NOVA IMS
Information Management School

MAA

Mestrado em Métodos Analiticos Avançados
Master Program in Advanced Analytics

On the Use of Semantic Awareness to Limit Overfitting in Genetic Programming

Paul Joscha Englert

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

ON THE USE OF SEMANTIC AWARENESS TO LIMIT OVERFITTING IN GENETIC PROGRAMMING

by

Paul Englert

Dissertation presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Advanced Analytics

Advisor: Dr. Leonardo Vanneschi

May, 2017

ABSTRACT

Machine learning and statistics provide powerful tools to solving problems of many different shapes. But with the algorithms searching for approximations the problem of overfitting remains present. Genetic Programming describes an algorithmic approach that is likely to produce overfitting solutions. Thus, in order to lessen the risk of overfitting and increasing the generalization ability of genetic programming the use of semantic information is assessed in different ways. A multi-objective system driving the population away from overfitting solutions based on semantic distance is presented alongside alternatives and extensions. The extensions include the use of the semantic signature to increase the amount of information available to the system, as well as the consideration to replace the validation dataset. It is on the one hand concluded that the described approaches and none of the extensions have a positive impact on the generalization ability. But on the other hand it seems that the semantics do contain enough information to appropriately discriminate between overfitting and not overfitting individuals.

KEYWORDS

Machine Learning; Statistics; Computational Intelligence; Genetic Programming; Genetic Algorithm; Evolutionary Algorithm; Optimization Algorithm; Optimization Problem; Overfitting; Semantic Awareness; Semantic Repulser; Multi-Objective System; Semantic Signature;

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

EA	evolutionary algorithm
eSRGP	extended semantic repulser genetic programming
GP	genetic programming
GSGP	geometric semantic genetic programming
ML	machine learning
OP	optimization problem
RMSE	root mean squared error
SRGP	semantic repulser genetic programming

1 INTRODUCTION

This chapter provides an introduction to this thesis by generally explaining the motivation, the underlying problem, the objectives, as well as the structure of this work.

1.1 MOTIVATION

As the field of machine learning (ML) is progressing, it branches out into uncountable concepts and approaches [Mit97, see p. 2f.]. Many of which being black-box[1] algorithms producing incomprehensible solutions, while other creating clear and understandable models. The increasing number of concepts, extensions and adaptations create new possibilities, new applications and new insights. Along with the exploding amount of information and data available, opportunities of unthought nature arise. As Gartner, Inc. has stated the ML field is the leading technology trend in business [Gar16], suggesting that it is mature enough to be productively used. Yet despite its maturity, innovation and research, not only new approaches and improvements are unveiled, but old problems become present again and new problems are detected. [KM14, see p. 4f.]

1.2 PROBLEM DEFINITION

One of the oldest and best known problems of any algorithm searching to solve a problem is the case of *overfitting*. Overfitting describes the situation in which an algorithm retrieves a solution well fit to the dataset it was trained upon, but incapable of keeping the level of accuracy on observations excluded during training, while another solution exists, that could perform better on the unseen observations. Typically datasets containing a lot of noise or few observations are bound to make algorithms overfit, but it might also lie in the nature of the algorithm itself to increase the probability of overfitting with increasing training time. [Mit97, see p. 67]

[1] Black-box algorithms, such as neural networks process information without revealing the inputs exact impact, thus producing models which humans are not able to understand. White-box algorithms, such as linear regressions give clear information about the impact of the input. A grey-box algorithm refers to an algorithm creating solutions that potentially could be understood, but require additional processing in order for the information to be comprehensible or traceable. [Lju01, see p. 138]

Many of the applications of statistical tools rely on models providing accurate information outside the documented observations, thus the generalization ability of an algorithm is a key criteria to many fields. Algorithms falling short on their accuracy outside the known spectrum, no matter how powerful, won't be usable and many opportunities are lost. Taking into consideration i.e. pharmaceutical analysis, needing to be able to predict, for instance, the toxicity of a given substance, rely on concepts first used decades ago, as well as on methods that could be completely replaced by modern algorithms, if their accuracy was impecable. [Haw04]

1.3 Objectives

In this thesis the focus lies on adapting a genetic programming (GP) system to actively prevent the tendency to overfit. Meaning that a GP system should be found, minimizing the overfitting of the final solutions, especially regarding long running training scenarios. All this should be achieved while not compromising the overall quality of the solutions, thus the system needs to be able to produce solutions comparable to the standard system and maintain a very low risk of overfitting.

Additionally, as GP systems are by themselves computationally complex, an approach is to be found not drastically increasing i.e. the memory load or computational operations, as even a slightly larger time consumption, might render the approaches impractical or infeasible for use in a productive environment.

1.4 Structure

The thesis is structured into different parts spanning multiple different approaches based of similar ideas. Firstly the underlying theoretical framework is outlined and important components are presented in detail. The third chapter focusses on an initial approach to using semantics to discriminate against overfitting solutions. Subsequently in the fourth chapter, an extended approach is evalutated, followed by a third system being based on a different concept, but using similar information. The fifth and last chapter provides an overview of the results and summarizes the possibilities and open points unveiled by the evaluted approaches.

2 OPTIMIZATION ALGORITHMS

The following chapter presents an overview of the theoretical framework in which GP is embedded, including optimization problems (OPs), search algorithms, evolutionary algorithms (EAs) and GP itself.

2.1 OPTIMIZATION PROBLEM

The term OP refers to the formal abstraction of a set of real-world problems with similarly shaped constraints and solutions. Mathematically speaking, an *instance of an OP* I is defined as the pair (S, f), with S being a set of admissible solutions and $f : S \rightarrow \mathbb{R}$ a quality measure (or cost) function returning a real-valued score for each $s \in S$. In case of a minimization problem[2], the objective is to find

$$s \in S \quad \text{such that} \quad f(s) \leq f(y) \quad \forall\, y \in S.$$

The solution s, of which there might be multiple, is also referred to as the *global optimum* [PS13, see p. 5];[AK89, see p. 5f.].

2.2 SEARCH ALGORITHMS

The set S of solutions, referred to as the search space, has to be traversed by a search algorithm in order to find a solution s satisfying the search objective (minimizing/maximizing the quality/cost function). Since an exhaustive search is mostly infeasible in non-trivial problems[3], a traversal operator $N : S \rightarrow 2^S$ has to be defined, called *neighborhood*, such that for each solution $s \in S$ the function $N(s)$ returns a set of neighboring solutions [PS13, see p. 7]. A search algorithm can then attempt to find a solution by recursively searching the neighborhoods. [AK89, see p. 6f.]

The neighborhood operator has an immediate impact on the efficiency of the search - it shapes the search space according to the quality measure $f(s)$ either unimodal, or multimodal. The plane resulting of arranging the solutions in

[2] A minimization problem in this context refers to a quality function f defined such that $f(a) \leq f(b)$ if a is better than b (maximization would use an inversed function f). For maximization the objective of an OP is inversed to $s \in S$ such that $f(s) \geq f(y) \quad \forall\, y \in S$

[3] I.e. using the notation of P, NP, NP-complete and NP-hard, this refers to any problem of class NP-complete or more complex.

S according to neighborhood and quality f, called *fitness landscape*, determines the complexity of the problem and thus the efficiency and effectiveness of the algorithm used. An unimodal fitness landscape allows any *greedy*[4] search algorithm to effectively retrieve a global optimum for the problem. A multimodal fitness landscape on the other hand is a plane with multiple peaks distributed over the whole search space, thus a greedy search algorithm would always terminate with returning a solution[5]

$$s \in S \quad \text{such that} \quad f(s) \leq f(y) \quad \forall\, y \in N(s).$$

The solution s is referred to as a *local optimum*[6], since the operator N imposes locality on the search [PS13, see p. 8].

2.3 Evolutionary Algorithms

Simple search algorithms, such as Hill-Climbing, have been thoroughly studied and improved upon with different approaches, typically using heuristic methods and targeting premature convergence [Kok05, see p. 4]. One of the several resulting concepts is the area of evolutionary programming, specifically EAs, which mimic processes found in nature and *Darwin's Theory of Evolution*. Conceptually the main abstracted principles are *populations*, *individuals* (see section 2.3.1) and mechanisms such as *reproduction*, *recombination* and *mutation* (see section 2.3.2). [Kok05, see p. 4f.]

EAs may be split into groups of *phenotypic* and *genotypic* algorithms, in respect to the definition of a solution. A genotypic algorithm, such as *Genetic Algorithm* traverses the search space by modifying a representation of the solution, while a phenotypic algorithm, i.e. *GP*, modifies the actual solution. Therefore a genotypic algorithm depends on solutions being a representation defined with a fixed structure, i.e. fixed-length array of characters from a defined alphabet, while a phenotypic algorithm processes dynamic solutions. [Als09, see p. 790]

The main difference to simple search algorithms, i.e. Hill-Climbing, is that EAs search the neighborhood of multiple solutions simultaneously (using populations) with a neighborhood created by the combination of multiple different processes (reproduction, survival, recombination and mutation). [Kok05, see p. 4];[Mit97, see P.250]

[4] A greedy search algorithm, such as Hill-Climbing, traverses the search space by always moving to a better solution in a neighborhood and terminates if none can be found. [Kok05, see p. 2]

[5] If the problem is minimization, otherwise the condition is inversed.

[6] The phenomenon of terminating with a local optimum instead of searching on for a global optimum is called *premature convergence* and is primary subject of research in optimization algorithms [Kok05, see p. 3]

2.3.1 Population and Individuals

In the context of EAs a solution s is renamed to *individual* i and the quality or cost function $f(s)$ is referred to as the *fitness* of individual $f(i)$ [Bli97, see p. 21]. As previously mentioned an EA tries to overcome premature convergence by searching multiple solutions simultaneously, the collection of individuals searched at each step is called *population* P [Mit97, see p. 250].

2.3.2 Search Process

The search process is based of an initial population, which is typically generated randomly, with an uniform spread over the search space [YG10, see p. 17]. Fundamentally EA iterates over the population by executing different transforming phases. At each iteration, referred to as generation, EAs apply three transformations to the held population subsequently [Mit97, see p. 250f.]:

1. **Selection:** Determines which individuals will be processed further and which will be dropped from the search focus, especially granting individuals with higher fitness better odds for surviving. [YG10, see p. 18f.]
2. **Variation:** Explores the neighborhoods of the selected individuals, by applying *genetic operators* such as mutation and crossover. [YG10, see p. 20f.]
3. **Transition:** Creates the new population to be used in the next iteration, also referred to as *replacement*. [YG10, see p. 22]

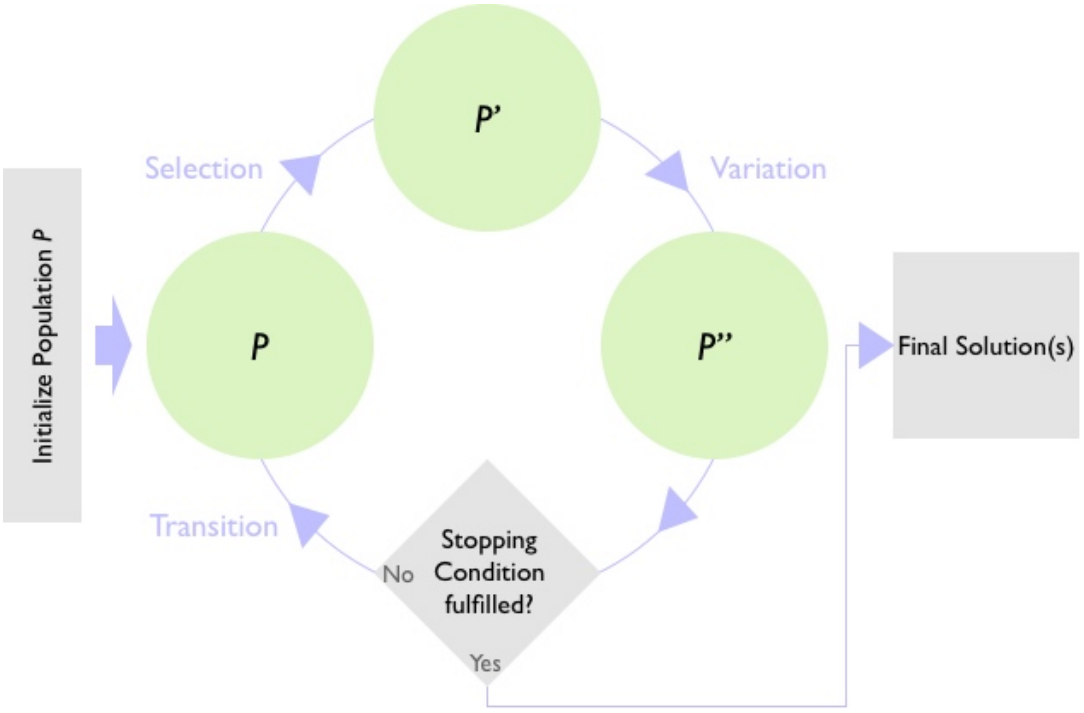

Figure 2.1.: Evolutionary algorithm process overview

Overall the process iterates until a stopping criteria has been reached (such as sufficient fitness), or a certain number of generations have been executed (see figure 2.1 for a process overview) [YG10, see p. 81].

Selection Phase

The first phase a population undergoes is a method to discriminate against solutions of lesser quality. Due to the problems of premature convergance and genetic drift, the selection of individuals has to be non-deterministic, yet not completely random, otherwise the algorithm will loose it's effectiveness [YG10, see p.64f.]. Generally speaking EAs employ a probabilistic selection of $n = |P|$ individuals, where each individual is chosen based on probabilities derived from their respective fitness. This aims at mimicing the idea of "survival of the fittest" according to evolutionary theories, where no individual will be brought to extinction by design, but by likelyhood and fitness. [YG10, see p. 18f.]

Out of many, three important approaches are:

- **Fitness Proportionate Selection (Roulette Wheel):**

  The fitnesses of all individuals are transformed into a probability relative to the total sum of fitnesses. An individual i will be selected with $p_i = \frac{f(i)}{\sum_{j=1}^{N} f(j)}$ with N being the number of individuals. [YG10, see p. 67]

- **Ranking Selection:**

  Similar to fitness proportionate selection, with the difference being that the individuals are sorted by fitness and the resulting rank is used to determine the probabilities instead of directly using the fitness. [YG10, see p. 71]

- **Tournament Selection:**

  The intermediary population is created by repeatedly ($n = |P|$ times) selecting a random subset T of size t of the population. Then an individual $w \in T$ is selected such that[7] $f(w) \leq \quad \forall o \in T$, which is then placed into the new population. [YG10, see p. 74]

Which of the methods is used has to be determined by empirical studies, for each problem at hand[8]. Nevertheless the probabilistic selection will make the population converge to a set of dominant solutions, without regards to which exact technique was used[9].

[7] for minimization (in maximization the objective is inversed)
[8] see no free lunch theorem [WM97]
[9] see schema theorem [Mit97, see p. 261f.]

Variation Phase

The intermediary population retrieved from the previous phase will subsequently be transformed by genetic operators in order to traverse the neighborhood, thus the search space [Mit97, see p. 259]. Typically, and in accordance with naturally observed phenomena, two types of transformation are used: [YG10, see p. 21]

- Crossover: two random individuals are merged and two new individuals result, based on only the genetic material present in the two parent individuals.
- Mutation: an individuals genetic material will be randomly changed.

Each of the methods usually is executed with a certain probability, which should be a parameter of the system. Generally the mutation operator has a very low probability, compared to the crossover operator. [YG10, see p.21]

Transition Phase

In order to begin the next generation, the population has to be replaced by the created offspring population of the variation phase. This replacement can either be total or constrained by i.e. *elitism*: the best n individuals (with n being a parameter of the system) are moved to the next generation without any transformation in order to keep the best found solution(s).[YG10, see p.75f.]

2.4 Genetic Programming

GP as defined by John R. Koza ([Koz92]) is part of a collection of phenotypic EAs, aiming at solving a variety of problems by generating and processing programs as solutions. This section and the following chapters will assume individuals to be structured as trees, as Koza suggested in his original implementation ([Koz92, see p. 71]). The following sections will assess the differences of GP to EA and elaborate on the GP implementation used throughout this thesis.

2.4.1 Particularities of Genetic Programming

The process of GP is mostly equivalent to any other EA. The difference results from the hierarchical structure introduced by the individuals representing computer programs. This includes the way fitness is measured as well as the way individuals are combined during crossover and manipulated during mutation. [Koz92, see p. 73f.]

Solution Structure

An individual itself is built from a defined set $C = F \cup T$, where F is the function set and T is the terminal set. The function set contains any number of domain-specific, recursively combineable functions, taking a specified number of arguments (referred to as arity). Elements of the terminal set are constant components of the program, for instance input variables or constant values and take no arguments, thus closing the program at the given point[10]. [Koz92, see p. 80]

Taking for example the field of symbolic regression, the function set could contain arithmetic operations, such as addition or subtraction, while the terminal set could contain input variables and several different fixed numbers. The following figure (2.2) illustrates this example, by using a tree-representation of the program.

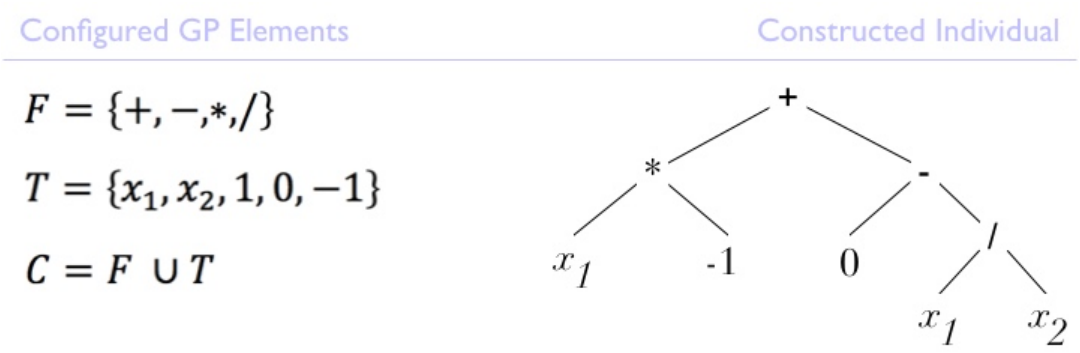

Figure 2.2.: Example function & terminal set and individual in symbolic regression

As can be seen in the figure (2.2), functions are used as internal tree nodes and terminals as leaves. This illustrates the need for functions to be recursively combinable, otherwise it would be impossible to represent complex solutions. Furthermore terminals are required by the system to provide closure to the tree and the represented solution. Other examples, such as the "Artificial Ant" (see appendix A.1), show how Koza's approach can be applied to a variety of fields, as long as solutions can be split into defined functions and terminals.

Fitness

The fitness of an individual i is not directly coded into the individuals, as in other EAs and has to be measured over a set of training examples, so called *fitness cases* [Koz92, see p. 74]. The result of applying the fitness cases to an individual, also referred to as semantics[11] of individual i, can be compared to a target, which

[10] It is crucial for the configuration of any GP to follow the *closure property*, as Koza describes: "The closure property requires that each of the functions in the function set be able to accept, as its arguments, any value and data type that may possibly be returned by any function in the function set and any value and data type that may possibly be assumed by any terminal in the terminal set." [Koz92, p. 81]

[11] Several uses of the term semantics are possible, all referring to a non-syntactical represenation of an individual [MKJ12, see p. 21]

allows for instance the calculation of the root mean squared error (RMSE) that can be used as a fitness measure. [Koz92, see p. 94f.]

x_1	x_2	y
1	2	3
3	4	1
2	1	-1

Table 2.1.: Example fitness cases X with target y

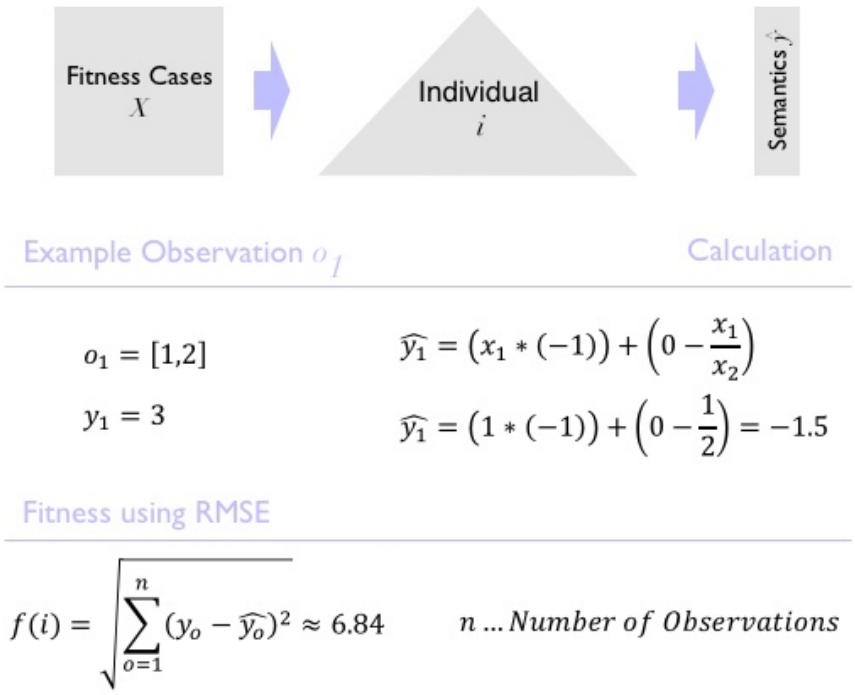

Figure 2.3.: Example fitness computation

Using the example in figure 2.2, figure 2.3 shows the computation of the semantics of the tree, as well as the computation of the fitness, given the sample data in table 2.1.

Standard Variation

As individuals are a collection of recursively arranged functions and terminals, any individual can be easily manipulated by exchanging or replacing subprograms. Crossover is thus implemented by selecting a random branch of one parent and replacing a random branch of the second parent. The replaced branch will then take the place of the missing branch of the first parent [Koz92, see p. 101].

As can be seen in figure 2.4 the tree representaion allows for simple exchange of subprograms between two parents, after selecting random crossover points uniformly over all nodes in the trees.

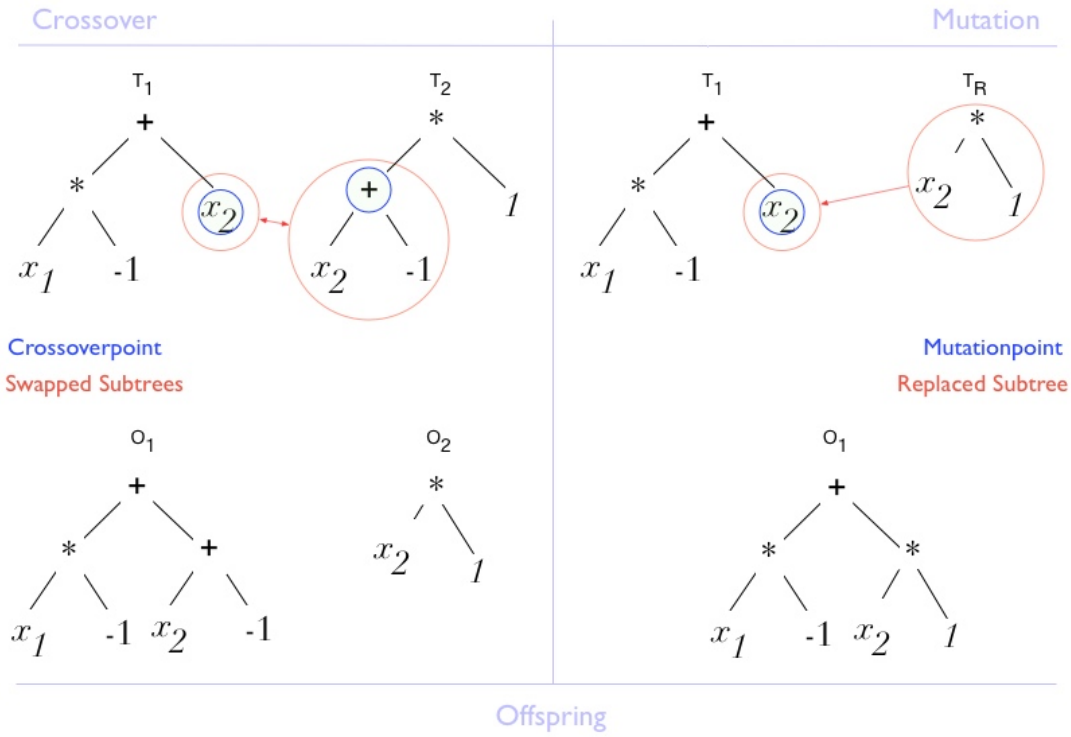

Figure 2.4.: Example variation with crossover and mutation

The mutation operator, analogous to crossover, creates a new random tree (possibly constrained by a maximum depth), and uses it to replace a randomly selected subtree (again using a uniformly distributed selection over all nodes of the tree) of an individual (see figure 2.4) [Koz92, see p. 105].

2.4.2 Implementation

The approach of algorithm 1 conceptually follows the previously, in section 2.3.2 and 2.4.1, presented process. The GP presented here and used in the following chapters aims at solving symbolic regression problems, and has been configured with the arithmetic operators addition, subtraction, multiplication and division[12], as well as several constants in $[-1, 1]$.

In regards to the evolutionary process the implementation is using tournament selection, standard variation operators and an elitist transition phase. In order to, for instance, reduce memory usage or increase efficiency during the execution, a few adaptations have been incorporated, without changing the behavior dramatically:

- No intermediary population: the process of selection and variation can be easily combined into one iteration over the population, eliminating the need to save an intermediary population;

[12]The division operator has been protected from division by zero: it returns its first argument, in case the second argument is zero.

- Conditional evaluation: if an individual is not modified (by crossover or mutation) it is unnecessary to compute the semantics of the fitness cases again;
- Depth constraint: if an individual is too deep, the algorithm discards the individual and replaces it with it's parent;
- Initial depth constraint: maximum depth an individual can reach during the initialization phase[13].

The approach has been implemented in Python, as well as in Java. The latter has been used as a basis for the first part of the thesis, chapter 3, while the Python version was used in the second part, chapter 4.

2.5 OVERFITTING

Overfitting describes the phenomena of any algorithm chosing a hypothesis h_1 as the best solution in regards to a training dataset, while there exists a hypothesis h_b which is worse than h_1 on the training dataset, but better than h_1 on data outside of the training dataset [Mit97, see p. 67]. This negatively impacts the generalization ability of an algorithm and is generally due to noise, such as erronous labels, or irrelevant patterns in the data [Mit97, see p. 68]. Overfitting can be controlled for instance by regularizations or training an algorithm on a subset of the available data while monitoring the performance on the excluded part (i.e. to then find an optimal stopping point, such as when the error on the excluded data starts to increase again).

In the context of this thesis overfitting is required to be automatically detectable, quantifiable and usable by the GP system, therefore a measure has been defined which determines an individuals degree of overfitting (overfitting severity from now on). It uses a collection of individuals, the *validation elite*, to evalutate any individual at any given time. The validation elite is a pool of individuals with fixed size $n_{valelite}$ that contains the globally best individuals in regards to the fitness on a validation dataset. At each generation, the best individual on the training dataset will be evaluated on the validation dataset and becomes a candidate for the validation elite. The validation fitness (fitness of an individual on the validation dataset) of the candidate will be compared to the worst individuals validation fitness of the validation elite. In case the candidate has a better fitness, it will replace the worst individual of the validation elite.

The overfitting of any individual i is then defined[14] as

$$is\_overfitting(i) = f_{validation}(i) > \frac{\sum_{j=1}^{n_{valelite}} f_{validation}(j)}{n_{valelite}}.$$

[13] For details on how the population is initialized, see appendix A.2.1
[14] Minimization. In case of maximization, comparison operators have to be inversed.

The overfitting severity, calculated for every overfitting individual i is defined as follows:

$$severity(i) = \frac{1}{2}\sqrt{(f_{training}(i) - f_{training}(j))^2 + (f_{validation}(i) - f_{validation}(j))^2},$$

with j being a representative of the validation elite, i.e. average, or median. These two measures will be used throughout this thesis as methods to automatically make decisions during GP executions.

Algorithm 1 Standard Genetic Programming

$X :=$ Fitness Cases; $N :=$ Population Size
$t :=$ Tournament Size
$p_c :=$ Crossover Probability
$p_m :=$ Mutation Probability
$d_{max} :=$ Maximum Individual Tree Depth
$id_{max} :=$ Maximum Initial Individual Tree Depth
$P =$ INITIALIZE(id_{max},N) ▷ Initialize N individuals randomly
for all $i \in P$ **do**
    EVALUATE(i, X) ▷ Evaluate individual i on X
end for
while stopping condition not reached **do**
    $P_{offspring} = \{$BEST(P)$\}$ ▷ Elitsm: copy fittest individual in P
    while $|P_{offspring}| < N$ **do**
        $p_1 =$ TOURNAMENT(t,P) ▷ Tournament of size t applied to P
        $r =$ RAND(0,1) ▷ Random number in $[0, 1]$
        if $r < p_c$ **then**
            $p_2 =$ TOURNAMENT(t,P)
            $p_{new} =$ CROSSOVER(p_1,p_2)
            EVALUATE(p_{new}, X)
        else if $r < p_c + p_m$ **then**
            $p_{new} =$ MUTATE(p_1)
            EVALUATE(p_{new}, X)
        else
            $p_{new} = p_1$
        end if
        if DEPTH(p_{new}) $> d_{max}$ **then**
            $P_{offspring} = P_{offspring} \cup \{p_1\}$
        else
            $P_{offspring} = P_{offspring} \cup \{p_{new}\}$
        end if
    end while
    $P = P_{offspring}$
end while
return BEST(P)

3 SEMANTIC REPULSERS IN GENETIC PROGRAMMING

As described in section 2.5, the implications of overfitting can render solutions derived by ML algorithms unusable. The following aims at developing a framework to limit the overfitting in GP by the means of repulsing from such solutions. Firstly conceptual ideas are presented, followed by the implemented algorithm. Lastly the results of the approach will be discussed.

3.1 CONCEPT

It has been shown that forcefully optimizing a ML algorithm might compromise its ability to effectively retrieve general solutions [Die95, see p. 326]. Thus a system has to be developed that provides the algorithm with an "incentive" to evolve towards better solutions, or to evolve away from bad solutions. The latter approach has been used in order to implement the concept of repulsers in GP.

Standard GP has been modified to be able to detect and quantify overfitting according to the definition in section 2.5, as well as extended to incorporate the following aspects:

1. maintain a pool of "repulsers", a pool of fixed size containing overfitting individuals;
2. let GP evolve based on multiple objectives, specifically the fitness and difference to the pool of repulsers.

Using these modifications it is possible to define a system that automatically detects unfit solutions and avoids similar ones by modifying probabilities thus avoiding the application of hard criteria. The following sections will explain details and design choices regarding the additions to standard GP.

3.1.1 Repulser Pool

The repulser pool, similar to the validation elite pool, is a global collection of individuals maintained by the means of predefined overfitting measures (see section 2.5). In each generation the best[15] individual is tested for overfitting and is han-

[15] Or the best n. The number of individuals used can be specified as a parameter of the system. The following assumes one indivdual per generation.

dled as a repulser pool candidate in case the test is positive. The candidate will be compared to the individual in the repulser pool with the smallest severity, the "best" of the repulser pool, if the candidates severity is larger, it will replace the individual in the repulser pool. In case the pool has not reached its maximum size yet, the candidate will simply be added instead of replacing the best individual in the pool. This ensures that over all generations, the most overfitting individuals are collected into the pool, while keeping it at a reasonable size. Additionally the parameter *skip_generations* allows to delay the start of collecting repulsers by n generations. This is used to prevent the pool being filled with the initial, random individuals which are typically bad solutions and not fit for the problem.

3.1.2 Multi-Objective Genetic Programming

As the system is supposed to get an "incentive" to move away from the repulser pool rather than being forced to not choose any solution that overfits, a second objective has to be incorporated into GP. Alongside fitness, the similarity/dissimilarity to the repulser pool needs to have influence on the selection probability during the selection phase of GP. The "fast-non-dominated-sort" algorithm developed by Deb, et al. [Deb+02] in the context of "NSGA-II" is an efficient and simple approach to identify the pareto hierarchie introduced through multiple objectives. It consists of two stages [Deb+02, see p. 184]:

1. Pair-wise comparison: each individual will be compared to any other individual in the population, while checking domination criterias based on the objectives;
2. Extraction of the levels: based on a list collected during phase 1, all levels of the pareto hierarchie can be extracted in one, simple loop. Each individual will be assigned a rank according to its respective pareto level.

The first phase of the sorting algorithm requires the definition of a domination measure which can be applied to two individuals i, j and will return three states, i dominates j, j dominates i or i equals j. Two alternatives for this measure have been used in this thesis, one using a dynamic number of objectives, according to the number of individuals in the repulser pool. The other using exactly two objectives, the fitness and an aggregated measure over the repulser pool. In the first case any individual i dominates an individual j if i is better in fitness, and if all its semantic distances to all individuals in the repulser pool are larger than individual js. In case of the second measure i dominates j if i is better in fitness and if the average semantic distance to all repulsers is larger than individual js.

As mentioned the second objective (or objectives if not aggregated) incorporates the *semantic distance* between individuals. It describes the euclidean distance between the vectors of semantics of two individuals (semantics are the ouput values of the solution over all observations, see section 2.4.1 Fitness) and is used to overcome the expensiveness of tree comparisons. Individuals in GP tend to grow quickly to very large and complex structures, which makes it a highly resource expensive task to syntactically compare two individuals. Therefore a represenation of the individuals has to be found which can be used to efficiently evaluate

similarities. Inspired by geometric semantic genetic programming (GSGP) which focusses on the value of semantics, here the semantics have been chosen to represent an individual and to determine similarities among them.

3.2 Implementation

As seen in figure 3.1 the process of using semantic repulser genetic programming (SRGP) disturbs the general way of retrieving solutions very little: after the variation phase, repulsers are extracted[16] and individuals are processed through the fast-non-dominated-sort algorithm. The repulsers only point of influence is the selection phase, where individuals more different from the repulser pool will be gaining a higher probability to survive than others that are less different.

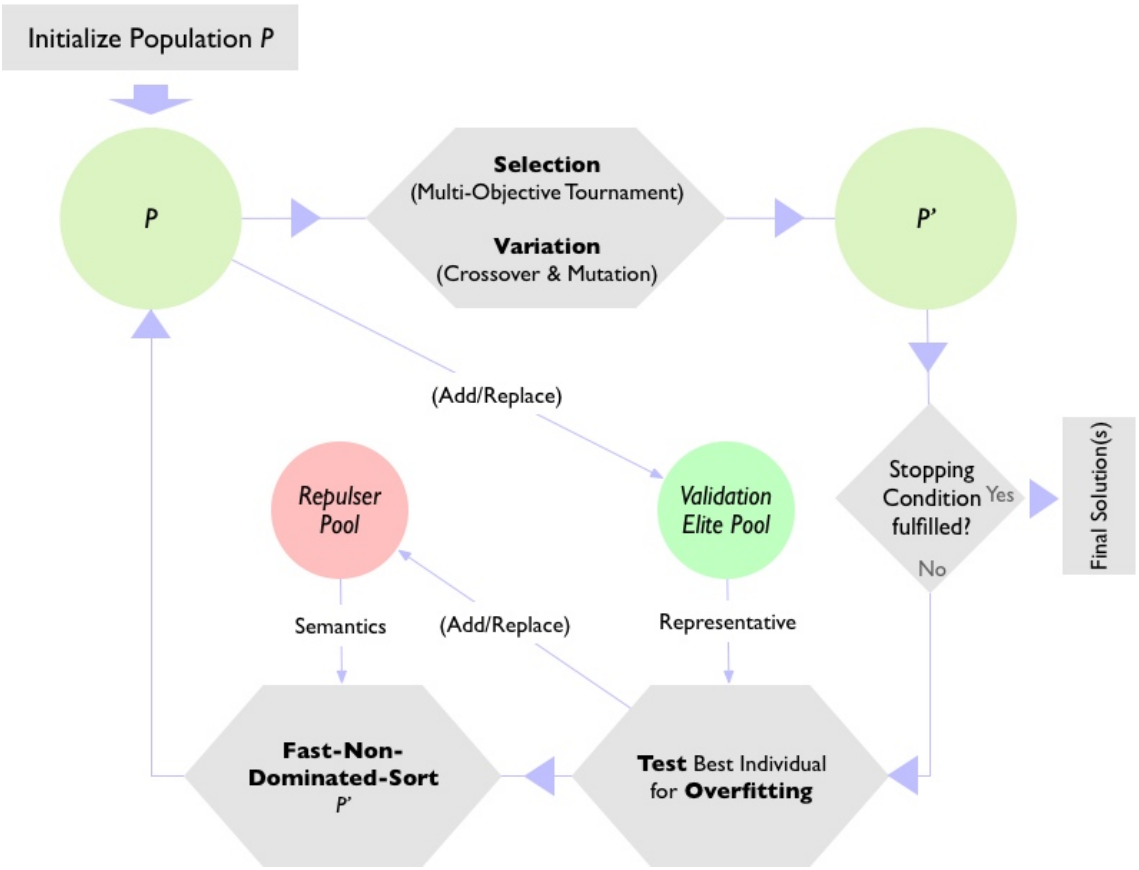

Figure 3.1.: Process Overview Semantic Repulsers in Genetic Programming

To achieve the desired behavior and modify selection probabilities, the tournament selection presented in section 2.3.2 has been used with slight changes to accommodate the multi-objective nature of the system. The tournament, as formulated in algorithm 2, will not discriminate based on fitness, but based on the previously calculated rank of the individual.

[16] See algorithm 4 for details on how the measuring of overfitting and the associated severity is implemented.

Algorithm 2 Tournament Selection: Multi-Objective Implementation

```
function TOURNAMENT(P, n)                          ▷ Apply tournament of size n to P
    w =RANDOM(P)                                   ▷ Select random individual from P
    for all t ∈ {1..n − 1} do
        o =RANDOM(P)
        if RANK(o)<RANK(w) then
            w = o
        else if RANK(o)=RANK(w) & FITNESS(o)<FITNESS(w) then
            w = o
        end if
    end for
    return w
end function
```

Since pareto levels can be occupied by multiple individuals at once, the tournament might run into the situation, where it is impossible to make a decision, as two or more individuals could have equal ranks. In this case the rank is ignored and the tournament falls back to using fitness, being still the primary decision criterion during the evolution.

The implementation of SRGP brings along several new parameters that either need to be adapted according to the problem at hand or could not be defined in an intuitive manner. The following table 3.1 summarizes the most important parameters of the system (for a complete list see appendix A.2.3):

Parameter	Type	Description
validation_set_size	Decimal	Size of the validation set to split of the training dataset
max_repulsers	Integer	Maximum number of repulsers to collect
validation_elite_size	Integer	Maximum number of individuals to keep in validation elite pool
skip_generations	Integer	Number of generations to skip before starting to collect repulsers
use_best_as_rep_candidate	Integer	Number of individuals to use as repulser pool candidates per generation
overfit_by_median	Boolean	Use median instead of average as validation elite representative
aggregate_repulsers	Boolean	Use average distances to repulser pool as objective

Table 3.1.: Parameters of Semantic Repulser Genetic Programming

As mentioned any constant used by the system has been translated into a parameter, which does not necessarily mean they should be treated as such. But through leaving them open to configuration, initial tests can be made with a variety of modifications without the need to rewrite code. For instance, the parameter *overfit_by_median* provides a switch to calculate the overfitting measures not

based on the average, but on the median of the validation elite, which might be a usefull change to avoid outlier-sensitivity.

3.3 Alternative Functionality

The concept of SRGP relies on several predefined principles, such as the measure of overfitting or the nature of the multiple objectives. As these elements can be easily exchanged, various alternatives have been defined during the process of development. The following will describe several additional elements that can be used interchangable with existing functionality.

Overfitting: Slope-based

The default measure of overfitting, defined in section 2.5, relies on building a pool of *best-of* individuals regarding the fitness on the validation dataset and then using its average or median to determine overfitting of individuals. This method will trap the algorithm in assigning the label "overfitting" to increasingly more individuals as evolution progresses, since barely any individual will be able to exceed the validation elites representative. One way to prevent labelling increasing amounts of individuals as overfitting is to increase the size of the validation elite pool, which will make it easier for any individual to reach the upper half of the fitness spectrum of the validation elite. Yet this way the process might ignore crucial individuals that overfit slightly, but are the starting point of overfitting.

Therefore a second measure for overfitting has been defined based on the slope of the curve[17] created through collecting the validation fitness of every best individual at every generation:

- let $Hist_n$ be the validation fitnesses of the last n individuals that have been labelled as the best of a generation and
- let $\bar{f}$ be the average (or median) of $Hist_n$;
- let m_h be the slope of the simple linear regression $y = m_h * x + b$ using $Hist_n$;
- let f_i be the validation fitness of any individual i to be tested for overfitting;
- let m_i be the slope of the simple linear regression $y = m_i * x + b$ using $n - 1$ fitnesses of $Hist_n$ and f_i (leaving out the oldest value in $Hist_n$);

Overfitting and the associated severity are then defined as:

$$is\_overfitting(i) = \begin{cases} true, \text{ if } f_i > \bar{f} \text{ and } m_i - m_b > 0.1 \\ false, \text{ otherwise} \end{cases}$$

$$severity(i) = m_i - m_b$$

[17] Assuming minimization as the problem basis. For maximization the process can be used inversely, but is not considered in the following.

Intuitively this measure tests the change in the slope of the validation fitness curve, assuming individual i is accepted as the next best individual. In case the change in slope is larger than the threshold of 0.1 (and positive) the individual will significantly bend the curve upwards indicating overfitting. But on the otherhand, if the curve has already an upwards tendency, no other individual will be made responsible for the overfitting as long as they keep the same momentum. This has been introduced as one solution to the problem of the previously defined overfitting measure, which would label any individual as overfitting without regards to previous generations.

The alternative overfitting measure comes along with an important parameter, the number of generations to consider for the slope. The parameter is named *n_sighted_steepness* and can be set to any positive integer, while 0 is used as the switch to deactivate it and fall back to the standard overfitting measure. Similarly to the size of the validation elite pool, this parameter changes the pressure on individuals to perform well on the validation dataset. As the parameter increases the slope m_h becomes more stable and a single additional individual will likely have a rather insignificant influence on it. Yet if the parameter is set to smaller values, m_h is increasingly changed by fluctuations and follows even the slightest changes, thus increasing the influence of a single individual.

Objectives: Excluded Fitness

The fast-non-dominating-sort algorithm is used here in conjuction with the fitness and n distance objectives (one for each repulser; unless aggregated by average). Due to the fact that no individual can be dominated by another if it is better in fitness[18] the distance to repulsers becomes a secondary criterion in the decision of the algorithm. The focus on fitness is additionally increased during the modified tournament selection which will turn to fitness in case of equal ranks.

As to restructure the process into one being focussed on the distance objective, a simple switch has been included as the parameter *domination_exclude_fitness* to disregard fitness completely during the fast-non-dominated-sort. This results in fitness being unused in tournament selection, unless two individuals have not been able to dominate each other, in that case the tournament will again use the fitness to handle the equal rank situation. It is important to note, that this does not imply the algorithm will loose the ability to evolve solutions to an optimum. It will be firstly converging away from bad solutions, which is inversely moving the solutions towards an optimum, secondly the tournament will provide better chances to individuals with better fitness.

Semantic Distance: Restricted to Validation Semantics

In the default configuration the distance objectives are calculated based on the combined semantics of training and validation dataset. This provides the largest

[18] Any individual i dominates any individual j only if i is better (and not equal) than j on all existing objectives.

amount of information for the decisions during the fast-non-dominated-sort, yet could introduce a problematic situation. Assuming a repulser r is, in comparison to an individual i, semantically closer to the global optimum regarding the training data, and semantically further away regarding the validation data. The problem arises when i is semantically close to r on the training data, but for instance distant on the validation data. Now the objectives would likely force the algorithm away from individual i since it is overall quite similar to repulser r (training data is usually around 80% of the data), altough individual i is an improvement regarding the generalization ability.

To overcome this problem the parameter *repulse_with_validation_only* allows to disregard the training semantics during the distance calculations, such that the objective only focusses on moving away from inferior semantics of the validation dataset.

Repulser Pool Maintenance: Merge Repulsers

The method described in section 3.1.1 to maintain a fixed-sized pool of overfitting individuals relies on removing the "best" (in terms of least severely overfitting) individual before adding another individual. Since this might be removing individuals from the pool that contain crucial information a second method has been implemented: merging of similar individuals. The mechanism searches to pool for the two semantically closest (based on euclidean distance) individuals and averages their semantics into a new individual[19].

General: Force Distance to Repulsers

Using the distances to repulsers as additional objectives is supposed to create a tendency to more general solutions (see section 3.1). In order to determine the necessity of using a multi-objective system an opposing approach has been embedded, forcing the algorithm away from the individuals collected in the repulser pool. If activated by the parameter *force_avoid_repulsers*, it influences the variation phase, such that if an individual is similar to any of the individuals in the repulser pool, it will be discarded. Similarity in this context is defined as follows:

- let d_{ir} be the semantic distance between individual i and repulser r;
- let d^P_{max} be the maximum semantic distance between any two individuals of population P;
- let e be a parameter of the system (referred to as parameter *equality_delta*).

$$i \simeq r \text{ if } d_{ir} < d^P_{max} * e$$

This implies that the variation phase can take indefinitely long, in case not enough individuals are found satisfying the criterion. Therefore the iteration has been escaped to 50000 loops, in order ensure a timely stopping of the algorithm.

[19] Note: This results in the loss of the syntactic component of the two merged individuals, the resulting individual is therefore only usable by its semantics.

3.4 RESULTS

The scope of the following tests is to determine the effectiveness of the previously developed concept of repulsers. Thus parameters inherited of standard GP, such as population size, or variation probabilities have been fixed to the values presented in table 3.2. These values represent sensible starting points when tuning a GP system and since the objective of this thesis is not to optimize a specific GP instance, but to implement an improving addition, no fine-tuning of these parameters has been conducted.

Parameter	Value
generations	200
population_size	200
tournament_size	4
crossover_probability	0.9
mutation_probability	0.1
max_init_depth	6
depth_limit	true
apply_depth_limit	17

Table 3.2.: Fixed parameter configuration inherited from standard GP

Furthermore a dataset had to be selected accomodating the assumptions of the developed approach. Specifically a dataset had be used that inherently overfits at a given point during the evolution to initially prove the concept usable. The dataset selected contains information retrieved from the stock market in istanbul, consisting of 7 features, 1 target and 536 observations. In all of the tests, the dataset has been split into 70% training data (375 instances) and 30% test data (161 instances). Standard GP has been executed over 30 folds with the above configuration (500 generations instead of 200) in order to get a baseline performance to which the new concept can be compared.

From figure 3.2 it becomes clear, that standard GP is beginning to overfit already after around 50 generations, which is why the parameter *skip_generations* has been set to 50. This ensures the initial, positive phase will not be disturbed. Following the curve of the test fitness, it is also visible that the early overfitting renders 200 generations more than enough to test the basic impact of the new concept.

Since the implementation of SRGP required the creation of several new parameters, the testing has been performed in stages. Instead of cross testing different parameter configurations, one parameter at a time has been evaluated using the previously evaluated parameters. This does leave out certain combinations of parameters, yet is necessary due to the expensive computation. Table 3.4 contains the different settings that have been used to determine the impact of the concept.

The inital tuning has been determined by running the algorithm with each setting on 5 different sortings of the dataset (see appendix A.2.4 for a detailed report

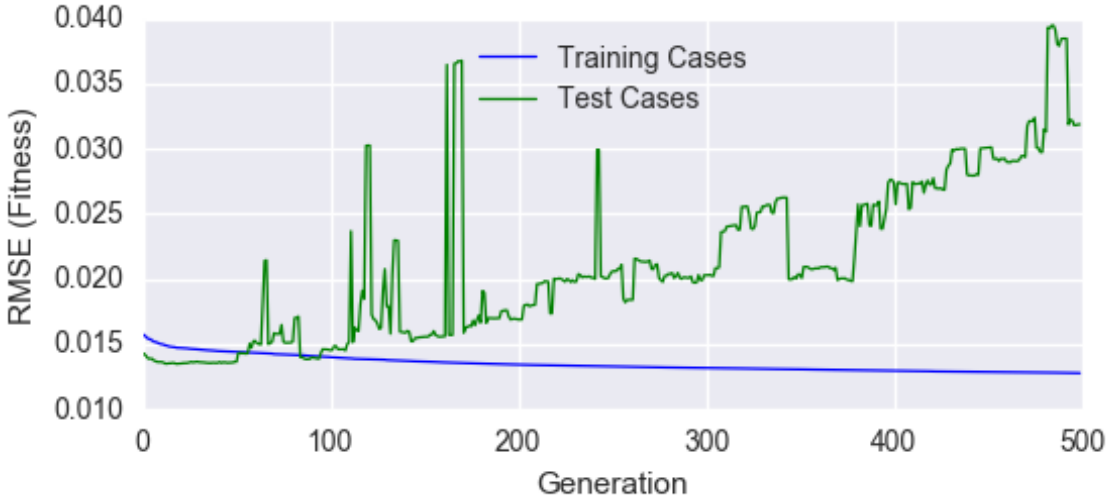

Figure 3.2.: Standard GP: Training vs. Test Fitness on Istanbul Stock Dataset

Parameter	Tested Values	Ideal Value
Validation Elite Size	$10, 25, 50$	10
Repulser Pool Size	$10, 25, 50$	10
Aggregate repulsers	true/false	false
Overfit by Median	true/false	false
Best n as repulser candidates	$1, 10$	1

Table 3.3.: Parameter Tuning: tested and selected settings (ordered by testsequence, based on 5 folds)

on the decisions). Based on the average performance the ideal value has been chosen and the tests moved to the next parameter. A deeper analysis has then been carried out, by running the ideal configuration over 30 different sortings, equal to the folds on which the baseline algorithm had been evaluated.

Figure 3.3 shows in yellow the training and test results achieved by SRGP. Both have been collected over the 30 folds in comparison to standard GP. It becomes immediately clear, that using this configuration, the repulsers have no impact on the general tendency of the algorithm. It does not even create outlying runs, which exceed the perfromance of standard GP (see figure 3.4[20]).

The alternative approaches discussed in section 3.3 have been evaluated one at a time with the best found configuration of the previous tests using the settings presented in table 3.4.

Most of the changes were unable to exceed the default configuration of SRGP, but on the other hand, did not worsen the performance significantly. In these cases the entries have been labelled as *indifferent*. Judging from the analysis described in the appendix (see A.2.4 Additional Tuning) the most promising approach is to exclude training semantics during the distance calculation. The figures 3.3 and

[20] Boxplot whiskers represent maximum 1.5IQR (Tukey Boxplot). This is valid for all boxplots throughout this thesis.

Parameter	Tested Values	Ideal Value	Exceeds default
Merge repulsers	true/false	false	indifferent
Only repulse validation semantics	true/false	true	yes
Slope-based overfitting (n)	$5, 10, 25$	10	indifferent
Exclude fitness	true/false	false	indifferent
Forced avoid repulsers (delta)	$0.1, 0.01, 0.001$	0.01	indifferent

Table 3.4.: Additional Parameter Tuning: tested and selected settings (ordered by test-sequence, based on 5 folds)

3.4 present the result of said approach using 30 folds over the dataset. The results are contrasted with the previously as best defined configuration of SRGP, as well as with standard GP. As can be seen in the fitness plot (figure 3.3) the only improvement over standard GP is a slight smoothening of the curve, meaning both SRGP versions tend to be more stable. Yet the final outcome of the SRGP versions is more spread out than the standard GP.

Based on the results of the described tests, it could be argued that the semantics of an individual do not carry the information necessary to discriminate between overfitting and not overfitting. Thus, in order to support this claim, a series of tests has been executed using the mechanism to forcefully avoid overfitting individuals. As this test has already been carried out over 5 folds, another 25 folds have been executed to remove any doubt of its outcome, which is presented in figure 3.5. Clearly the test negates the argument of the semantics not containing the necessary information, since the solutions generated are not subject to overfitting. Unfortunately, the approach to ignore individuals based on distance to the repulser pool is unusable, as it takes an uncontrollable amount of time to evolve. Moreover the test shows a different learning curve regarding the fitness on the training data. It seems, that even though the curve is rather steep in the beginning, it flattens out after around 100 generations and continues to improve slowly. All in all the test indicates, that the multi-objective approach to providing the “incentive” of moving to non-overfitting solutions is flawed, as it is, based on the semantics, not pressuring the evolution enough.

Figure 3.3.: Final: Training & Test Fitness

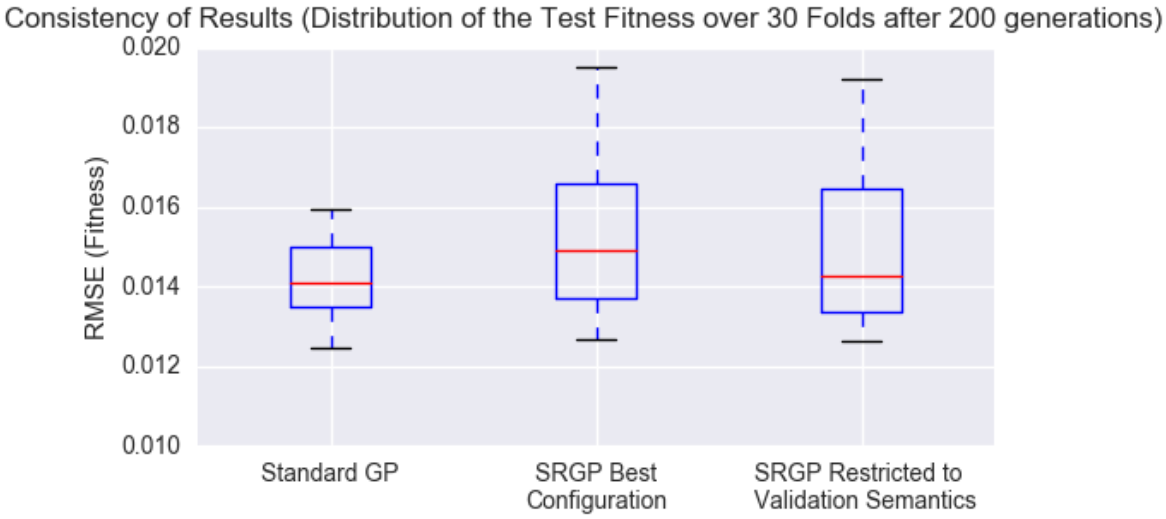

Figure 3.4.: Final: Consistency after 200 generations

Figure 3.5.: Control Series: Forcefully avoiding repulsers (30 folds)

4 INDIVIDUAL'S SIGNATURES IN GENETIC PROGRAMMING

Krawiec and O'Reilly proposed a strategy to capture an individual's behavioral characteristics, in which the semantic vector is extended to a semantic matrix containing the *trace* (from now on semantic signature) of an individual's tree [KO14]. The signature provides a deeper description of an individual and can be used in several ways. After presenting the retrieval of the signature, the following focusses on two approaches targeting overfitting, of which the first is an extension of the SRGP proposed in the previous chapter and the second is a signature-based substitute of the validation dataset embedded into a multi-objective GP.

4.1 Signature of an Individual

The signature of an individual is gathered by evaluating a fitness case and saving the return value of each node into a vector in the order in which the tree is traversed. Doing so for every observation in a dataset results in a $m \times n$ matrix with m being equal to the number of observations and n being equal to the size of the individual's tree. Each column vector in the signature matrix represents the semantics of the respective subtree of the individual, thus the last column vector equals the individual's semantics. As can be seen in the example in figure 4.1 a dataset of two observations and an individual of size 5 will create a 2×5 matrix where each column vector represents the subtrees prediction (the semantics). [KO14, see p. 936]

4.2 Extended Semantic Repulsers in Genetic Programming

As concluded in chapter 3, despite the fact that the semantic vectors of individuals might carry enough information to distinguish between overfitting and not overfitting, the multi-objective system did not create enough pressure on the evolution. Using the signature matrices instead of the semantic vectors might provide the system with the additional information to effectively repulse from overfitting individuals, as pareto fronts can possibly be established in a clearer manner and probabilities are modified unambigously to individuals with better quality. To incorporate the semantic signatures into the concept of SRGP, resulting in the extended semantic repulser genetic programming (eSRGP), only a few changes

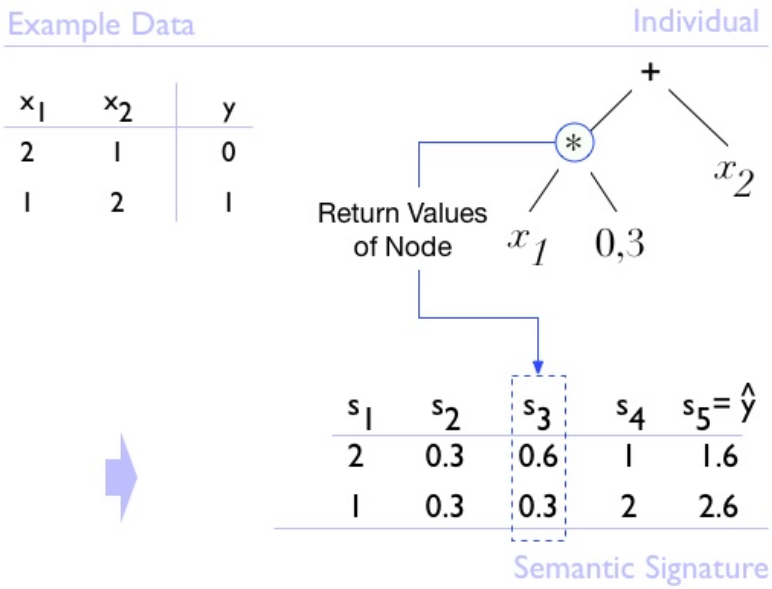

Figure 4.1.: Example of Semantic Signature Retrieval

have to be made which are described in the following sections alongside the implementation and results.

4.2.1 Concept

Using the semantic signature instead of the semantics of an individual poses only a small difference in the implementation. Firstly, while each individual is being evaluated, the signatures have to be extracted as described above (see section 4.1). Secondly, during the fast-non-dominated-sort, a matrix dissimilarity measure has to replace the euclidean distance as a domination criteria. The algorithm can then proceed in exactly the same manner as described in section 3.1 in order to move the population away from a pool of repulsers.

Matrix Dissimilarity Measures

Using the signature matrices as descriptions of individuals opens new opportunities, yet comes along with additional difficulties. The dimensionality of the signature matrices is directly bound to the size of an indiviual, meaning that the system has to be able to compare matrices of unequal size. Multiple measures, including *distance correlation* [SRB07], the *RV coefficient* [RE76], as well as the *modified RV coefficient* (RV2 coefficient) [Smi+09], have been implemented in order to allow tuning towards the most suitable results. Generally the measures should return low values in case of independence, while they should assume high values for largely equal matrices. In order to initially decide for one of the coefficients a simple test has been conducted, determining their discriminative power by applying them in two scenarios. Firstly two random, independent matrices (ideally in

Parameter	Type	Description
max_repulsers	Integer	Maximum number of signatures to keep as repulsers
validation_elite_size	Integer	Number of individuals to keep in the validation elite pool
search_operator	String	Use Multi-Objective (mo) or Single-Objective (so)
penalty_cost	Decimal	Cost parameter for fitness penalization

Table 4.1.: Parameters of the extended semantic repulsers in genetic programming

different random intervals) are compared by the measures and secondly two dependent matrices (i.e. one being a subset of the other) are used. The coefficient with the largest interval between the two scenarios has been chosen to be the default, in this case the modified RV coefficient.

Single-Objective GP

Additionally, the whole system can be switched to a single-objective concept reducing individual's fitness according to dissimilarities to the repulsers. Subsequently the penalized fitness[21] f_p of any individual i has been defined as $f_p(i) = f(i) + SIM(i, R) * c$, with SIM being the average over the similarities of i to all repulsers in R and c being a cost parameter. The cost parameter c is necessary to adjust the penalty to an appropriate impact, since the fitness is problem dependent and the similarity measures are bound to the interval $[0, 1]$.

4.2.2 Implementation

The eSRGP has been implemented in Python in order to leverage the existing third-party libraries covering large parts of the ML paradigms. As this is an extension of the SRGP, most of the functionality has been copied (see figure 3.1), while leaving out approaches that did not show an improvement (such as the slope-based overfitting, or others). In addition to exchanging the use of the semantics with the semantic signatures, the tournament is modified to include a switch to change the search operator between multi-objective and single-objective. The reused and new components can be summarized by the table 4.1, which describes the parameters[22] of the extended approach.

4.2.3 Results

The presented approach functions as an extension to the original SRGP concept, trying to overcome the problem of overfitting with a deeper description of the individuals. Therefore the initial tests have been based of the previously determined configuration for SRGP (see table 3.4 in section 3.4), such that two opposing strategies were examined:

[21] The penalty is added to the fitness in the case of minimization. For maximization the penalty would be subtracted from the fitness.

[22] Excluding the from standard GP inherited parameters, see section 3.4 table 3.2.

1. Best-Of SRGP using the signature and the standard multi-objective setting
2. Best-Of SRGP using the signature and a single-objective setting

Regarding the latter, the cost parameter has been defined to keep the penalty in between $[0, 0.01]$, as the typical fitness using the istanbul stock dataset lies in $[0.01, 0.02]$. The two tests have then been run over 30 folds and compared to standard GP, see figure 4.2.

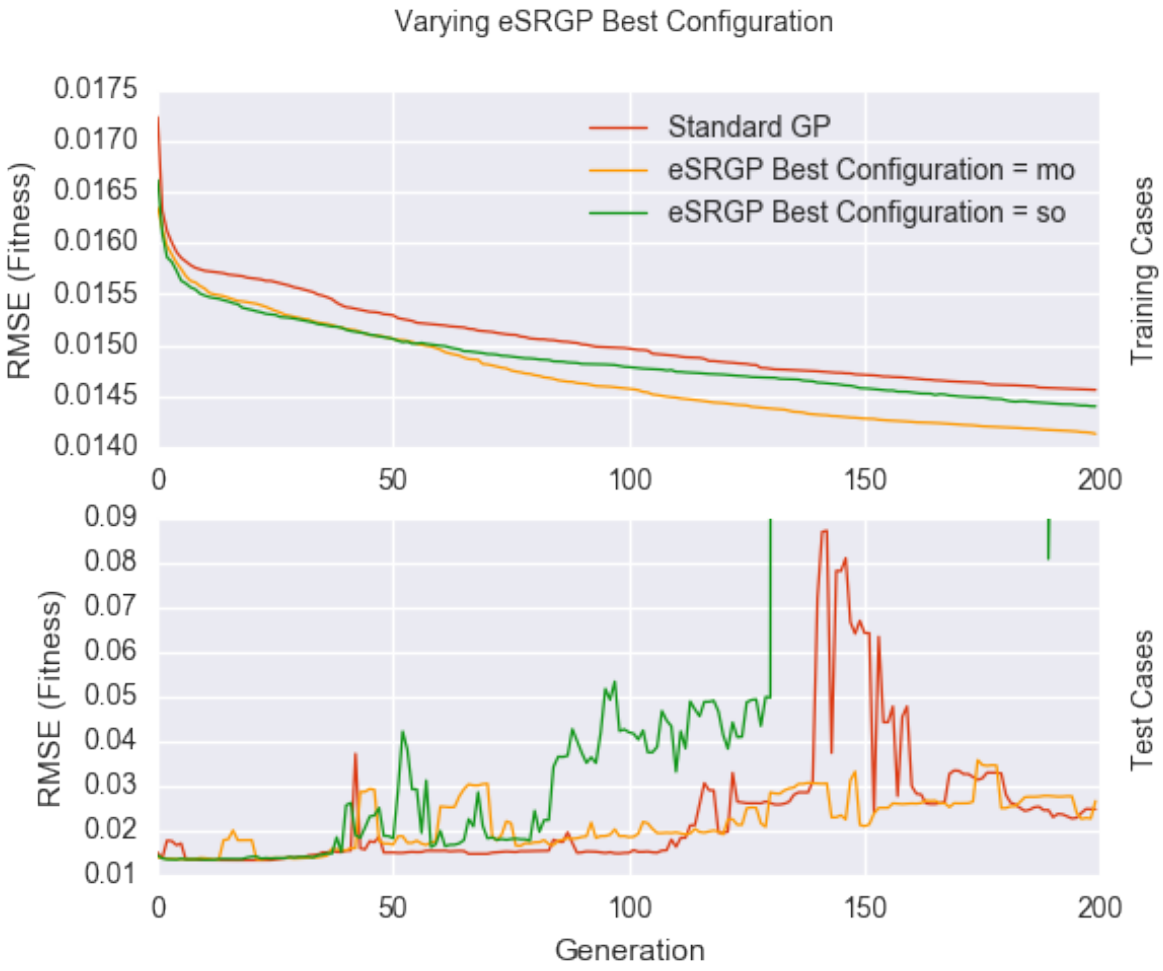

Figure 4.2.: Extended SRGP: Initial Test using Multi-Objective (mo) and Single-Objective (so) Settings (y-axis is truncated for better visibility)

From the above initial tests it becomes clear that neither the multi-objective nor the single-objective system is capable of moving the population away from overfitting solutions. Especially the single-objective system seems to push the population into a wrong direction, as the training error decreases and the test error increases drastically. As can be seen the multi-objective system behaves quite similar to standard GP, implying that using the eSRGP does not overcome the problems mentioned earlier. Even with the increased amount of information, by using the signatures, the multi-objective setting is not creating enough pressure to develop towards higher quality solutions with regards to their generalization ability. Consequently no futher tests regarding the approach of repulsing the evolution of GP

from a collection of overfitting individuals have been carried out.

4.3 Signature-based Substitute of Validation Dataset

The previously described concepts, improving the generalization ability of GP or not, are built with the need to split the available data into a training and validation set in order to run. This need reduces the amount of data, and thus information, on which individuals are built upon, smallering the possibilities of retrieving a solution of high quality. As an opposing approach, allowing to train the GP on the full dataset, while incorporating means to discriminate against overfitting solutions, a two-phased simple multi-objective system based on the signatures of individuals is proposed, described and evaluated in the following sections.

4.3.1 Concept

The system bases of the idea to replace the validation set used in other approaches with an external ML algorithm capable of predicting whether an individual is overfitting, as well as to what degree it is overfitting. The model is trained using individual's signatures extracted in a GP run using a validation dataset and is then applied to a GP run not using a validation dataset, but incorporating multiple objectives, the fitness and the degree of overfitting (both to be minimized). This concept is therefore split into two phases:

1. **Extract signatures and train classifier(s)**: Over several folds of standard GP tuned to the problem, extract individual's signatures (using all available observations) and collect information about overfitting and severity (i.e. using a validation set and a validation elite pool as described in section 2.5). Subsequently the signatures are used to create a classifier capable of discriminating between overfitting and not overfitting. Additionally, but optional, a classifier can be created to predict the severity associated with the overfitting.
2. **Run multi-objective GP**: Use the classifier to run a multi-objective GP using all available observations, while optimizing fitness, as well as the severity/overfitting measures retrieved through the classifier.

Both phases exhibit the need to transform an individual's signature into a feature vector usable for training and prediction respectively. That implies a constantly sized feature vector has to be created from an individual's signature, which is sized according to the underlying program. As a result, signatures cannot be simply "unstacked" into a row vector, since it would lead to differently sized observations. Two approaches overcoming the problem have been considered during the testing of the concept, one truncating the signatures to the first n nodes of the tree (from the root) and then unstacking it into a feature vector, while the other applies feature selection techniques to reduce the dimensionality of the signature to n columns before unstacking it into a feature vector.

Additionally, the concept allows for two different kind of models. As one option a two-layered approach can be chosen, using a binary classifier to predicting whether an individual overfits and subsequently applying a regression model to determine the severity. On the other hand a single model could be used to predict a discretized severity, deticating one bucket solely representing 0-valued, non-overfitting individuals.

4.3.2 Conceptual Tests

As mentioned two approaches for preprocessing the individual's signatures have been used during the evaluation of the concept. The first approach, truncating the signature to the first n nodes of the tree is in the following referred to as *depth-truncated*. Regarding the second approach, *feature-selected*, a recursive feature elimination using SVMs has been performed setting the last column vector (semantics) of the signature as the target. Additionally the continous severity measure has been discretized for the second model choice. Based on the distribution of the severity values, four buckets have been chosen, as depicted in 4.3, and used as labels to train the multi-class model. Generally it has to be noted, that in order to achieve a more clear distiction between overfitting and not overfitting observations, any vector falling below a severity of 0.0005 has been dropped from the training dataset.

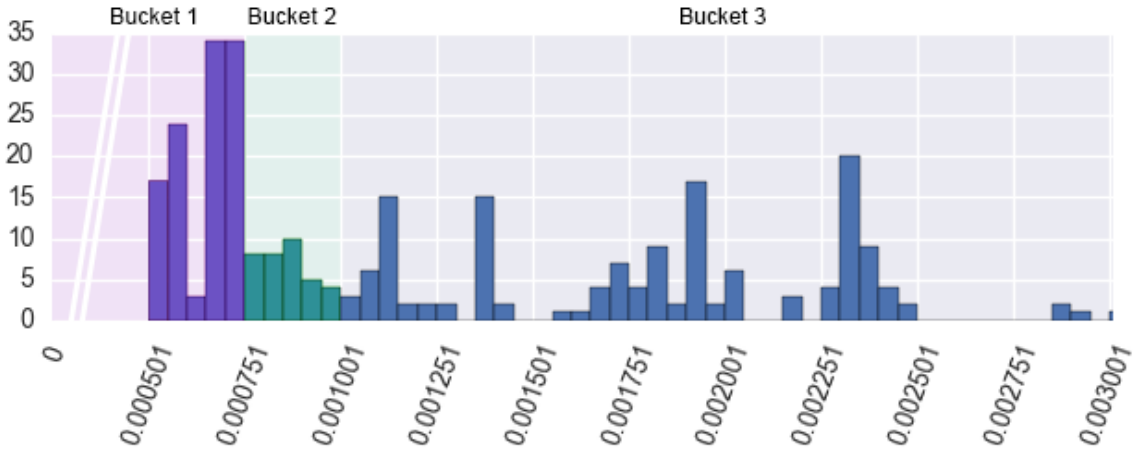

Figure 4.3.: Histogram of severity with assigned buckets (x-axis is truncated at 0.0032, bucket 0 contains all not overfitting observations)

Initially a *depth-truncated* dataset (7 components) has been used to establish a choice of algorithms for the classifiers. Among the models tested were SVMs, MLPs and random forests (as well as decision trees) for the binary, as well as the multi-label classifier. OLSs and gradient boosting regressors have been evaluated for the severity in case the binary model is used. Overall the random forests, as well as the gradient boosting regressors achieved the highest scores on the used dataset and have been chosen for further testing of the approach. Table 4.2 summarizes the best results gathered during the evaluation of several different

Model	Dimensionality Reduction	# Nodes	∅ Training Score	∅Test Score	Measure
Multi-Class (severity bucket)	feature-selected	5	0.9891	0.3800	F1
Binary (overfitting)	feature-selected	15	0.9968	0.4900	F1
Continous (severity)	feature-selected	15	0.000006	0.0030	MSE
Binary (overfitting)	depth-truncated	3	0.9817	0.5500	F1
Continous (severity)	depth-truncated	3	0.0000001	0.00006	MSE
Multi-Class (severity bucket)	depth-truncated	3	0.9259	0.2900	F1

Table 4.2.: Performance of overfitting/severity classifiers in training and test environment (5 folds, see measure column for score meaning)

szenarios, varying the model type, dimensionality reduction technique and number of nodes (columns) to use during the dimensionality reduction (see table A.9 for the complete list of test results).

Intuitively a higher number of components should result in a more accurate prediction. Unfortunately GP is likley to produce highly imbalanced trees, thus using more than 7 components in the depth-truncated approach is infeasible, since almost no individual will be able to fulfill the dimensionality required by the resulting models. Contrarily it is visible, that all of the above test szenarios, even using only 3 components, show the capability of the models to predict precisely the flag of overfitting, as well as the value/bucket of the severity (the latter has slightly worse performance).

Subsequently, in order to verify the classifiers performance in a real GP run, the second phase has initially been replaced with a standard GP run, evaluating the best individual with the classifiers and an independent test set. The prediction of the classifier can then be compared against the true characteristics calculated from the test set and the resulting scores can be derived (see table 4.2 5th column). Unfortunately the test runs were unable to support the expected scores retrieved during the training of the model. None of the tests showed an acceptable performance with regards to precision, recall, F1-Score, as well as MSE, thus negating the assumption on which this approach is based on. The derived models from previously, diversely extracted semantic signatures cannot be used as a substitute to the validation dataset.

There are several underlying aspects which could possibly cause the models to fail during the tests:

- **Individuals are different**: as GP heavily relies on randomized processes, the individuals, thus their signatures, are likely to be different every time the

system is run again[23]. A model trained on the signatures of one batch of runs, won't be able to recognize any of the signatures in another batch of runs.

- **Dimensionality Reduction**: individual's signatures directly depend on the size of the individual's program, thus have to be truncated or reduced to a common size. This reduction implies a loss of information, especially high when truncating everything but a couple of low-level nodes (which are likely to be the same for many individuals), resulting in undistinguishable feature vectors.

Due to the nature of GP being a *grey-box* algorithm[24], it is a highly complicated task to follow through the evolution and determine the aspects responsible for these models to fail. Additionally the models used to classify individual's signatures are themselves *black-box* models not allowing for an analysis of the learned patterns and driving factors during prediction. As these issues are rendering the concept of using a simple multi-objective system to avoid overfitting solutions flawed, further tests and implementations have not been carried out.

[23] Even if the solution trees of two individuals are equal, they could be arranged differently, resulting in different signatures.

[24] see section 1.1 for an explanation of the terms grey-box, black-box and white-box

5 DISCUSSION

The problem of algorithms retrieving overfitting solutions is wide spread and a well-known phenomena in the field of statistics. It is a problem immanent to the algorithms and datasets used, independent of their individual quality - a robust algorithm overfits on heavily noised data, while a fragile algorithm overfits on clean data already [Mit97, see p. 66f.]. While algorithms advance in their ability and datasets explode in size (typically containing large amounts of noise, i.e. sensor data), the problem remains as present as ever. As specifically GP is a very sensible system in regards to ovefitting on unclean and noisy datasets it has been the focus of this thesis. The three different proposed and evaluated concepts have been based of the semantic structure of individuals, rather than working with the syntax of solutions, and have been designed to draw the population away from low-quality solutions rather than moving the population to higher-quality regions.

1. **Semantic Repulsers in Genetic Programming**: using the semantics of individuals the population is pushed away from a pool of overfitting individuals by the means of a multi-objective system.
2. **Extended Semantic Repulsers in Genetic Programming**: in order to support the multi-objective system proposed in SRGP the individual's signatures have been used in this approach.
3. **Signature-based Substitute of Validation Dataset**: modelling the overfitting of individuals based on the signature promised to help building a simple multi-objective system using all available data for training.

The initial approach, SRGP, has been proven to be ineffective, as it did not reduce the level of overfitting throughout the tests. Even through multiple variations, such as modifing the objectives or the way overfitting is detected, no improvements could be generated. Thus it has been concluded on the one hand that discriminating through the multi-objective system using the semantic distance to the repulsers does not have the necessary power to push the population away from low-quality solutions. On the other hand one important aspect has been confirmed: the semantics of an individual could be used to identify overfitting, by comparing them to overfitting individuals (see section 3.3 Force Distance to Repulsers).

The second approach, eSRGP, resulted in similar conclusions, being that the semantic signature does neither support the multi-objective system in increasing

Figure 5.1.: Running times of semantic approaches

the pressure on overfitting individuals, nor provide a stable basis for a single-objective system using penalization of the fitness. Additionally problematic have been the differently sized signature matrices, since the used similarity measures might not provide the coefficients that express what is necessary to distinguish a not overfitting individual from an overfitting one, therefore diminishing any positive impact the signatures could have on the multi-objective system.

The third approach has not been followed longer than the conceptual phase in which testing already revealed ineffectiveness. Despite individual's signatures producing acceptable results during training of the models, the application in a GP run showed large differences in comparison to a real validation dataset. The problem of the differently sized signatures, as already mentioned for eSRGP, possibly is a large contributor to the failure of this approach. As the signatures have been truncated or feature selected in order to provide a fixed sized feature set, valuable information might have been removed, preventing models from effectively discriminating between overfitting and not overfitting.

As summarized above, none of the approaches was proven to be a usable concept to overcome or at least limit the overfitting of GP. From figure 5.1 it becomes clear, why the single working variation of SRGP to forcefully avoid repulsers is infeasible. The computational effort seems to be 4-5 times higher than the average standard GP run needs (measured in seconds per 200 generations). The inefficiency is directly bound to the number of trials, which the variation phase is allowed to perform in order to create individuals being distant from the repulser pool (in the tests this threshold was set to 50000).

Although the suggested approaches did not provide the expected outcome, many more options to incorporating the semantics or the signatures in a GP system working against overfitting exist and should be examined in the future. Addition-

ally and setting the problem of overfitting aside, the signatures of individuals can be used in alternative ways, of which not many have been researched yet.

Multiple objectives in a GP system do not necessarily imply an equal importance of the objectives and therefore the classical multi-objective approach might not create the needed environment to accomplish such goals. All of the described concepts, with one exception, built on using a multi-objective system not discriminating between the objectives and have been shown to not effectively limit the overfitting. Subsequently additional work is necessary in determining an appropriate construct around the repulsing objective in order to provide a significant influence. Such constructs include, i.e. a single-objective system using a linear combination of fitness and repulser distance, a single-objective system restricted to only repulser distance, or even completely different approaches, where the distance criteria leads to the complete purging of individuals from the population. The latter could be approached similarily to the forceful avoidance of repulsers, dropping any individual from the population, that is below a certain distance threshold (this could imply dynamically sized populations, or a mechanism to *refill* the population).

Regarding the use of the semantic signatures one key aspect has been identified to be problematic: the size of the signature matrices. Finding a sensible way to reduce (or increase) the dimensionality to a fixed number of components while retaining as much information as possible and not loosing comparability could increase the effectiveness of the evaluted approaches. The repulser system presented in section 4.2 would benefit from such findings, as the similarity measure could be providing more adequate results and the computational effort could be decreased. Additionally the concept of substituting a validation dataset with pretrained models based on the signatures could become possible again, as it seemed to fail due to a lack of information.

Krawiec and O'Reilly (see [KO14]) not only defined the idea behind the semantic signature but also presented several ways to use the information. One of which is to process the signature performing an automatic feature selection resulting in the n most important columns of the signature matrix. Since columns of the signature matrix correspond to a specific node in the individual's tree, the subtree belonging to that column can be extracted, including the associated subsignature. Generalizing the eSRGP and adding the extraction of subtrees, or -signatures, results in a system depicted in figure 5.2. This system, independent of how the objectives are used to influence the probabilities during the selection phase, can switch between two modes, syntactic repulsing or semantic repulsing. The latter is what has been used throughout this thesis but instead of using the full repulser signature, it uses subsignatures representing the most influencial parts of the repulser. Similarily the syntactic mode will as well not *ban* whole individuals, but rather filter the individual for the parts likely to be responsible for the overfitting. Clearly, as it cannot be a distance anymore, the notion of similarity to repulsers has to be redefined for the syntactic approach to, i.e. the number of occurences of a repulser tree in an individual's tree.

Furthermore the signature can be used to analyze and process individuals after

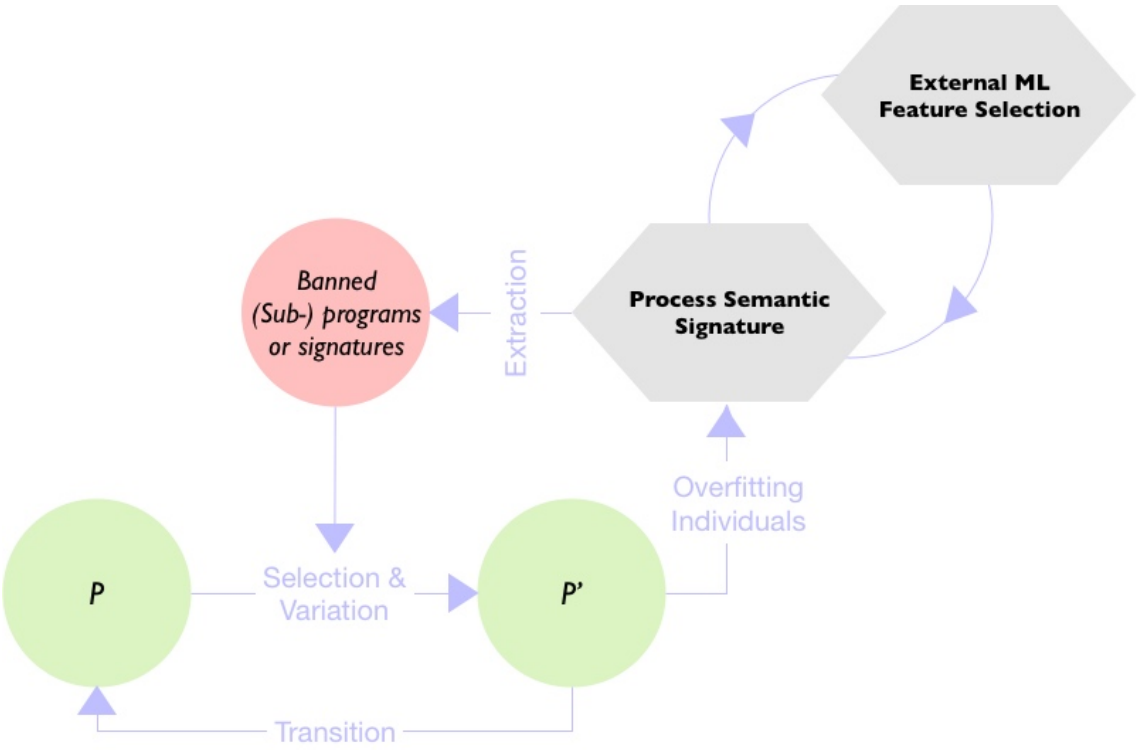

Figure 5.2.: Signature-base approach to repulsing from overfitting individuals using syntactical or semantic information

a GP has returned a final solution. As signatures reveal the behavior of the individuals on the training data, one possible use could be to determine the imporance of specific parts of the solutions. Such analysis might provide the means to group and abstract the solutions such that they can be reformulated into a less complex, possibly even comprehensible representation. It could as well be possible to not only simplify the solutions but even to exclude certrain parts due to insignificant influence. I.e. a column of the signature matrix containing always the same value, could be transformed into a constant, as it would not make sense to keep the subtree associated to it.

Even though most of the above ideas and suggestions pertain to the subject of this thesis, many possibilities exist targeting the same objective, overfitting. And as this issue persists in the field of machine learning, so does the task of improving upon the algorithms aiming at a better generalization ability, especially regarding long-running GP systems.

A APPENDIX

A.1 GENETIC PROGRAMMING APPLICATION: THE ARTIFICIAL ANT

The *artificial ant problem* describes a scenario in which the GP is configured to create a simple program "navigating an artificial ant so as to find all the food lying along an irregular trail." [Koz92, p. 54] The function set consists of several operators capable of chaining the execution of the terminals. These operators are a conditional, "if-food-ahead" which allows for a branching of the program, and two simple chains, "progn2" and "progn3", which allow the unconditional execution of two respectively three subprograms. The terminal set on the other hand contains three elements corresponding to a turn left by 90°, a turn right by 90°and a step forward. [Koz92, see p. 148f.]

Using the functions and the terminals the program needs to move the artificial ant over a square matrix in which cells along a path are marked as food. The fitness of a program is the amount of cells marked as food it passes in a finite number of steps. The single fitness case used by Koza is the "Santa Fe Trail" which incorporates several difficulties along the path, such as angles and gaps of different length [Koz92, see p. 54]. Koza showed the effectiveness of GPs using this example as after already 21 generations an optimal solution was derived. [Koz92, see p. 154]

A.2 Semantic Repulsers in Genetic Programming

The following contains parts of the implementation and results, not included in the main body of the thesis in order to prevent redundancy.

A.2.1 Population Initialization

According to Kozas implementation the initialization of the population should be fully random, such that trees are built from the root up by randomly selecting elements from the combined set. Only the root node, is forced to be an element of the function set, in order to prevent the creation of degenerate trees. [Koz92, see p. 91f.]

Koza has shown, that a more uniform distribution can be reached by applying algorithm 3, which splits the number of individuals into $n = id_{max} - 1$ (id_{max} as used in algorithm 1 is the maximum initial depth parameter) equal sized buckets and numbers them from 1 to id_{max}. The process then iterates over the buckets growing half of the assigned individuals to a full depth of the buckets number (by chosing only elements from the function set, until the last level is reached). The other half of the buckets will be grown randomly according to Kozas approach, stopping with a terminal when the tree is about to exceed the buckets number in depth. This process is referred to *ramped up half and half* initialization. [Koz92, see p. 93]

Algorithm 3 Population Initialization: Ramped Up Half and Half

```
function RAMPED(id_max, N) ▷ Create N individuals with maximum depth id_max
    P = ∅
    n_b = N / (1+id_max)
    for all b ∈ {1..id_max} do
        for all i ∈ {1..n_b} do
            if i%2 = 0 then
                P = P∪ FULL(b)
            else
                P = P∪ GROW(b)
            end if
        end for
    end for
    return P
end function
function FULL(d)
    ... Grow individual to full depth d
end function
function GROW(d)
    ... Grow individual to random depth up to d
end function
```

A.2.2 Overfitting Measurement

The following algorithm depicts two functions used to determine the binary flag "overfitting" of an individual and its associated degree of overfitting, the severity.

Algorithm 4 Overfitting Calculation

function OVERFITTING(i) ▷ Return if i is overfitting or not

$avg_{valelite} = \frac{\sum_{n_{valelite}}^{j=1} f_{validation}(j)}{n_{valelite}}$

if $f_{validation}(i) > avg_{valelite}$ **then**

return True

else

return False

end if

end function

function SEVERITY(i) ▷ Return if overfitting severity of i

$avg_{valelite}^{training} = \frac{\sum_{j=1}^{n_{valelite}} f_{training}(j)}{n_{valelite}}$

$avg_{valelite}^{validation} = \frac{\sum_{j=1}^{n_{valelite}} f_{validation}(j)}{n_{valelite}}$

return $\frac{1}{2}\sqrt{(f_{training}(i) - avg_{valelite}^{training})^2 + (f_{validation}(i) - avg_{valelite}^{validation})^2}$

end function

A.2.3 Full Parameter List

The following table contains the full list of parameteres of the implementation of SRGP.

Origin	Parameter	Type	Description
Standard Genetic Programming	generations	Integer	Number of generations to execute the evolution
	population_size	Integer	Number of individuals in the population
	tournament_size	Integer	Number of individuals to use for tournament selection
	crossover_probability	Decimal	Probability to apply crossover operator
	mutation_probability	Decimal	Probability to apply mutation operator
	max_init_depth	Integer	Maximum depth to which individuals are initialized
	depth_limit	Integer	Maximum depth an individual is allowed to grow to during evolution
	apply_depth_limit	Boolean	Apply the depth limit
Semantic Repulser Genetic Programming	validation_set_size	Decimal	Size of the validation set to split of the training dataset
	max_repulsers	Integer	Maximum number of repulsers to collect
	validation_elite_size	Integer	Maximum number of individuals to keep in validation elite pool

Origin	Parameter	Type	Description
	skip_generations	Integer	Number of generations to skip before starting to collect repulsers
	use_best_as_rep_candidate	Integer	Number of individuals to use as repulser pool candidates per generation
	overfit_by_median	Boolean	Use median instead of average as validation elite representative
	aggregate_repulsers	Boolean	Use average distances to repulser pool as objective
Additional (Semantic Repulser Genetic Programming)	force_avoid_repulsers	Boolean	Discard individuals that are semantically too close to any repulser
	equality_delta	Decimal	Fraction of the intra-population, maximum semantic distance used to determine the threshold for forcefully avoiding repulsers
	domination_exclude_fitness	Boolean	Evolve only with semantic distance to repulsers as objective
	merge_repulsers	Boolean	Average two repulsers instead of replacing the better one in the repulser pool
	n_sighted_steepness	Integer	Use n-sighted steepness measure (n being the value of the parameter) for overfitting (0 prevents the use)
	repulse_with_validation_only	Boolean	Use only the validation semantics during fast-non-dominated-sort

Table A.1.: Parameters of Semantic Repulser Genetic Programming

A.2.4 Results

The following section explains the steps taken to evaluate SRGP and determine the best parameter settings on the Istanbul Stock dataset. As mentioned in section 3.4 the tests have been executed in stages, meaning that each parameter has been evaluated by itself using arbitrarily defined values for the others, and the best values for the previously evaluated parameters. If not mentioned otherwise, each configuration has been run 5 times on the dataset, using 5 differently sorted versions, any data presented in the following will refer to the average over those 5 folds.

Validation Elite Size

For the size of the validation elite pool the values $10, 25, 50$ have been tested using for the other parameters (see table 3.2 for configuration of the standard GP parameters) the following configuration:

Parameter	Value
max_repulsers	25
use_best_as_rep_candidate	1
overfit_by_median	false
aggregate_repulsers	false

Table A.2.: Parameters for variying validation elite pool size

It can be seen in figure A.1[25], with standard GP in red, that generally the repulsers seem to affect the evolutionary process. From the graph it becomes clear, that none of the configurations shows an improvement over standard GP, yet regarding the parameter of the validation elite size, the number 10 appears to be a rather good setting.

Expressing the outcome of the 5 folds of each configuration in a box-plot shows the consistency with which the algorithm finishes the evolution. It is important to highlight in figure A.2[26], that all three configurations (including standard GP) have very similar behavior.

[25] In this and the following figures, the y-axis might be truncated in order to show relevant data only.

[26] Boxplot whiskers represent maximum 1.5IQR (Tukey Boxplot). This is valid for all boxplots throughout this thesis.

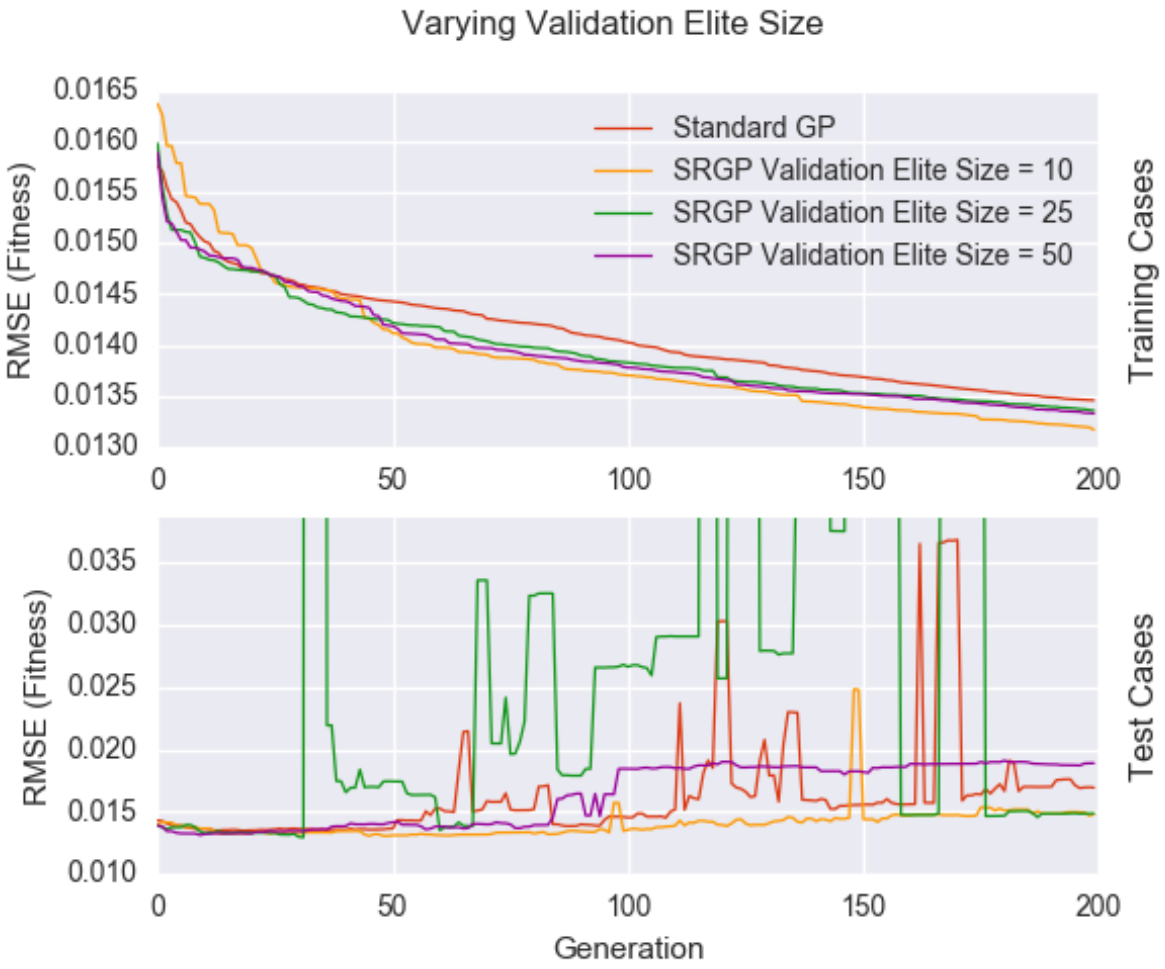

Figure A.1.: Validation Elite Size: Training & Test Fitness

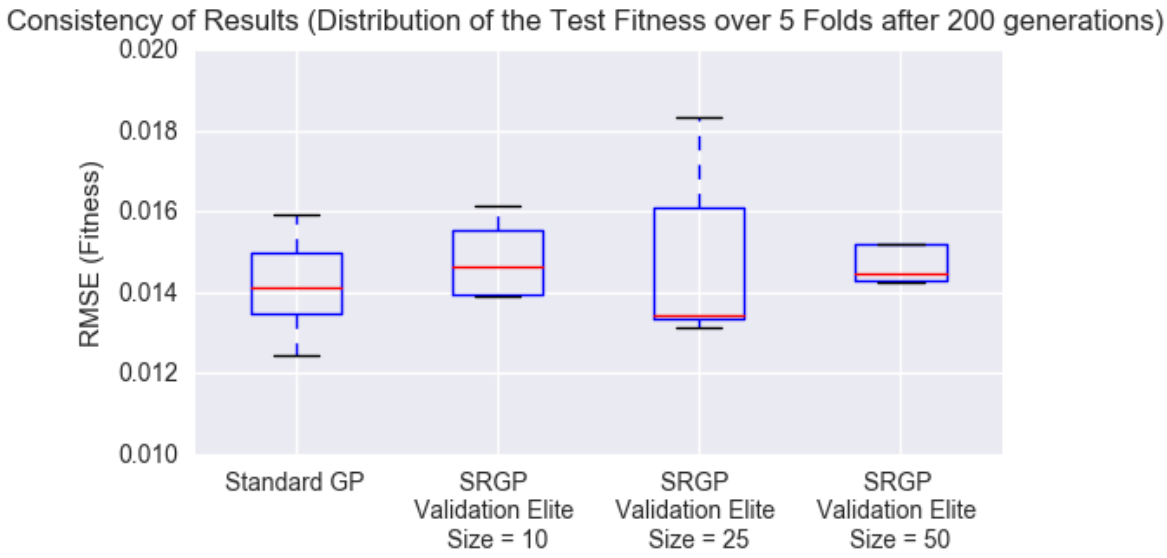

Figure A.2.: Validation Elite Size: Consistency after 200 generations

Repulser Pool Size

As the number of collected repulsers has an immediate impact onto the multi-objective system, it will be tested in three settings: $10, 25, 50$. The same parameters are being used as before, with the exception of the validation elite pool size, which is being set to the previously best found value.

Parameter	Value
validation_elite_size	10
use_best_as_rep_candidate	1
overfit_by_median	false
aggregate_repulsers	false

Table A.3.: Parameters for variying repulser pool size

From assessing figure A.3 it can be said, that the configuration with a repulser pool size of 10 seems to be working the most reliable, as well in regards to the consistency visible in figure A.4. It has to be noted here though, that even tough the test with a repulser pool size of 25, in green, is equal to the winning test of the validation elite size, it has a completely different behavior (compare to yellow line in figure A.1). That shows that possible assumptions derived from the graphs have to be assessed later with further detail, thus the here derived parameter configurations will be tested again using 30 folds, as has been used for standard GP.

Figure A.3.: Repulser Pool Size: Training & Test Fitness

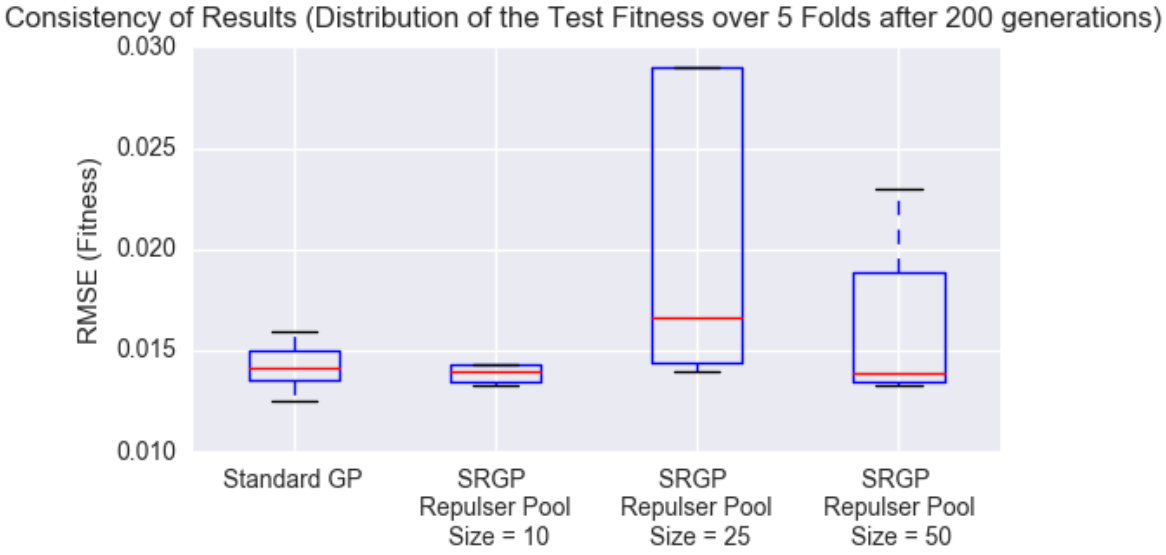

Figure A.4.: Repulser Pool Size: Consistency after 200 generations

Aggregate Repulsers

Section 3.1.1 described a second approach to using the semantic distances of an individual to the repulsers, which is aggregating them to an average in order to obtain a fixed, two-objective system. Using the parameters in table A.4 it has been found, that using the distances as distinct objectives provides a better base for the algorithm (see figure A.5 and figure A.6), although the difference seems to be marginal.

Parameter	Value
max_repulsers	10
validation_elite_size	10
use_best_as_rep_candidate	1
overfit_by_median	false

Table A.4.: Parameters for varying aggregation of repulsers

Figure A.5.: Aggregate Repulsers: Training & Test Fitness

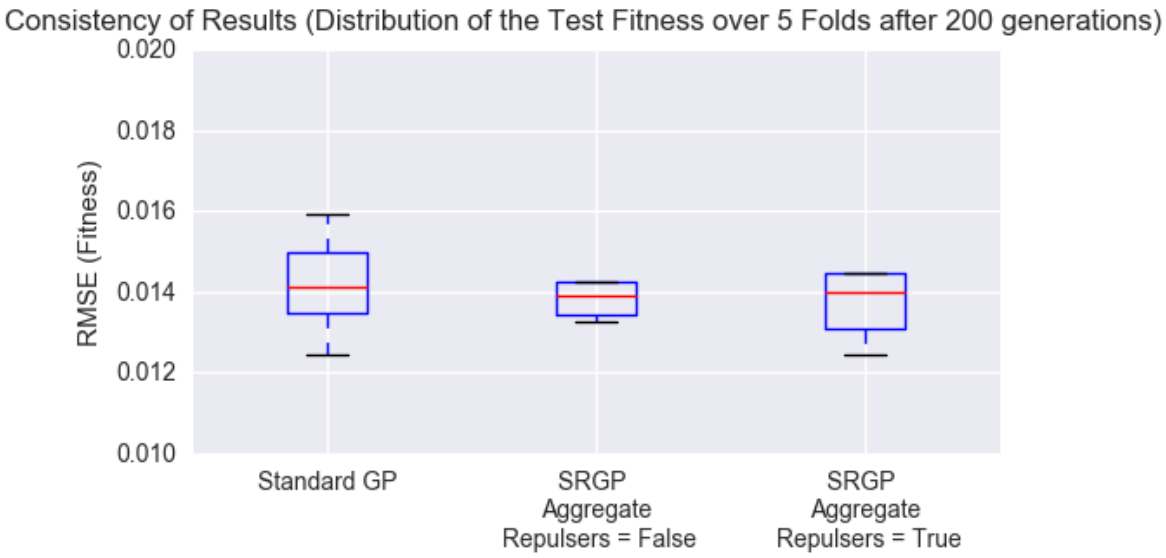

Figure A.6.: Aggregate Repulsers: Consistency after 200 generations

Overfit By Median

The fact that the validation elite is a collection of the best found individuals (see section 2.5) opens the possibility of collecting one or more outliers into the pool. In order to overcome the problem, that anormally well performing solutions distort the validation elite representative, the median can be used to determine a representative individual (the representative will then be used to determine overfitting of any individual). The following parameters (table A.5) have been used in order to test the impact of switching the representative computation:

Parameter	Value
max_repulsers	10
validation_elite_size	10
use_best_as_rep_candidate	1
aggregate_repulsers	false

Table A.5.: Parameters for varying validation elite representative

The figures (A.7 and A.8) show that using the average has a better influence on the overall tendencies of the system. Again it has to be noted, especially regarding the spike around generation 35 in the green line, during which no repulsers are used yet, that bad results here might be due to the probabilisitic and random movement of the algorithm.

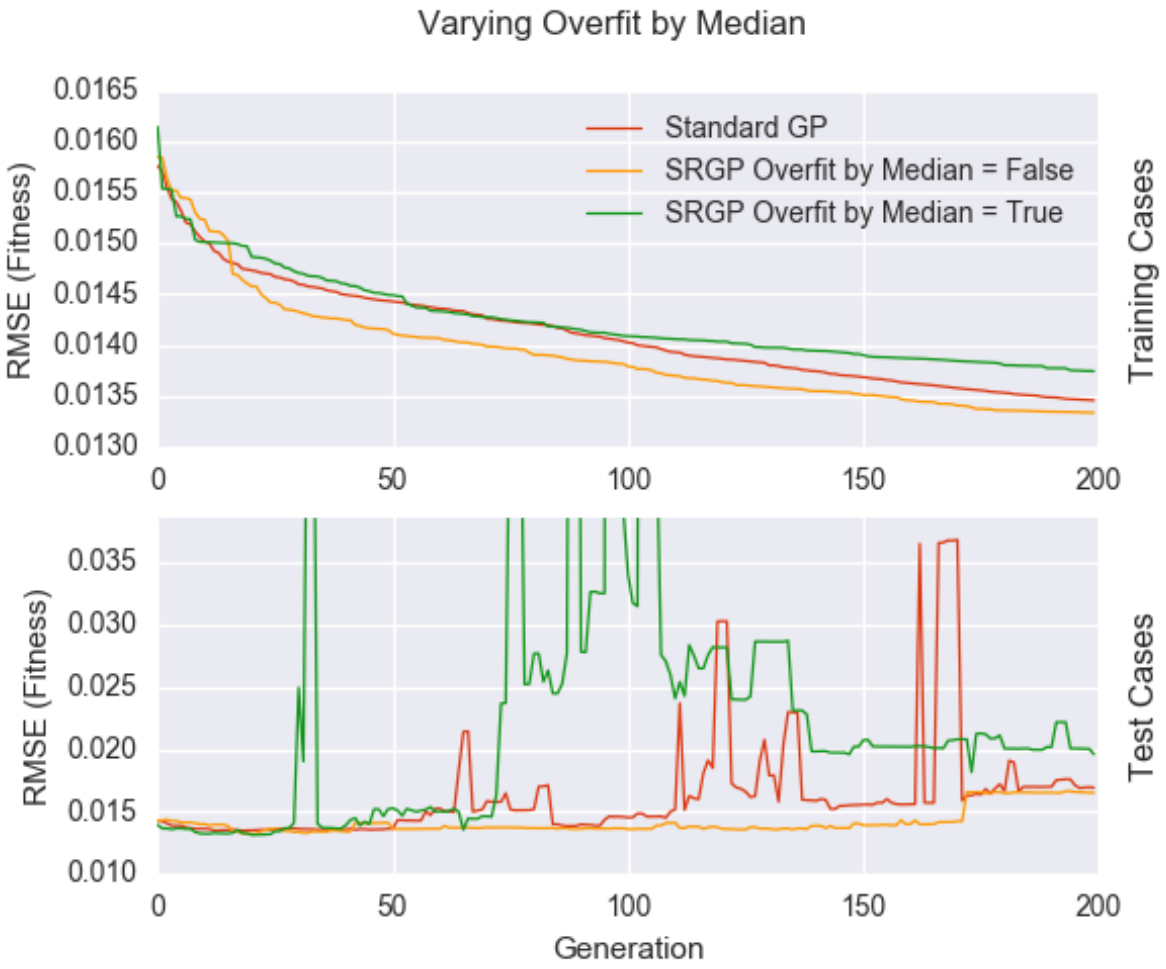

Figure A.7.: Validation Elite Representative: Training & Test Fitness

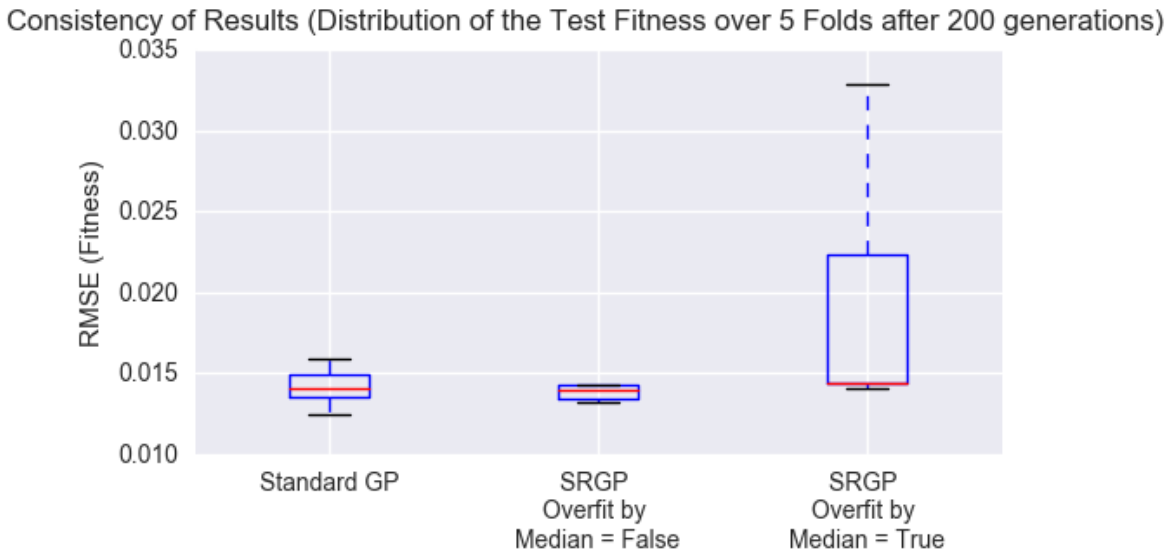

Figure A.8.: Validation Elite Representative: Consistency after 200 generations

Repulser Pool Candidates

Restricting the selection of candidates to only the best individual per generation, might lead to missing a crucial individual regarding future development. Therefore the number of tested candidates per generation has been increased to 10 and compared to the baseline version, using the following parameters:

Parameter	Value
max_repulsers	10
validation_elite_size	10
aggregate_repulsers	false
overfit_by_median	false

Table A.6.: Parameters for varying number of repulser pool candidates

The graphs of fitness (figure A.9), as well as consistency (figure A.10) show clearly that the selection should be restricted to only the best individual per generation. This might be due to the fact, that increasing the number of individuals merged into the repulser pool, increases the number of individuals removed from it, due to the fixed size. A high fluctuation rate in the repulser pool might prevent a consistent direction of the impulse given from the pool, such that the algorithm looses its ability to move towards an optimal and distant to the repulser pool solution.

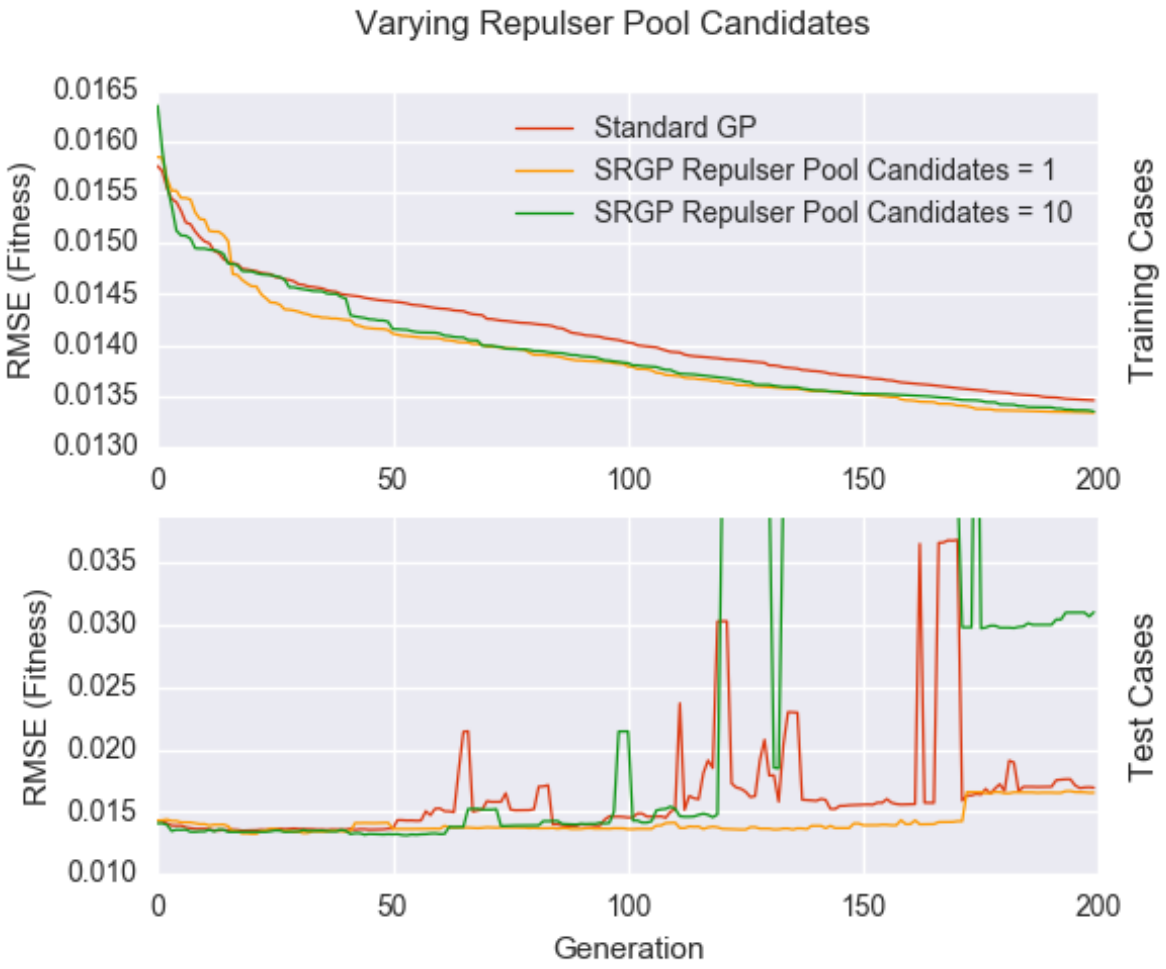

Figure A.9.: Repulser Pool Candidates: Training & Test Fitness

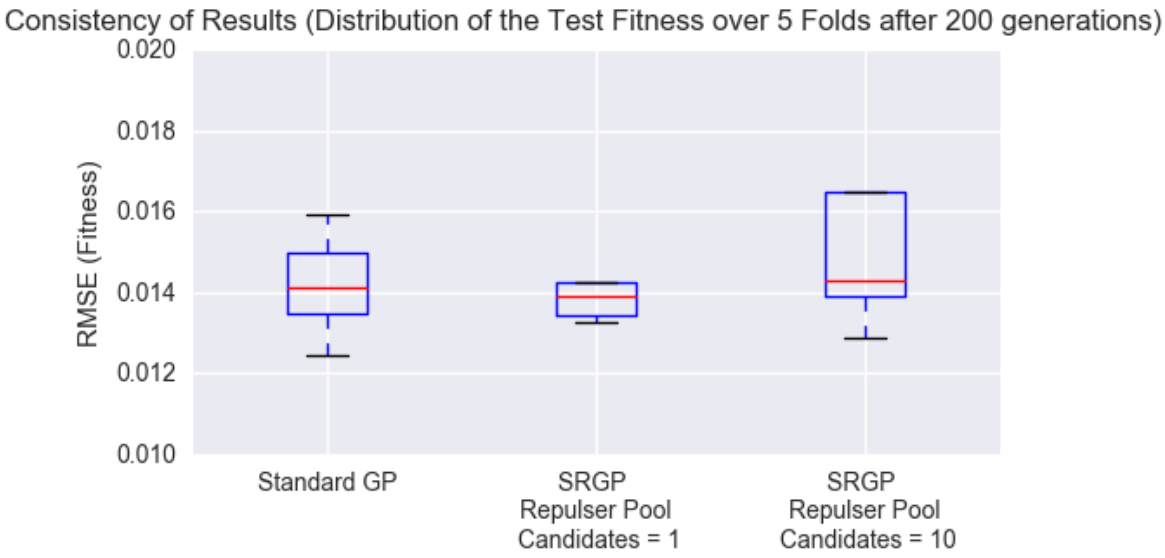

Figure A.10.: Repulser Pool Candidates: Consistency after 200 generations

Merge Repulsers

Merging individuals in the repulser pool instead of replacing them, when a new individual is added, was suggested in section 3.3 and has been tested using the previously found parameters:

Parameter	Value
max_repulsers	10
validation_elite_size	10
use_best_as_rep_candidate	1
aggregate_repulsers	false
overfit_by_median	false

Table A.7.: Parameters for alternative functionality

Based on the results, none of the settings is dominating the tests. Using the replacement technique, as well as the merging, results in equally stable behavior. The only visible difference is the consistency, yet this might be due to a random influence. Generally it seems that either technique fits the concept well.

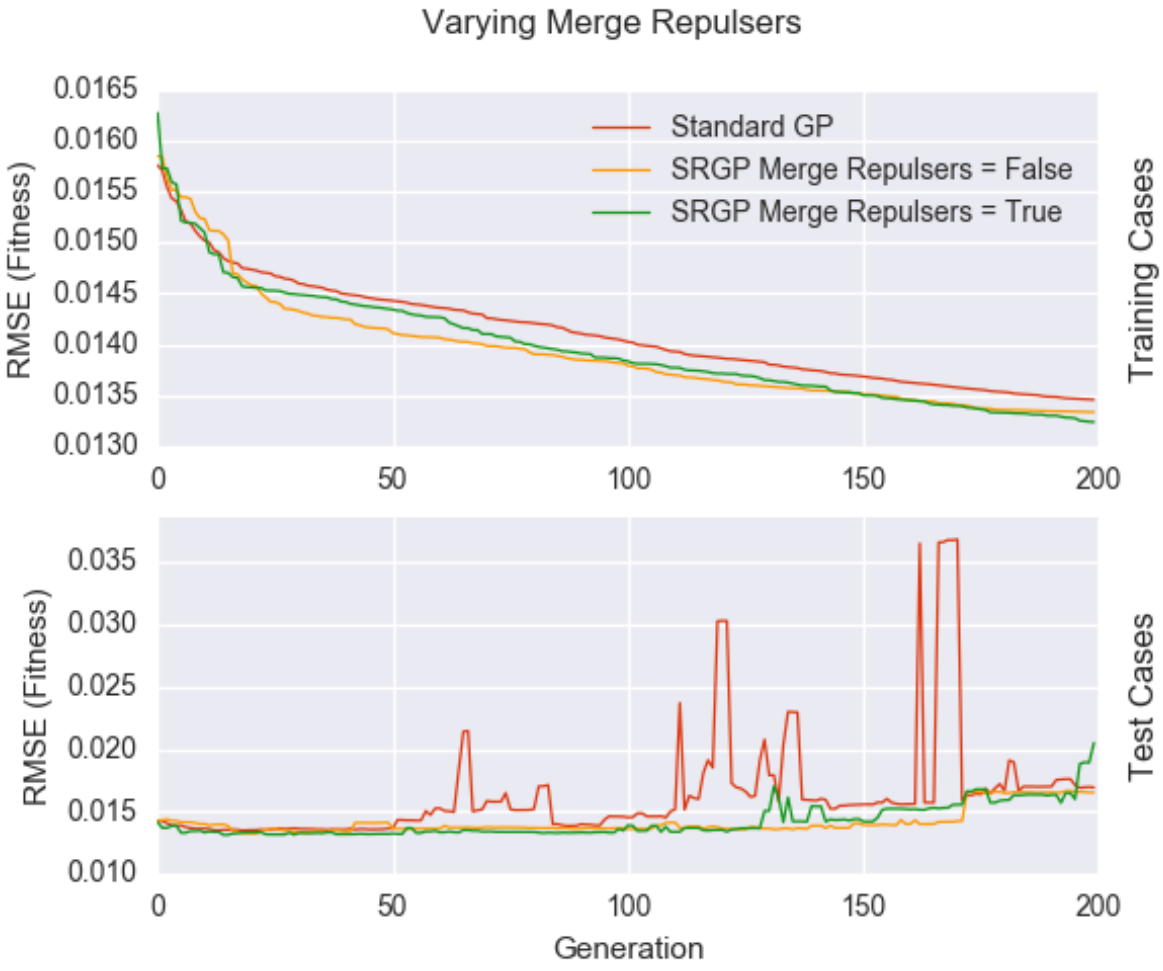

Figure A.11.: Merge Repulsers: Training & Test Fitness

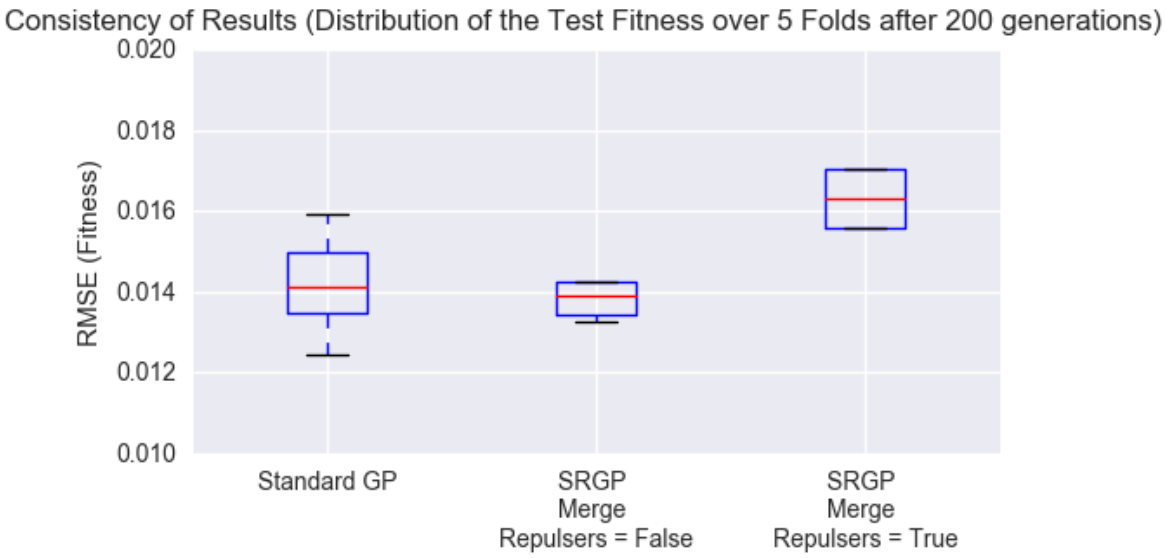

Figure A.12.: Merge Repulsers: Consistency after 200 generations

Restrict to Validation Semantics

Described in section 3.3, the following presents the results of testing the exclusion of training semantics from the process of fast-non-dominated-sort. The tests have been executed using the above described settings shown in table A.7.

The plots show only slight differences in the behavior, yet restricting the distance calculation to only the semantics of the validation dataset, seems to improve the convergence, as well as the consistency of the results slightly (see figure A.13 and A.14).

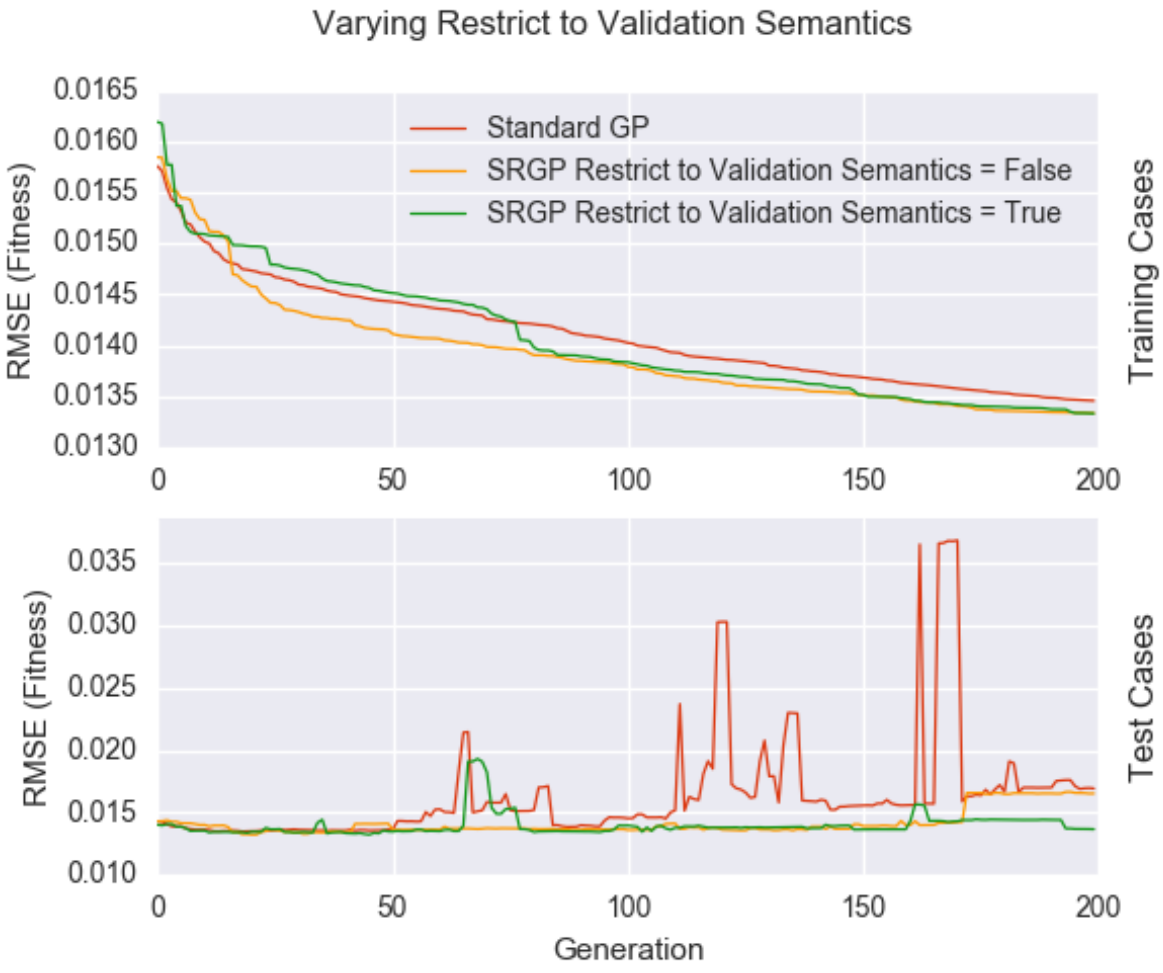

Figure A.13.: Restrict to validation semantics: Training & Test Fitness

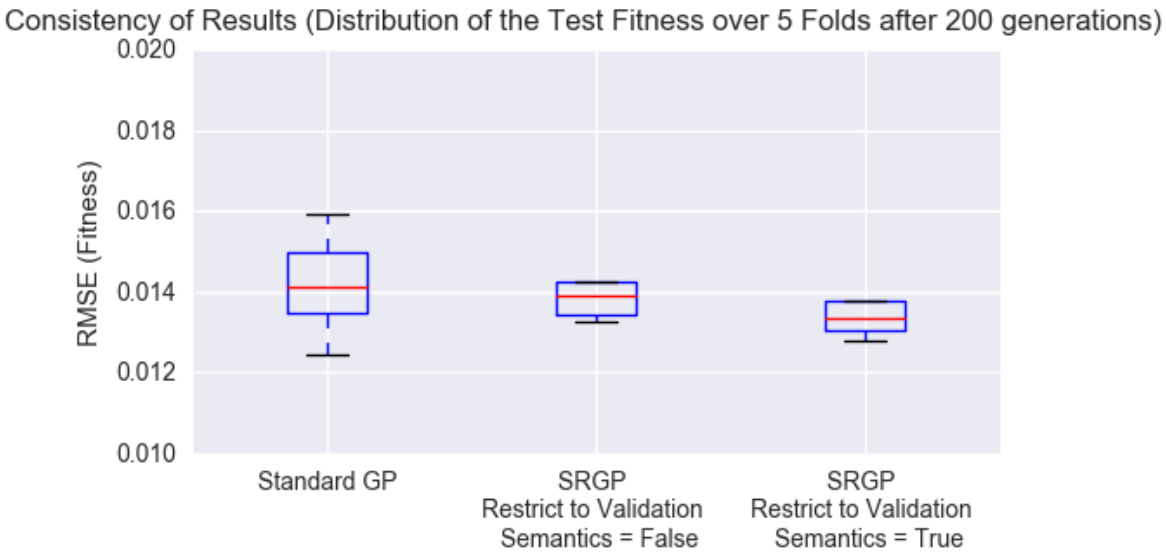

Figure A.14.: Restrict to validation semantics: Consistency after 200 generations

Slope-based Overfitting

The alternative measure of overfitting and the associated severity described in section 3.3, was tested using $5, 10$ and 25 as the number of generations to include in the calculation of the slope. The other parameters have been kept at the settings derived in A.7.

Assessing the behavior over 200 generations, leads to the conclusion, that using the last 10 generations in the slope-based measure results in the best outcome. In comparison to the default overfitting measure, the slope-based approach has a rather similar outcome. One reason for the outcome to be equal, might be the fact, that both mechanisms classify individuals as overfitting with a similar frequency. Figure A.17 depicts the absolute size of the repulser pool over time, as well as the number of individuals added to the pool at every generation. Both methods quickly fill the pool up to its limit and then seem to consistently add individuals at more or less every generation.

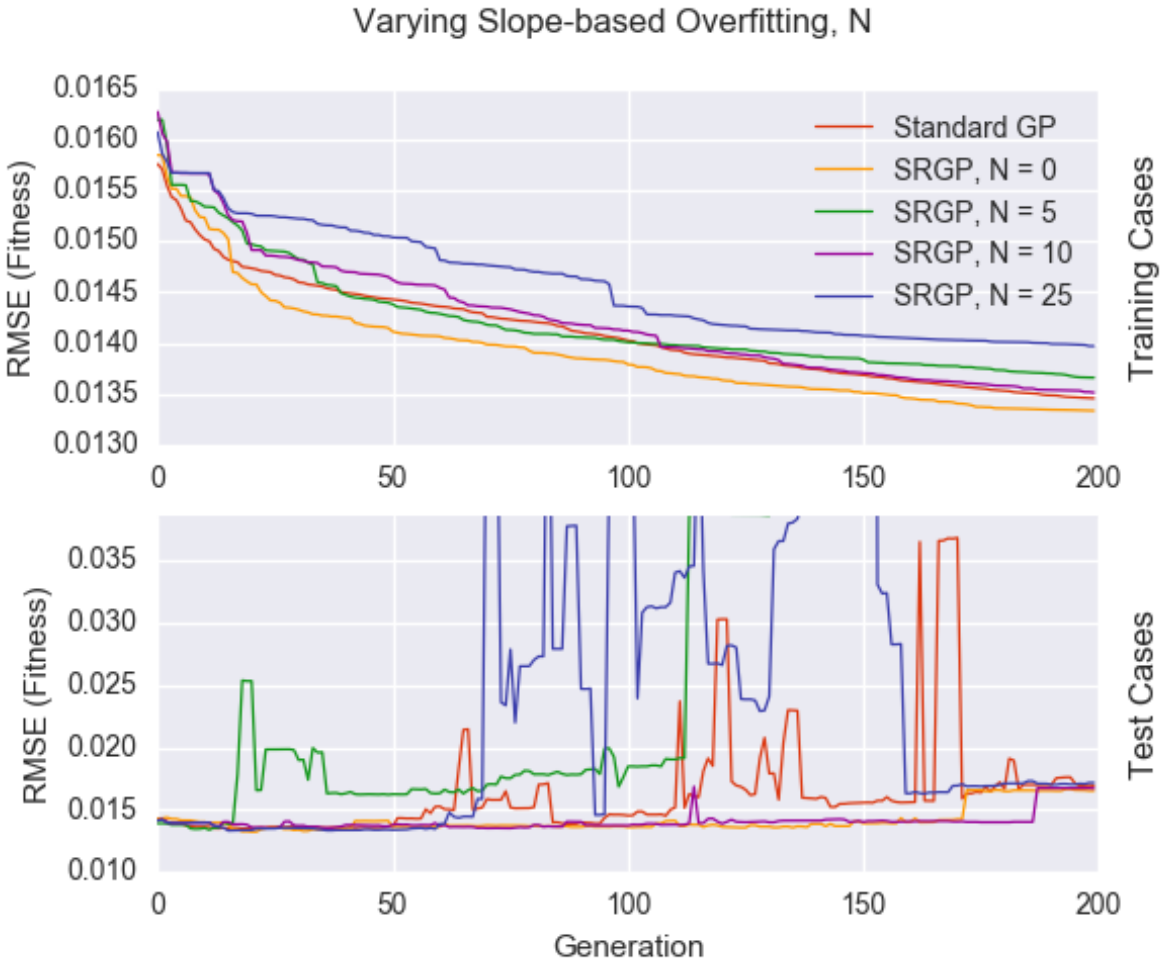

Figure A.15.: Slope-based Overfitting: Training & Test Fitness

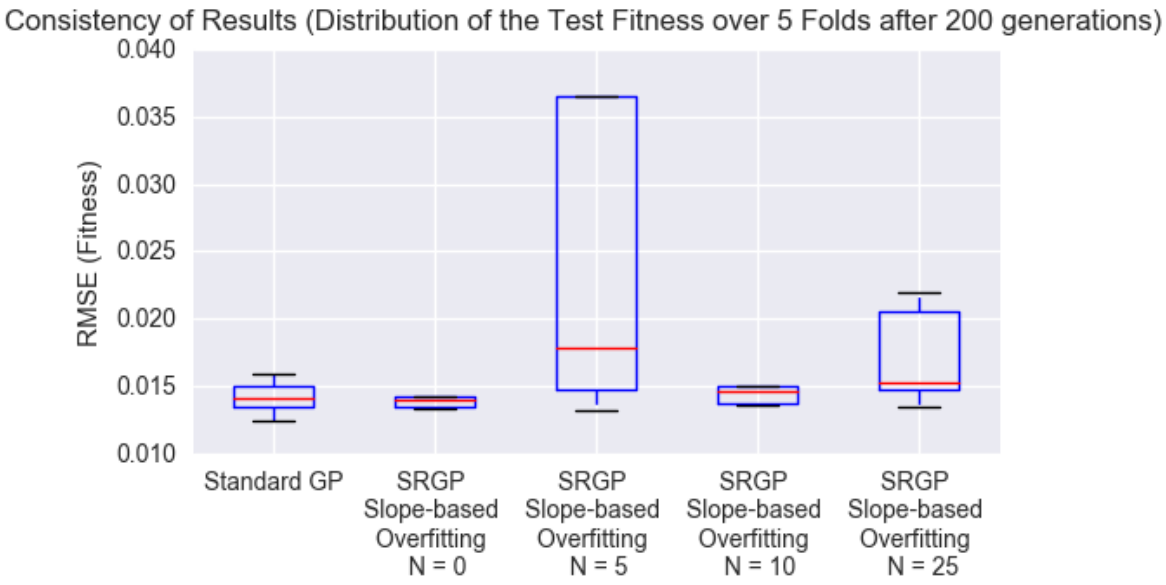

Figure A.16.: Slope-based Overfitting: Consistency after 200 generations

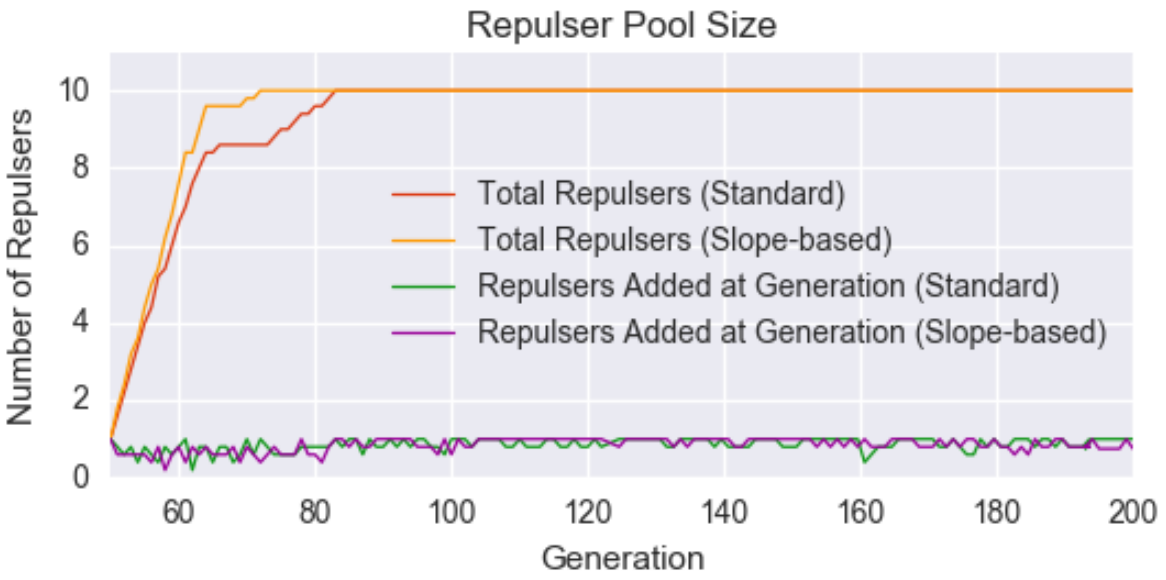

Figure A.17.: Overfitting in SRGP: Number of Repulsers during Evolution

Excluded Fitness Objective

To increase the influence of the distance to repulsers, the system can be modified to ignore the fitness objective during the fast-non-dominated-sort. The following results are based on the same parameters as above (see table A.7) and are comparing the inclusion and exclusion of the fitness as an objective against the baseline standard GP.

Generally speaking, no significant differences can be found, both versions seem to smoothen the behavior of standard GP in a similar manner. As expected the exclusion of the fitness objective has no negative influence on converging towards an optimal solution. On the other hand it does not have a positive impact either.

Using this parameter in conjunction with the aggregation of the semantic distances renders the system single-objective, with the only target being a large average distance to all repulsers. The figure A.20 presents the results of such configuration and interestingly the algorithm is following a steeper learning curve on the training data, yet behaves uncontrollable for anything outside the training observations.

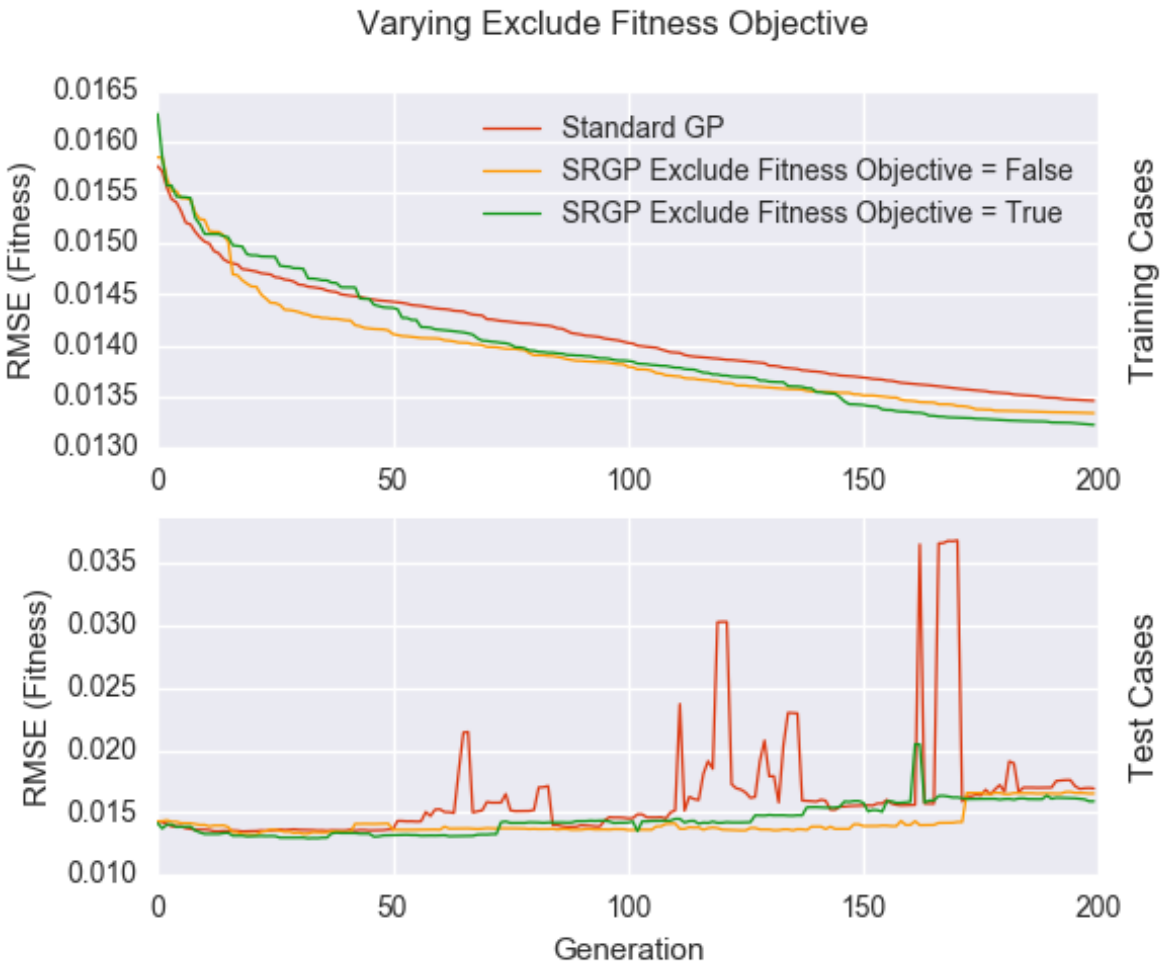

Figure A.18.: Excluded Fitness Objective: Training & Test Fitness

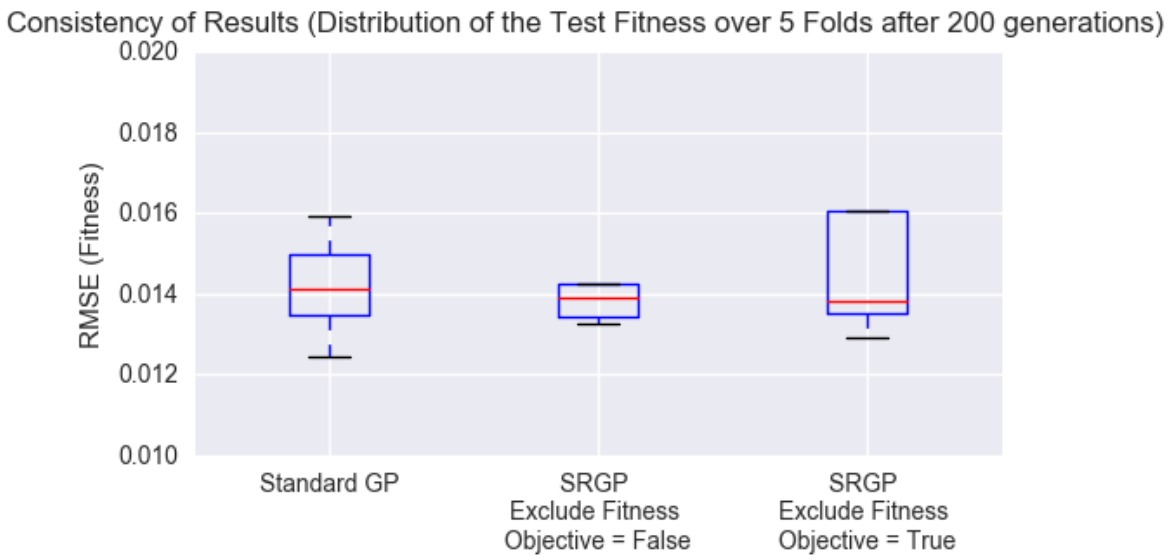

Figure A.19.: Excluded Fitness Objective: Consistency after 200 generations

Figure A.20.: Excluded Fitness Objective & Aggregate Repulsers (Training and Test Fitness)

Force Avoid Repulsers

Forcefully avoiding the creation of individuals similar to the repulser pool has been tested in three configurations. The equality delta parameter has been set to $0.1, 0.01$ and 0.001 in order to have different levels of pressure on the variation phase. As can be seen in the graphs A.21 and A.22 this approach keeps the overfitting down to a minimum and increases the stability of the results after 200 generations. Overall it seems to have comparable outcomes to the best previously found configuration of SRGP.

As the tests have been carried out it became apparent, that the iteration limit mentioned in section 3.3 (the number of trials to create a non-similar individual per generation) has been applied frequently. This is due to the difficulty of GP to create new individuals, lying outside the populations genetic material. As a result any test using this approach runs infeasibly and unpredictably long, thus being not a useful alternative to SRGP in its default configuration.

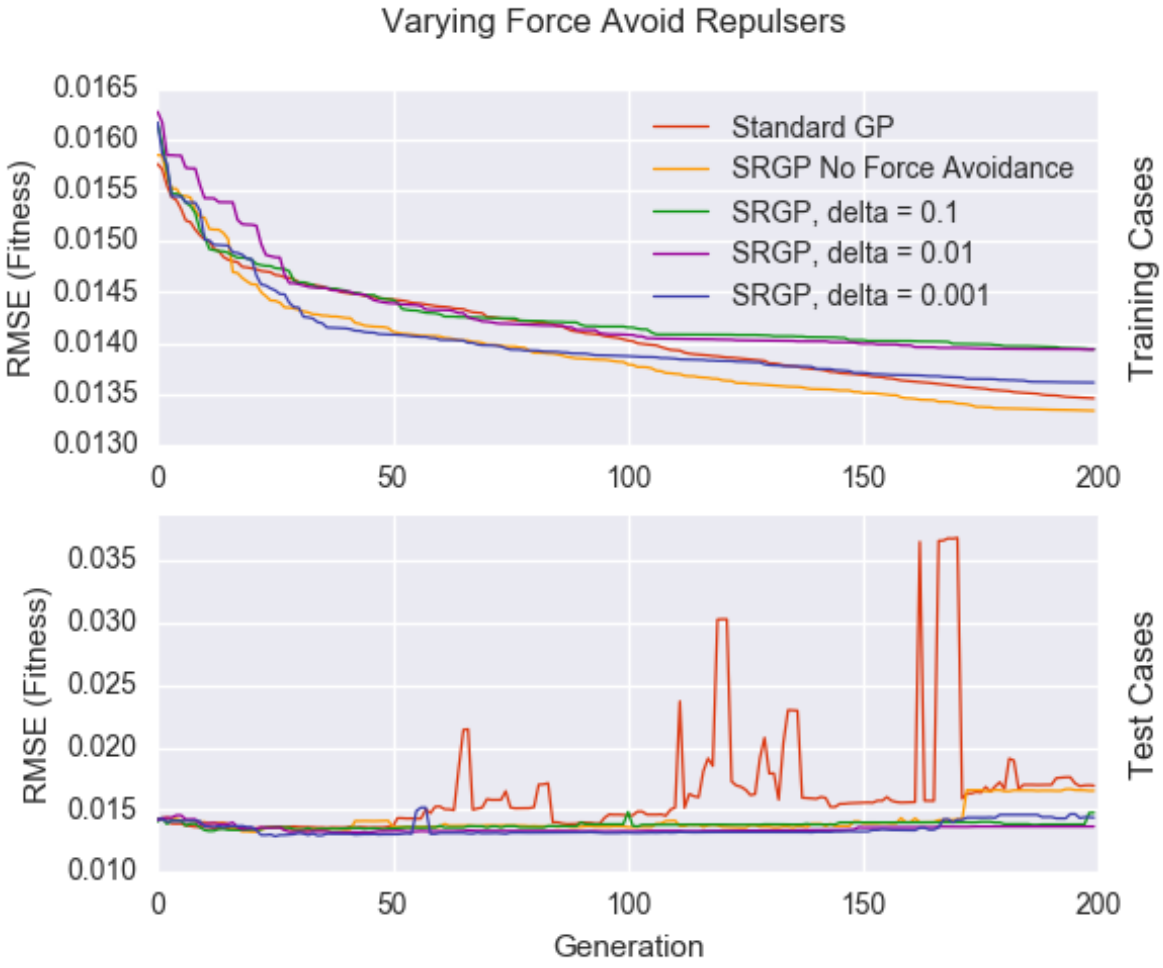

Figure A.21.: Force Avoid Repulsers: Training & Test Fitness

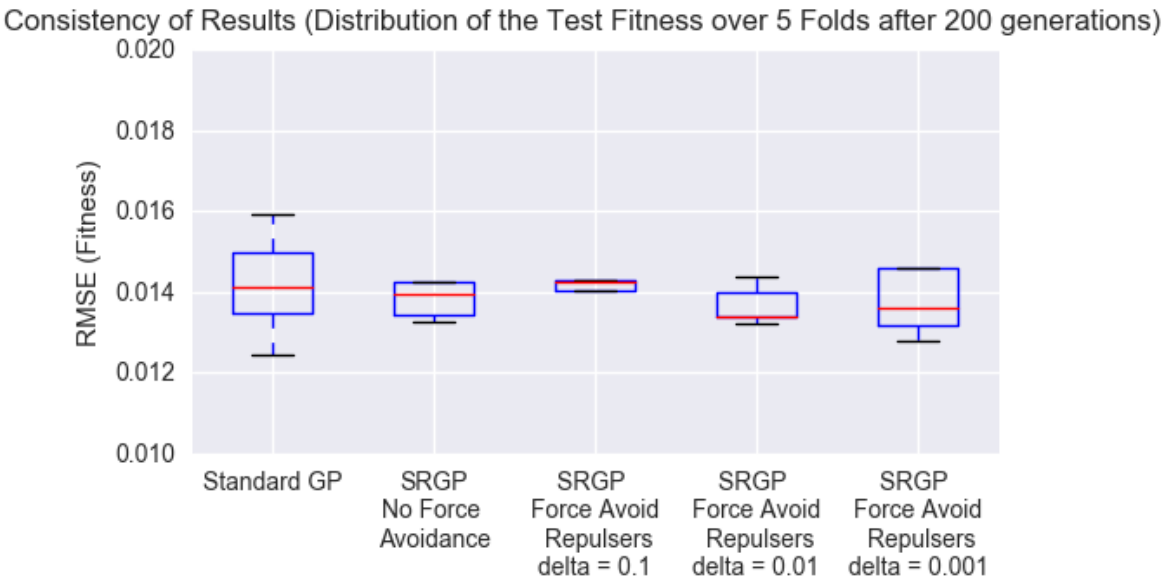

Figure A.22.: Force Avoid Repulsers: Consistency after 200 generations

Additional Tuning

In order to support the data gathered previously several more tests have been carried out using 30 folds per configuration. The following will examine the specific tests and the implications.

As mentioned, excluding the training semantics during the distance calculation for the fast-non-dominated-sort had a positive effect on SRGP. Thus the following plot presents the results using the parameters determined previously (table A.7) in conjunction with only using the validation semantics (it is opposed by the default SRGP configuration). As can be seen, neither of the two approaches introduces significant improvements, besides a slight smoothening of the evolution.

Additionally to support the final claim of the repulsers not impacting significantly, more tests have been executed varying the validation elite pool size, as well as the repulser pool size. The following figure A.25 contains the fitness over the 200 generations using the default configurations (table A.7) and the following three settings:

Validation Elite Pool Size	Repulser Pool Size	Identifier
10	50	SRGP 10/50
50	10	SRGP 50/10
25	25	SRGP 25/25

Table A.8.: Parameters of Final Test for Semantic Repulsers in Genetic Programming

As can be seen in comparison to standard GP, none of the configurations overcomes the problem of overfitting. Standard GP is perfroming better than any of the last three configurations and similarly to the previously tested, best configuration.

Figure A.23.: Additional Tuning: Training & Test Fitness

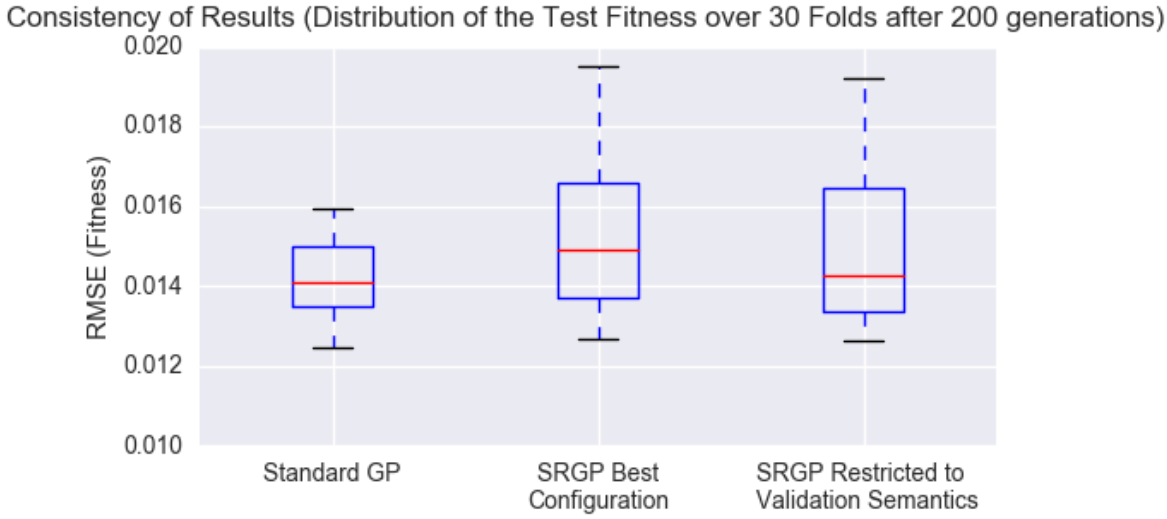

Figure A.24.: Additional Tuning: Consistency after 200 generations

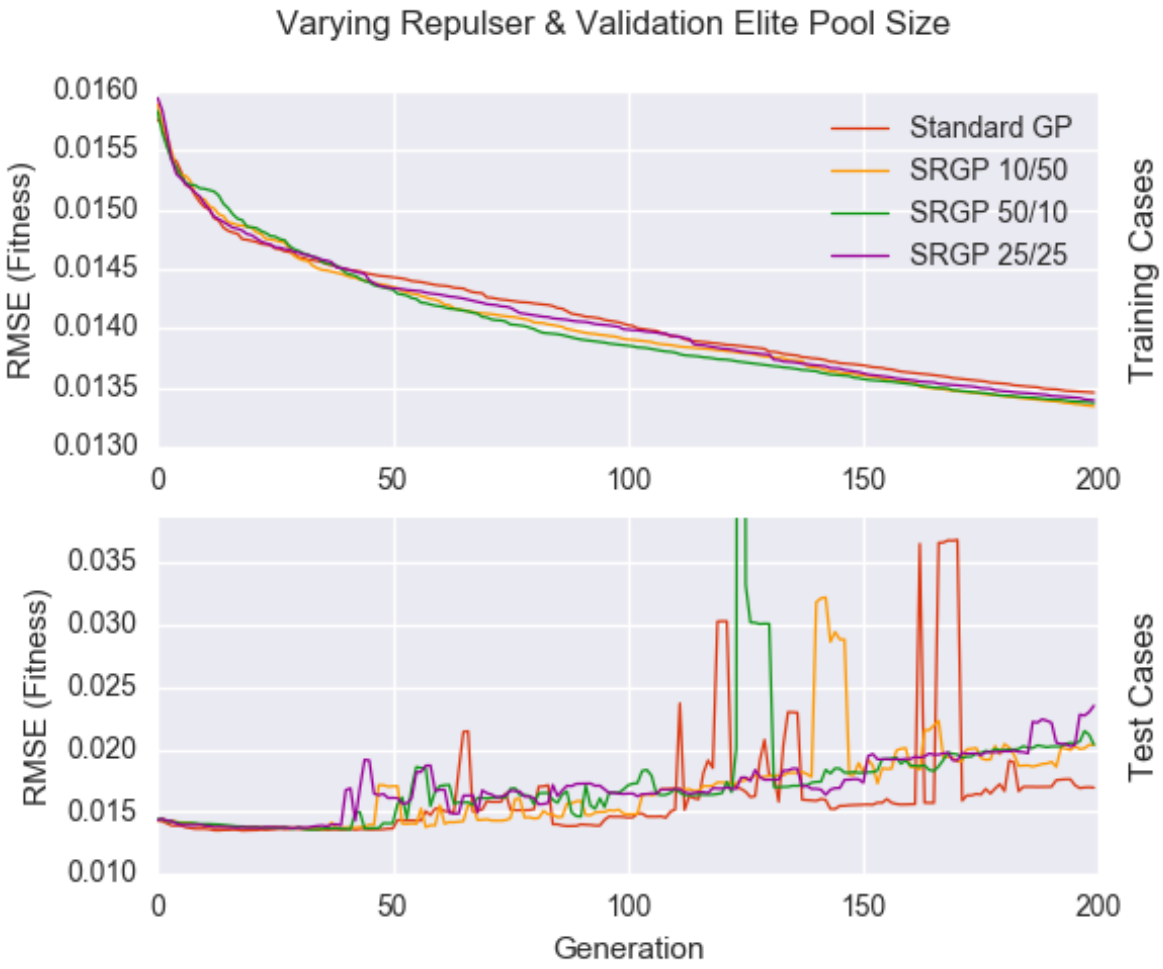

Figure A.25.: Final Test for Semantic Repulsers in Genetic Programming: Training & Test Fitness

A.3 Signature-based Substitute of Validation Dataset

The following table A.9 presents the results of the conceptual tests carried out in order to assess the general possibility of the concept presented in section 4.3. Three different model types, two different dimensionality reduction techniques, as well as two different signature sizes have been used in order to establish a basis for an evaluation of the models performance.

Model	**Dimensionality Reduction**	**# Nodes**	**∅ Training Score**	**∅Test Score**	**Measure**
Binary (over-fitting)	feature-selected	5	0.9963	0.2800	F1
Continous (severity)	feature-selected	5	0.000003	0.0010	MSE
Multi-Class (severity bucket)	feature-selected	5	0.9891	0.3800	F1
Binary (over-fitting)	feature-selected	15	0.9968	0.4900	F1
Continous (severity)	feature-selected	15	0.000006	0.0030	MSE
Multi-Class (severity bucket)	feature-selected	15	0.9832	0.0200	F1
Binary (over-fitting)	depth-truncated	3	0.9817	0.5500	F1
Continous (severity)	depth-truncated	3	0.0000001	0.00006	MSE
Multi-Class (severity bucket)	depth-truncated	3	0.9259	0.2900	F1
Binary (over-fitting)	depth-truncated	7	0.9668	0.5400	F1
Continous (severity)	depth-truncated	7	0.0000005	0.000005	MSE
Multi-Class (severity bucket)	depth-truncated	7	0.8534	0	F1

Table A.9.: Performance of overfitting/severity classifiers in training and test environment (5 folds, see measure column for score meaning)

GLOSSARY

algorithm A sequence of (computational) instructions for solving a well-defined problem by transforming input into output [Cor09, see p. 5].

decision tree A model using trees made of data separating decision in order to perform classifications [Mit97, see p. 52ff.].

euclidean distance Euclidean distance refers to the pythagorean theorem applied to p dimensions and corresponds to $ED_{i,h} = \sqrt{\sum_{j=1}^{p}(a_{i,j} - a_{h,j})^2}$, where a is a component of the vectors to be compared [MGU02, see p. 46].

gradient boosting regressor An algorithm deriving a solution by minimizing a loss function using gradient descent [Fri01].

MLP A multi-layer perceptron is an implementation of an artificial neural network using the backpropagation algorithm. The model is based of several connceted layers of neurons that subsequently transform the input into an output vector/value [Mit97, see p. 81f.].

NP Complexity class containing all problem solvable in polynomial time on a non-deterministic Turing machine; The class contains all problems (P and NP-complete) [Kok05, see p. 4].

NP-complete Complexity class containing all problems solvable in polynomial time on a non-deterministic Turing machine to which all other problems can be reduced to [Kok05, see p. 4].

NP-hard Complexity class containing all NP-complete problems and all problems for which no algorithm exists [Kok05, see p. 4].

OLS A generalized linear model using the least squares estimation method [NW72, see p. 370ff.].

P Complexity class containing all problems solvable in polynomial time on a deterministic Turing machine [GGL95, see p. 1609];[Kok05, see p. 4].

pareto hierarchie In a multi-objective setting a hypothesis is only better than another one, if it exceeds the other one in all objectives (domination). The best n solutions are referred to as pareto optima solutions [CD08, see p. 12f.] Applying this concept recursively to all hypotheses a hierarchie of domination can be derived in order to compare hypothesis among each other.

random forest An ensemble model made of multiple decision trees returning the majority vote of the trees as a predictions result [Bre01, see p. 5f.].

SVM A classification algorithm mapping the input data into a very-high dimensionality space in order to construct a separating hyperplane [CV95, see p. 273f.].

tree A graph consists of a set of vertices, which form nodes and edges [Die06, see p. 2]. An acyclic, conected arrangement of the vertices in a graph, creates a *tree*. Trees consist of branches, whose ending vertices (of degree 1) are referred to as leaves [Die06, see p. 13f.].

BIBLIOGRAPHY

[AK89] Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. New York, NY, USA: John Wiley & Sons, Inc., 1989.

[Als09] Nada MA Al-salami. "Evolutionary Algorithm Definition". In: *Am. J. Eng. Appl. Sci* 2.4 (2009), pp. 789–795.

[Bli97] T. Blickle. *Theory of Evolutionary Algorithms and Application to System Synthesis*. TIK-Schriftenreihe. Hochschulverlag AG an der ETH, 1997.

[Bre01] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[CD08] Massimiliano Caramia and Paolo Dell'Olmo. *Multi-objective Management in Freight Logistics*. Springer-Verlag London, 2008.

[Cor09] T.H. Cormen. *Introduction to Algorithms*. MIT Press, 2009.

[CV95] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning* 20.3 (1995), pp. 273–297.

[Deb+02] K. Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002), pp. 182–197.

[Die06] R. Diestel. *Graph Theory*. Electronic library of mathematics. Springer, 2006.

[Die95] Tom Dietterich. "Overfitting and Undercomputing in Machine Learning". In: *Computing Surveys* 27 (1995), pp. 326–327.

[Fri01] Jerome H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232.

[Gar16] Kasey Gartner Inc. Contributer: Panetta. *Gartner's Top 10 Strategic Technology Trends for 2017*. 2016. URL: `http://www.gartner.com/smarterwithgartner/gartners-top-10-technology-trends-2017/` (visited on 04/20/2017).

[GGL95] R. L. Graham, M. Grötschel, and L. Lovász, eds. *Handbook of Combinatorics (Vol. 2)*. Cambridge, MA, USA: MIT Press, 1995.

[Haw04] Douglas M. Hawkins. “The Problem of Overfitting”. In: *Journal of Chemical Information and Computer Sciences* 44.1 (2004), pp. 1–12.

[KM14] Y. Kodratoff and R.S. Michalski. *Machine Learning: An Artificial Intelligence Approach*. v. 3. Elsevier Science, 2014.

[KO14] Krzysztof Krawiec and Una-May O’Reilly. “Behavioral Programming: A Broader and More Detailed Take on Semantic GP”. In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. GECCO ’14. Vancouver, BC, Canada: ACM, 2014, pp. 935–942.

[Kok05] Natallia Kokash. “An introduction to heuristic algorithms”. In: *Department of Informatics and Telecommunications* (2005), pp. 1–8.

[Koz92] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. A Bradford book. Bradford, 1992.

[Lju01] L. Ljung. “Black-box models from input-output measurements”. In: *IMTC 2001. Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference. Rediscovering Measurement in the Age of Informatics (Cat. No.01CH 37188)*. Vol. 1. May 2001, 138–146 vol.1.

[MGU02] Bruce McCune, James B. Grace, and Dean L. Urban. *Analysis of ecological communities*. MjM Software Design, 2002.

[Mit97] Tom M. Mitchell. *Machine Learning*. Mcgraw-Hill Series in Computer Science. McGraw-Hill Science/Engineering/Math, 1997.

[MKJ12] Alberto Moraglio, Krzysztof Krawiec, and Colin G. Johnson. “Geometric Semantic Genetic Programming”. In: *Parallel Problem Solving from Nature - PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I*. Ed. by Carlos A. Coello Coello et al. Springer Berlin Heidelberg, 2012, pp. 21–31.

[NW72] J. A. Nelder and R. W. M. Wedderburn. “Generalized Linear Models”. In: *Journal of the Royal Statistical Society* 135.3 (1972), pp. 370–384.

[PS13] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Books on Computer Science. Dover Publications, 2013.

[RE76] P. Robert and Y. Escoufier. “A unifying tool for linear multivariate statistical methods: The RV-coefficient”. In: *Applied Statistics* 25 (1976), pp. 257–265.

[Smi+09] Age K. Smilde et al. “Matrix correlations for high-dimensional data: the modified RV-coefficient.” In: *Bioinformatics* 25.3 (June 22, 2009), pp. 401–405.

[SRB07] Gábor J. Székely, Maria L. Rizzo, and Nail K. Bakirov. “Measuring and testing dependence by correlation of distances”. In: *Ann. Statist.* 35.6 (Dec. 2007), pp. 2769–2794.

[WM97] D. H. Wolpert and W. G. Macready. “No Free Lunch Theorems for Optimization”. In: *Trans. Evol. Comp* 1.1 (1997), pp. 67–82.

[YG10] X. Yu and M. Gen. *Introduction to Evolutionary Algorithms*. Decision Engineering. Springer London, 2010.