE** | | **Maximizing Delta Error** | |
|---|---|---|---|---|
| **Range** | **Opt. for GP** | **Opt. for GSGP** | **Opt. for GP** | **Opt. for GSGP** |
| [-500:0] | 0.16 | 0.279 | 12.858 | -0.479 |
| [-1:0] | 0.183 | 0.133 | 0.072 | 0.163 |
| [-1:1] | 0.211 | 0.063 | 0.23 | 0.24 |
| [0:1] | 0.247 | 0.102 | 0.04 | 0.321 |
| [0:500] | 0.165 | 0.352 | 13.806 | -0.44 |
| [-250:250] | 0.312 | 0.215 | 8.649 | -0.381 |

**Table 3.7:** Validated results after running the ODF with different ranges for different optimizable algorithms and different objective function: Optimizing for GP on NRMSE, optimizing for GP on delta error, optimizing for GSGP on NRMSE and optimizing for GSGP on delta error.

The ODF produces opposite results depending on which algorithm the datasets are being optimized for. Using bigger ranges converges to better solutions when optimizing for GP for both NRMSE and delta error as objective function, while the opposite is true for GSGP. GSGP converges better on smaller ranges. However, GP performs worse on NRMSE if the range crosses 0, as in [-250:250]. GP is still able to converge at small ranges, but with noticeably less convincing values than on higher ranges. The

validation shows that the results for GP on small ranges are neglectable when using delta error as a fitness function. GSGP is in turn not able to converge on higher ranges when using delta error as objective function. GSGP's overall best solutions are better than GP's best solution when looking solely at the results with NRMSE as objective function value, but GP's overall best solutions are better than GSGP's best solutions when looking solely at the results with the delta error as objective function.

This is just a very small experiment within the total scope of what can be done with range; the ranges could be set to different minimum and maximum value, but the distribution could also be altered. Currently, all independent variables are frozen at a uniform distribution. Also, all independent and dependent variables fall within the same range. It could also be possible, and is probable in real life applications, that different variables fall within different ranges. These findings make it so that the problem at hand is, again, even more complex than thought initially. The second observation follows:

**Observation 2.**
*The range of the dataset is an important ranking factor for GP/GSGP favorability.*

To add to Observation 3, it looks like bigger ranges tend to favor GP, while smaller ranges improve the performance of GSGP. Again, this holds under the current experimental context and is by no means a universal fact.

### 3.4.3 Discussion

A large number of particles is needed for the ODF to evolve and converge to a satisfiable solution. Since every instance that is added creates another infinite search space, this observation intuitively makes sense. It is believed that raising the number of instances will require a raise in the number of particles as well. Raising the number of instances requires more computational cost however, leaving this research to be bounded with using low number of instances. This phase did prove that the ODF is scalable in terms of number of instances used, and the goal is to prove the scalability even further in the next research phase by raising the number of instances again.

The experiments that followed led to a couple of key observations. It difficult to optimize datasets in favor of GSGP that consist of 2 variables. This is very unexpected behavior and it is not clear why GSGP is finding it difficult to converge on 2 variables. It turned out to be a lot easier for GSGP to converge on datasets with more than 2 variables. In fact, turning it to 3 variables already led to a huge increase in performance compared to when using only 2 variables. It is also not clear why GSGP's performance improves by that much when the number of variables is increased. Also the range of the dataset proved to be an important factor. A small range tends to favor GSGP, where big ranges favor GP. The ODF is even able to converge to a solution where GSGP

outperforms GP on 2 dimensions when using a small range, although it is not nearly by as convincing margins as seen the other way around. It is also not clear why the range is effecting the performance of GSGP.

What is clear is that these findings indicate a much more complex relation between the dataset composition and GP/GSGP performance than assumed at the start of this research, and requires further research towards it to understand the characteristics of this relationship. Expanding the research to include optimizing the ODF with a higher number of dimensions would increase the scope of this research beyond realistic boundaries on multiple levels. Firstly, there is no time to conduct this kind of research, mostly because of the extreme runtimes but also because of the time limit of this research. Also, the complexity increases significantly if another variable comes into play, raising the combinatorial options even further. Thirdly, it is not anymore possible to plot the results in 2 dimensional space, making it harder to compare the results of a dataset that favors GP versus a dataset that favors GSGP. All this makes it impracticable to focus this research on more than 2 variables. However, a small exception is made. It is not clear to how well GSGP is going to converge on 2 variables. To produce datasets that are favorable for GSGP as well as GP, the ODF will be run on 3 variables as well. This will proof that the ODF is able to produce solutions that are suitable for analyzing GSGP hardness and it will also prove that it is scalable in terms of number of variables.

Even though we feel like we're just scraping the surface, these observations did lead to valuable knowledge that can be applied to a multi-objective version of the ODF. The main goal has always been to design an algorithm that evolves data sets in favor of GP/GSGP, not to explain why GP or GSGP works better. Instead, this research is intended to lay a foundation towards those questions can be addressed. Although this would have been preferable, the optimization problem turns out to be too complex to derive any real conclusions about it at this stage. Still, if we can apply said knowledge to the multi-objective version of the ODF and produce data sets that satisfy our goals, this research will have achieved a successful outcome and lays a foundation for future research towards the subject of GSGP hardness.

## 3.5 Phase V

During the last phase of this research we return to MOOP as discussed in Phase II and Phase III. Phase V focuses on applying the knowledge attained from optimizing single objectives in Phase IV in terms of parameter settings and genotype settings. The goal is to use this knowledge in an attempt to produce multi-objective results that satisfy our goals, as in that they are relatively low in GP/GSGP error and there is a visible difference in terms of delta error, also after validation. Attempts to produce solutions that satisfy these goals were unsuccessful in Phase II and Phase III.

### 3.5.1 Experimental Setup

The algorithm evolves the solution in the same way as it did in Phase III, except for that it now includes the validator step and it freezes the inputs as proposed in Phase IV. As a final addition, this phase will also introduce running GP and GSGP multiple times every time a particle is evaluated to further address the heuristic nature of both algorithms. The validator is considered to not be enough as a standalone solution. Reason for this being that if a bad initialization is made in an early epoch, the ODF will spend the rest of its time searching around a solution that is falsely identified as a good solution, rendering the whole run useless after validation. Both GP and GSGP will be run 5 times, of which the best run is used. This decreases the possibility of a bad initialization of one algorithm biasing the results. Ideally the number of GP/GSGP runs would be higher, but it is set to 5 as a tradeoff between computational effort and accuracy. The validator will then pick out the remaining false runs.

The experiments in this phase build on the experiments in Phase IV, as they use the same settings only then in an multi-objective environment. Phase IV already showed promising results for both GP error and delta error but the ODF then only considered one optimization function. This phase tests what happens if both objective functions are optimized: GP/GSGP error and delta error. The number of instances will raised to 30 for this research phase. Ideally the number of instances would be even higher, but the computational effort and time limit don't allow for testing this. The ODF will run for 20 epochs. As we have seen in Phase IV, GP and GSGP both have different genotype settings in which they perform better or worse. Depending on the algorithm being optimized and the desired goal, the settings will be changed accordingly. There are four main goals for this final research phase:

1. Produce 2-variable datasets that are favorable for GP and GP predicts the dependent variable values accurately.
2. Produce 2-variable datasets that are favorable for GSGP and GSGP predicts the dependent variable values accurately.
3. Produce 3-variable datasets that are favorable for GP and GP predicts the dependent variable values accurately.
4. Produce 3-variable datasets that are favorable for GSGP and GSGP predicts the dependent variable values accurately.

The first goal proves that the ODF is scalable, as in that it is possible to produce solutions on 30 instances as well as on 10 and 20 instances as worked with previously. The second goal aims to prove whether or not it is indeed possible to evolve GSGP favorable datasets with 2 variables. Goals three and four will prove that the algorithm is also scalable in terms of increasing number of variables. Also, it is unclear that goal

2 will not be able to be met by a big margin, that is by producing a dataset in which the delta error is very big in favor of GSGP. Using 3 variables increases the chance of finding such a solution.

### 3.5.2 Results

As in Phase IV, the experimental results will be distinguished between original and validated results when necessary. Considering the multi-objective nature of the algorithm and the extensive running times the algorithm is only run once for each set of settings. Remember that both GP and GSGP are run 5 times for every particle in every epoch and that every member of the non-dominated set is validated after the original run. This means that multiple ODF runs are not a necessity to produce accurate results in terms of GP/GSGP error. The ODF has been run while optimizing for both GP and GSGP with 2 and 3 variables. The dataset range is [0:500] when optimizing for GP and [0:1] when optimizing for GSGP. These ranges have been found to be favorable for GP and GSGP respectively in Phase IV. The Pareto fronts produced by these four different runs are shown in Figure 3.19.

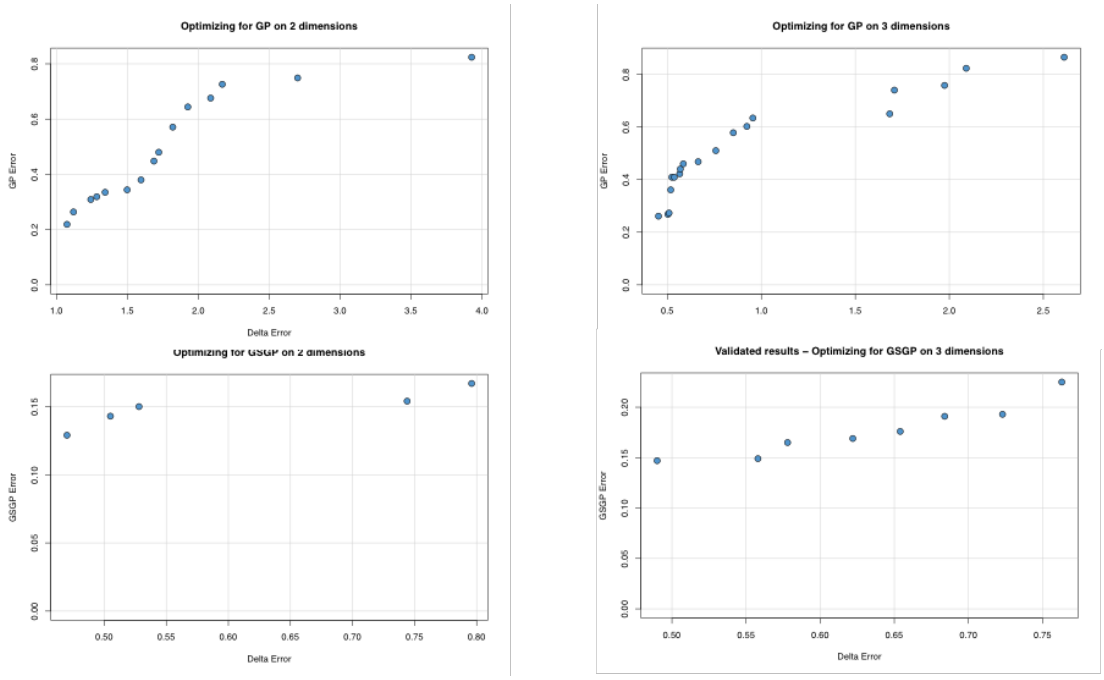


**Figure 3.19:** Pareto fronts produced by the original ODF runs when optimizing for GP and GSGP separately on 2 and 3 dimensions.

Running the ODF when optimizing for GP results in a lot of Pareto particles, both for datasets with 2 variables and 3 variables. As expected after observations made in Phase IV, using 2 variables produces more favorable results for GP than when using 3 variables. The minimum delta error for 2 variables when optimizing for GP is 1, for 3 variables this is 0.5. The GP error is similar with both variable settings. Also, GP

is more able to find solutions with high delta error values at 2 variables. The ODF produces a lot less solutions when optimizing for GSGP. Less solutions however, does not necessarily mean worse. The delta errors when optimizing for GSGP are not as high as when optimizing for GSGP, but they are still large with values of 0.5 and over for both 2 and 3 variables. Also, the GSGP error stays low between 0.15 and 0.22, which is lower than the GP error gets when optimizing for GP. There is potential to produce even better solutions, as can be derived from Figure 20: for all different settings the non-dominated sets do not seem like they have converged at termination, which was set to 20 epochs. However, the runtimes were already very high at only 20 epochs: optimizing for GP finished executing in just under 54 hours, while optimizing for GSGP finished executing in just under 30 hours.

The solutions change a lot in terms of objective function outcomes after validation as seen in Figure 3.20. The Pareto fronts are no longer actual Pareto fronts after validation. This is not an issue as long as the solutions remain that satisfy our goals, as they are visibly better than the opposing algorithm and not too high in NRMSE. This is most definitely the case when optimizing for GP. For datasets with both 2 and 3 variables there are validated solutions with an NRMSE around 0.2 with a large delta error: around 1.0 for 2 variables and around 0.5 for 3 variables. Intuitively, these datasets could be used to analyze GSGP hardness very well. GSGP shows a more significant negative difference between original and validated results. However, there are still solutions that can be used for analysis, especially when using 3 variables. This behavior was expected. For example, a NRMSE of 0.145 with a delta error of 0.16 (2 variables) and a NRMSE of 0.09 with a delta error of 0.35 (3 variables) can intuitively still be qualified as a dataset on which GSGP outperforms GP. Especially the latter looks like a very good solution.

Thus, it can be concluded that the ODF has been able to successfully satisfy all goals set for this Phase: producing datasets on 2 and 3 variables on which GP outperforms GSGP and producing datasets on 2 and 3 variables on which GSGP outperforms GP. Plots of a selection of the produced results using 2 variables can be found in Appendix A.

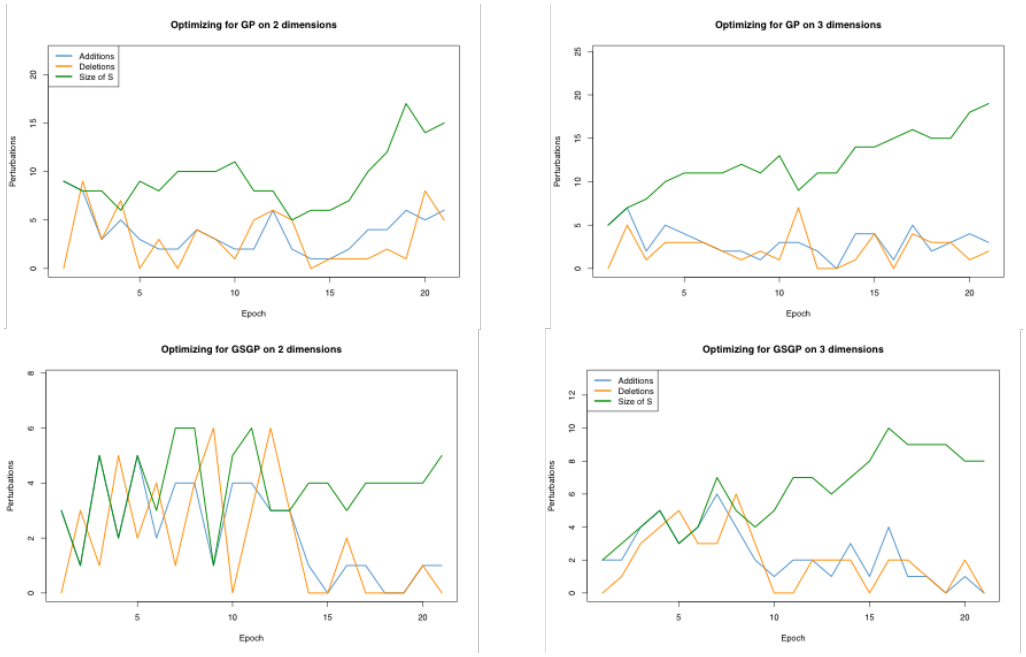


**Figure 3.20:** Perturbations to the non-dominated sets per epoch when optimizing for GP and GSGP seperately using 2 and 3 variables.

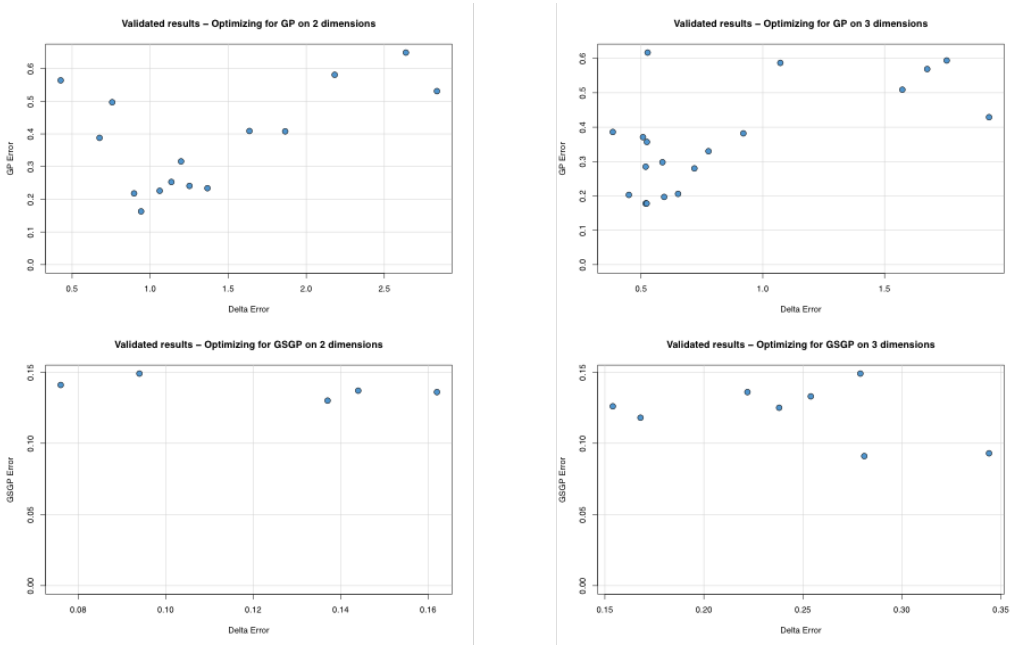


**Figure 3.21:** Validated results for the Pareto fronts when optimizing for GP and GSGP separately using 2 and 3 variables.

### 3.5.3 Discussion

The ODF has shown to be able to produce both datasets that favor GP and datasets that favor GSGP. It has been able to do so on an increasing number of instances and variables. By doing so, the ODF has proven to lay a foundation towards researching GSGP hardness by producing datasets that are either GP hard or GSGP hard, by choice of the user. An unexpected result is that it's easier for the ODF to evolve datasets that are favorable for GP than the other way around. This is also visible in the produced solutions, where the delta errors when optimizing for GP are a lot higher than the delta errors when optimizing for GSGP. In fact, when optimizing for GP the solutions produced are totally non-optimizable for GSGP. When optimizing for GSGP however, the solutions are still optimizable for GP but just a little less accurate than GSGP.

It is not clear why this is occurring. The opposite behavior would have been expected since GSGP outperforms GP in real life applications. However, the datasets that are produced are usually not very similar to real life datasets that are used for symbolic regression. Those datasets typically use more than 1 or 2 independent variables and more than 30 instances. At this stage it can't be said whether this is the cause of this unexpected behavior, but what is certain is that there is an apparent complex relationship between the dataset composition and GP/GSGP performance, as discovered in this research. It will therefore be interesting to see what happens if the ODF is used to produce datasets that resemble more real life datasets, by using more instances and more variables, but this is out of scope for this research.

## 3.6 Final Algorithm

The algorithm has gone through multiple alterations throughout the different phases of this research. Therefore, the final proposed algorithm is shortly reviewed for clarity purposes. The final algorithm is an implementation of the Multi Objective Particle Swarm Optimization algorithm with two objective functions: minimizing GP error and maximizing the difference between GP error and GSGP error, also called delta error (Equation 3.1). When optimizing for GSGP however, the first objective function becomes the GSGP error. Normalized Root Mean Squared Error (NRMSE) is used as error value for GP/GSGP, which is the RMSE (Equation 3.2) divided by the standard deviation of the dependent variable. The pseudocode of the final algorithm and further explanation of some of its concepts are given in Listing 3.1.

**Listing 3.1:** Pseudocode for the final ODF algorithm

```
Randomly initialize swarm [1]
Evaluate[2]  each particle in the swarm
Setup non dominant set S
While termination condition is not met
    For every particle
        Compute velocity according to Equation 2.4
        If velocity > VMax
            Velocity = VMax
        If velocity < VMin
            Velocity = VMin
        Update position according to Equation 2.2
    For every particle
        Process Outliers[3]
        Evaluate Particle
        If current position < PBest
            Current position is set to be new PBest
        If both current position and PBest don't dominate each other
            Randomly choose PBest
    Update non-dominated set S[4]
Terminate when maximum number of epochs is reached
Validate non-dominant set S[5]
```

[1] All particles are initialized with the same independent variable values and they remain frozen throughout the whole run: only dependent variables are evolved.

[2] Both GP and GSGP are run for 5 times. The run with the lowest NRMSE is used to assign fitness values to the two objective functions: GP/GSGP error and delta error.

[3] Data points are defined to be outliers when their value is above $Q3 + (1.5 \times IQR)$ or below $Q1 - (1.5 \times IQR)$, where $Q1$ is the first quartile, $Q3$ is the third quartile and $IQR$ denotes the interquartile range. If an outlier is detected, it is replaced with a value between $Q1$ and $Q3$.

[4] Non-dominated set is based on Pareto dominance: if a new position is dominated by any member ofS, the new position is rejected. If the new position is not dominated by any member of S, the new position is added. All members of S that are dominated by the new position are removed.

[5] All positions that are a member of $S$ at the end of a MOPSO run are run 50 times on a large number of GP/GSGP generations. These values are saved as validated results.

# Conclusion 4

There is much research done towards GP hardness within the context of symbolic regression, but research towards GSGP hardness is non-existent. Researching GSGP hardness has been impossible because of the excellent performance of GSGP on real life applications, which raised the impression that any problem is easy to solve for GSGP. An algorithm that is able to produce datasets that favors either algorithm could question this assumption and form a foundation for further research towards GSGP hardness, discovering more about the relationship between dataset patterns/functions and GP/GSGP problem hardness. Therefore, in this paper we proposed an implementation of the Multi-Objective Particle Swarm Optimization (MOPSO) algorithm that evolves datasets for either GP or GSGP on symbolic regression problems. It does so by simultaneously minimizing the GP error and maximizing the difference between GP and GSGP error in favor of GP. The opposite is true when optimizing datasets in favor of GSGP. Such an algorithm is interesting because it would allow for researching GSGP hardness for the first time. We initially limited the research to generating datasets with 1 independent variable and 1 dependent variable, or 2 dimensions. This would allow for analysis of the datasets by plotting them on 2 dimensional space.

Deriving conclusions from the results was, at this stage, not the main goal of this research. Inevitably however, several discoveries were done while designing and implementing the ODF. Mainly, it was found that the range of the dataset and the number of dimensions (dependent and independent variables) are an important ranking factor for GP/GSGP performance. GP tends to converge to better solutions when using only 2 dimensions, while GSGP performs better with the increase of the number of dimensions. GP prefers big ranges, specifically ranges that stay on one side of 0 , e.g. [-500:0], while GSGP prefers small ranges. Because of these findings we expanded the research to runs on 3 dimensions as well so that more convincing solutions would be produced when optimizing for GSGP and it would prove that the algorithm is scalable along dimensions. Although very computationally extensive, the ODF has showed to be able to produce datasets that are favorable for either GP or GSGP.

The algorithm was able to produce GP and GSGP favorable datasets on both 2 and 3 dimensions while using the settings that favor the corresponding algorithm.

Interestingly, the solutions produced when optimizing for GP were a lot better in terms of favorability towards GP than the solutions when optimizing for GSGP. Considering the outstanding performance of GSGP in real life applications, one would expect the opposite to happen. In fact, the ODF has been able to produce datasets on which GP is able to build an accurate model but GSGP is not able to converge at all on. When optimizing for GSGP however, the solutions do favor GSGP, but GP is still able to build a quite accurate model on the same dataset, just less accurate than GSGP. The relation between dataset composition and GP/GSGP performance proved to be complex and more research is needed towards it to explain this behavior.

In any case, the ODF has been able to produce results that satisfy our goals within the scope that we have set, as that they show a clear favorability towards one algorithm while keeping the error value low. It is now possible to produce datasets that favor either GP or GSGP by the push of a button. There is room left for improvement in accuracy and performance, but the datasets being produced could already be used to research GSGP hardness, making effective research towards this subject possible for the first time.

# Limitations and Recommendations for Future Research

5

The main limitations of this research were due to computational complexity and due to the optimization problem complexity. The computational complexity was the biggest drawback. When talking about execution time of the algorithm, we are talking in hours or even days. This is only with a maximum number of instances of 30. Raising the number of instances would require even more running time. Firstly because more instances take more computing time for MOPSO, GP and GSGP. Secondly because the number of particles will most likely have to be raised to cover the expanded search space. The test were run on a server with 4 Intel Xeon E5620, 2.4GHz processor cores. Each run of the algorithm took up a whole core, where running GP and GSGP take up almost all of the processing power. Making the ODF multi-threaded resulted in the use of a whole core per thread. Future research would ideally focus on more instances in the dataset but to do that within a reasonable time frame, either more computational power is needed or the GP/GSGP algorithms have to evolve more efficiently. In short, more computational power would allow for scaling the algorithm. This would also allow to run the ODF multiple times and run GP/GSGP more often during evaluation, allowing for even more accurate results. It would also allow for the algorithm to be run for more epochs, ensuring the algorithm has converged by the end of its run.

Furthermore, this research provided a way of producing datasets that are either GP hard or GSGP hard. For the first time we are now able to produce datasets out of nothing that favor either GP or GSGP. This means that there is no more lack of datasets on which GSGP finds it more difficult to converge than GP. These datasets can thus be used in future research to analyze GSGP hardness. Research of this sort can uncover some parts of the black boxes of GP and GSGP, possibly unraveling not only which functions or patterns define when GP is better than GSGP (or the other way around), but also on a more foundational level about the characteristic behavior of both GP and GSGP.

As shown in the research however, the relation between datasets and GP/GSGP performance is too complex to be fully covered by the current algorithm. Different dataset combinations were used that proved to have a big impact on GP/GSGP performance, but only a small fraction of possible combinations were tested in this thesis. Different combinations of dataset composition could be more thoroughly tested to reveal more of these characteristics. Given the large number of possible combinations, this is a whole new optimization problem on its own. It could even be that another optimization algorithm is required to adjust the parameters. Also here we revert back to the previous statement: computational cost is a big issue. To optimize such a problem would increase the running time by so much that it is not able to do it within any reasonable time frame. So once again, we'd like to emphasize that the computational complexity is an issue that needs to be addressed to further expand the algorithm and the research towards GSGP hardness.

# Bibliography


[1] J. Koza. "Genetic Programming as a Means for Programming Computers by Natural Selection." In: *Statistics and Computing*. 1994, pp. 87–112.

[2] A. Moraglio, K. Krawiec, and C. Johnson. "Geometric Semantic Genetic Programming." In: *Parallel Problem Solving from Nature - PPSN XII. PPSN 2012. Lecture Notes in Computer Science*. Ed. by C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone. Vol. 7491. Berlin, Heidelberg, 2012.

[3] L. Vanneschi, M. Castelli, L. Manzoni, and S. Silva. "A New Implementation of Geometric Semantic GP and Its Application to Problems in Pharmacokinetics." In: *Genetic Programming. EuroGP 2013. Lecture Notes in Computer Science*. Berlin, Heidelberg, 2013.

[4] L. Vanneschi, S. Silva, M. Castelli, and L. Manzoni. "Geometric Semantic Genetic Programming for Real Life Applications." In: *Genetic Programming Theory and Practice XI. Genetic and Evolutionary Computation*. 2014.

[5] M. Castelli, D. Castaldi, I. Giordani, S. Silva, L. Vanneschi, F. Archetti, and D. Maccagnola. "An Efficient Implementation of Geometric Semantic Genetic Programming for Anticoagulation Level Prediction in Pharmacogenetics." In:

[6] K. Stanislawska, K. Krawiec, and Z. W. Kundzewicz. "Modeling Global Temperature Changes With Genetic Programming." In: *Computers  Mathematics with Applications*. Vol. 64. 2012, pp. 87–112.

[7] T. Smetka., I. Homoliak, and P. Hanácek. "On the Application of Symbolic Regression and Genetic Programming for Cryptanalysis of Symmetric Encryption Algorithm." In: *IEEE International Carnahan Conference on Security Technology (ICCST)*. 2016.

[8] A. Adegboye, M. Kampouridis, and C. Johnson. "Regression Genetic Programming for Estimating Trend End in Foreign Exchange Market." In: *IEEE Symposium Series on Computational Intelligence (SSCI)*. 2017.

[9] M. Korns. "Accuracy in Symbolic Regression." In: *Genetic Programming Theory and Practice IX*. Springer, 2011, pp. 129–151.

[10] J. Martins, L. Oliveira, L. Miranda, F. Casadei, and G. Pappa. "Solving the Exponential Growth of Symbolic Regression Trees in Geometric Semantic Genetic Programming." In: *Proceedings of Genetic and Evolutionary Com- putation Conference*. 2018.

[11] M. Castelli, L. Vanneschi, and S. Silva. "Prediction of high performance concrete strength using Genetic Programming with Geometric Semantic Genetic Operators." In: *Expert Systems with Applications*. Vol. 40. Elsevier, 2013, pp. 6856–6862.

[12] J. Xu, Z. Shen, Q. Ren, X. Xie, and Z. Yang. "Geometric Semantic Genetic Programming Algorithm and Slump Prediction." In: *CoRR* (2017).

[13] M. Castelli, L. Vanneschi, and S. Silva. "Prediction of the Unified Parkinson's Disease Rating Scale Assessment using a Genetic Programming System with Geometric Semantic Genetic Operators." In: *Expert Systems with Applications*. Vol. 41. 2014, pp. 4608–4614.

[14] M. Graff, E. Tellez, H. J. Escalante, and S. Miranda-Jiménez. "Semantic Genetic Programming for Sentiment Analysis." In: *NEO 2015. Studies in Computational Intelligence*. Ed. by O. Schütze, L. Trujillo, P. Legrand, and Y. Maldonado. Vol. 663. Springer, 2017.

[15] I. Gonçalves, S. Silva, and C. Fonseca. "On the Generalization Ability of Geometric Semantic Genetic Programming." In: *Genetic Programming. EuroGP 2015. Lecture Notes in Computer Science*. 2015.

[16] K. Kinnear. "Fitness Landscapes and Difficulty in Genetic Programming." In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. 1994.

[17] J. Daida, H. Li, R. Tang, and A. Hilss. "What Makes a Problem GP-Hard? Validating a Hypothesis of Structural Causes." In: *Genetic and Evolutionary Computation — GECCO 2003. GECCO 2003. Lecture Notes in Computer Science*. Berlin, Germany, 2003.

[18] J. Daida. "Limits to Expression in Genetic Programming: Lattice-Aggregate Modeling." In: *Proceedings of the 2002 Congress on Evolutionary Computation*. 2002.

[19] J. Daida and A. Hilss. "Identifying Structural Mechanisms in Standard Genetic Programming." In: *Genetic and Evolutionary Computation — GECCO 2003. GECCO 2003. Lecture Notes in Computer Science*. Vol. 2724. Berlin, Heidelberg, 2003.

[20] M. Clergue, P. Collard, M. Tomassini, and L. Vanneschi. "Fitness Distance Correlation and Problem Difficulty for Genetic Programming." In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'02*. New York City, USA, 2002.

[21] T. Jones and S. Forest. "Fitness Distance Correlation as a Measure of Problem Difficulty in Genetic Algorithms." In: *Proceedings of the Sixth International Conference on Genetic Algorithm*. 1995.

[22] P. C. L. Vanneschi M. Tomassini and M. Clergue. "A Survey of Problem Difficulty in Genetic Programming." In: *Advances in Artificial Intelligence. AI*IA 2005. Lecture Notes in Computer Science*. Ed. by S. Bandini and S. Manzoni. Vol. 3637. Berlin, Heidelberg: Springer, 2005.

[23] L. Vanneschi, M. Tomassini, P. Collard, and S. Vérel. "Negative Slope Coefficient: A Measure to Characterize Genetic Programming Fitness Landscapes." In: *Genetic Programming. EuroGP 2006. Lecture Notes in Computer Science*. Ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustagson, and A. Ékart. Berlin, Heidelberg, 2006.

[24] S. Vérel, P. Collard, and M. Clergue. "Where are Bottlenecks in nk-fitness Landscapes." In: *CEC 2003: IEEE International Congress on Evolutionary Computation*. Canberra, Australia, Piscataway, NJ, 2003.

[25] Y. Ho and D. Pepyne. "Simple Explanation of the No-Free-Lunch Theorem and Its Implications." In: *Journal of Optimization Theory and Applications*, 115 (2002), pp. 549–570.

[26] J. Kennedy and R. Eberhart. "Particle Swarm Optimization." In: *Proc. of the IEEE international Conference on Neural Networks*. Piscataway, New Jersey, USA, 1995.

[27] P. Justesen. "Multi-Objective Optimization using Evolutionary Algorithms." Doctoral dissertation. University of Aarhus, 2009.

[28] C. Coello. "An Introduction To Multi Objective Particle Swarm Optimizers." In: *Soft Computing in Industrial Applications. Advances in Intelligent and Soft Computing*. Ed. by A. G.-C. ans R. Takahashi, G. Schaefer, and L. Costa. Berlin, Heidelberg, 2011.

[29] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. 2002, WILEY.

[30] J. Durillo, J. García-Nieto, A. Nebro, C. Coello, F. Luna, and E. Alba. "Multi-Objective Particle Swarm Optimizers: An Experimental Comparison." In: *Evolutionary Multi-Criterion Optimization. EMO 2009. Lecture Notes in Computer Science*. Ed. by M. Ehrgott, C. Fonseca, X. Gandibleux, J. Hao, and M. Sevaux. Berlin, Heidelberg, 2009.

[31] V. Kumar and S. Minz. "Multi-Objective Particle Swarm Optimization: An Introduction." In: *Smart Computing Review* 4 (2014), pp. 335–353.

[32] C. A. C. Coello and M. S. Lechuga. "MOPSO: a Proposal for Multiple Objective Particle Swarm Optimization." In: *Proceedings of the 2002 Congress on Evolutionary Computation, CEC '02*. Honolulu, HI, 2002.

# GP/GSGP Favorable Datasets A