

PARAMETER CALIBRATION USING META-ALGORITHMS

Wouter Alexander de Landgraaf

Master thesis Artificial Intelligence  
Vrije Universiteit, Amsterdam

Supervisor:  
A. E. Eiben

Assistent supervisor:  
V. Nannen

November 16, 2006

## Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Autonomous Selection . . . . .	6
1.2	Why Autonomous Selection? . . . . .	6
1.2.1	Scaling Genetic Algorithms . . . . .	6
1.2.2	Biological realism . . . . .	8
1.2.3	Spatial structure . . . . .	8
1.3	Drawbacks of Autonomous Selection . . . . .	9
1.4	Calibration and Meta-Evolutionary Algorithms . . . . .	9
1.5	Summary of Research Questions . . . . .	10
1.6	Overview of the approach . . . . .	10
1.7	Structure of this thesis . . . . .	11
<b>2</b>	<b>History of Evolutionary Computing</b>	<b>12</b>
2.1	Evolutionary Algorithms in general . . . . .	12
2.2	Genetic Algorithms . . . . .	12
2.3	Evolution Strategies . . . . .	13
2.4	Evolutionary Programming . . . . .	14
2.5	Genetic Programming . . . . .	15
2.6	Simple Genetic Algorithm . . . . .	16
2.6.1	Representation . . . . .	16
2.6.2	Mutation . . . . .	16
2.6.3	Crossover . . . . .	17
2.7	Selection in Genetic Algorithms . . . . .	17
2.7.1	Best Selection . . . . .	17
2.7.2	Fitness-proportional Selection . . . . .	18
2.7.3	Rank-based Selection . . . . .	18
2.7.4	Tournament Selection . . . . .	19
2.7.5	Age-Based Survival Selection . . . . .	19
2.7.6	Elitist Survival Selection . . . . .	19
2.8	Autonomous Selection Genetic Algorithm . . . . .	20
2.8.1	Algorithm Description . . . . .	20
2.9	Distributed Genetic Algorithms . . . . .	20
<b>3</b>	<b>History of Calibration and Meta-algorithms</b>	<b>22</b>
3.1	Parameter Calibration of Genetic Algorithms . . . . .	22
3.2	Adaptation and Parameter control . . . . .	22
3.3	Meta Genetic Algorithms . . . . .	23
3.3.1	The “Grefenstette” Meta-GA . . . . .	24
3.4	Relevance Estimation and Value Calibration . . . . .	24
3.4.1	Estimation of Distribution Algorithms . . . . .	25
3.4.2	Workings of REVAC . . . . .	26

<b>4</b>	<b>Problem description</b>	<b>28</b>
4.1	Research Questions: Autonomous Selection . . . . .	28
4.2	Research Questions: Meta-Evolutionary Algorithms . . . . .	28
4.3	Hypotheses . . . . .	29
4.3.1	Autonomous selection calibration . . . . .	29
4.3.2	Sensitivity of parameters . . . . .	30
4.3.3	Performance . . . . .	30
4.3.4	Comparable parameters via meta-evolutionary algorithms	31
4.3.5	Meta-evolutionary and simple genetic algorithms . . . . .	31
4.3.6	Applicability . . . . .	31
4.4	Summary of Hypotheses . . . . .	32
<b>5</b>	<b>Experimental Design</b>	<b>33</b>
5.1	Experimental Design: objectives . . . . .	33
5.1.1	Optimality . . . . .	33
5.1.2	Robustness . . . . .	33
5.1.3	Statistical Significance . . . . .	34
5.1.4	No Free Lunch? . . . . .	35
5.2	Problem instances . . . . .	36
5.2.1	Spears multi-modal problem generator . . . . .	36
5.3	Screening: Manual calibration and framework development . . .	37
5.3.1	Determining search space and granularity . . . . .	37
5.4	Verification . . . . .	38
5.4.1	Verifying Autonomous Selection results . . . . .	38
5.4.2	ASGA Verification Plots . . . . .	39
5.4.3	Verifying Simple Genetic Algorithm results . . . . .	44
<b>6</b>	<b>Experiments</b>	<b>46</b>
6.1	Optimization: Autonomous Selection Calibration . . . . .	46
6.1.1	Calibration using Meta Genetic Algorithm . . . . .	46
6.1.2	Calibration using REVAC . . . . .	49
6.1.3	ASGA Calibration Plots, Spears 1 Peak . . . . .	51
6.1.4	Calibration using REVAC, second parameter combination	63
6.1.5	7-Parameter ASGA Calibration using Meta-GA . . . . .	64
6.1.6	7-Parameter ASGA Calibration using REVAC . . . . .	65
6.1.7	7-Parameter ASGA Calibration Plots . . . . .	67
6.2	Optimization: Calibrating SGA . . . . .	69
6.2.1	Calibration using Meta Genetic Algorithm . . . . .	69
6.2.2	Calibration using REVAC . . . . .	70
6.2.3	SGA Calibration Plots . . . . .	72
6.3	Robustness: ASGA and Stability . . . . .	75
6.4	Robustness: SGA and Stability . . . . .	79

---

<b>7</b>	<b>Analysis &amp; Discussion</b>	<b>82</b>
7.1	Comparison between meta-algorithms on ASGA . . . . .	82
7.1.1	Diversity of Population . . . . .	82
7.1.2	Statistical Significance . . . . .	82
7.2	Simple GA analysis . . . . .	85
7.2.1	Statistical Significance . . . . .	86
7.3	Ease of applying methods compared to trial and error . . . . .	87
7.4	Quality of the autonomous selection solutions discovered . . . . .	87
7.5	Stability of results . . . . .	88
7.6	Comparable performance? . . . . .	88
<b>8</b>	<b>Conclusion</b>	<b>90</b>
8.1	Future Work . . . . .	91
<b>A</b>	<b>GAMP</b>	<b>92</b>
A.1	GAMP Overview . . . . .	92
A.2	Features . . . . .	92
A.2.1	Representation . . . . .	92
A.2.2	Recombination . . . . .	92
A.2.3	Mutation . . . . .	92
A.2.4	Parent selection . . . . .	93
A.2.5	Survival selection . . . . .	93
A.2.6	And more... . . . .	93
A.3	Conclusion . . . . .	94

## Preface

The other day I heard a short speech, in where a manager described the process of obtaining a university degree. The first part is knowledge, symbolized by the books and theoretical courses one needs to follow. The second part is ability, the ability to apply that knowledge to new problems in practical courses and exams. The third part is art, the art of turning that ability and knowledge into a masterwork with scientific relevance, a cumulation of months of dedication into a single task. That task came up during a discussion with Guszti quite a few months ago.

I have always been fascinated by evolution. The idea of trial and error, natural selection, being able to produce living, breathing, thinking creatures is truly mind-boggling. In the past I have enthusiastically taken part in courses on evolutionary computing and self-organization. A few years ago I completed a scheduling genetic algorithm for Martijn Schut, so I have seen the possibilities first-hand. For my master project however I wanted to dive into a theoretical problem and build a solution using genetic algorithms.

The project Guszti and I came up with fits perfectly into that category. Guszti was in the process of finishing the first tests of autonomous selection, however the results weren't very satisfying. After months of manual trial and error, we agreed to let evolution handle the task of calibrating this difficult problem, thus opening the door to meta-evolutionary algorithms. Two were selected: the Meta-Genetic Algorithm and the REVAC method. The latter was recently used by Volker for calibrating an economic simulation. We quickly found common ground and the project was underway.

I would like to thank first of all my supervisors, without all those discussions and their support I probably would have gone in a completely different direction, and looking at the final result I am glad I didn't. I would also like to thank my mother, my friends and my girlfriend Lisanne, who kept me on track and helped motivate me during the last few months. Thanks goes out to those who helped create the tools I used for the creation of this thesis. I couldn't imagine completing it without the help of L<sup>A</sup>T<sub>E</sub>X, Python, Emacs and Gnuplot.

This thesis is available under the GNU Free Documentation License (<http://www.gnu.org>) from my personal website (<http://www.alextrreme.org>).

## 1 Introduction

When researchers use an algorithm, typically little CPU-time is invested in the calibration of the algorithm. Instead, researchers either opt for “best-practice” parameters which have previously been used for similar applications or researchers invest a lot of their own time in manually calibrating the parameters. Both methods are clearly sub-optimal: small changes in requirements can lead to completely different results and manually tweaking the parameters of an algorithm is very costly, as a typical search space which results from the algorithm parameters becomes exponentially larger for each additional parameter.

The algorithms we will look at in this thesis are all related to evolutionary algorithms. These are algorithms inspired by the “trial-and-error” and natural selection seen in evolution. Using mutation, recombination and selection these algorithms try to find the optimal solution for a given problem. An in-depth look at evolutionary algorithms can be found in the next chapter.

### 1.1 Autonomous Selection

The need for automating the calibration of parameters of an algorithm comes from the development of a new type of selection for genetic algorithms, namely **autonomous selection** [6]. The properties of this algorithm make autonomous selection a preferable choice when it comes to scaling genetic algorithms in a distributed system over large numbers of computers. Instead of having individuals be selected via a tournament-type comparison they determine if they should survive or reproduce by comparing their own fitness with that of a number of their peers.

### 1.2 Why Autonomous Selection?

But why exactly is autonomous selection interesting? There are a number of cases where using autonomous selection is appropriate, compared to traditional types of selection.

#### 1.2.1 Scaling Genetic Algorithms

When scaling genetic algorithms (GAs) there is a large advantage in being able to run, performance-wise costly, evaluations on multiple machines. However, with typical genetic algorithms the results of all these evaluations have to be combined centrally in order to determine if the local individuals are allowed to survive into the next generation. A major advantage of autonomous selection is that there doesn't have to be a central authority that determines if an individual dies: the individual compares its own fitness to that of a number of other individuals, ranging up to the whole of the population on the current machine. The individual determines if it is allowed to survive into the next generation. As such, a population spread over multiple nodes does not have to communicate between nodes. Each subpopulation can evolve locally. When using autonomous

selection, selection isn't based on the fitness of all individuals system-wide, but the individuals instead are compared only to the average fitness of a certain subpopulation. This average can be estimated by obtaining a limited number of fitness levels from a random sample of the subpopulation.

There are of course other forms of selection that make scaling genetic algorithms easier. With tournament selection, a certain number of individuals are ranked according to their fitness. The rank of each individual determines the chance of surviving or reproducing. Autonomous selection differs from this approach in that it works only in one direction: with autonomous selection an individual compares its fitness with that of its peers, however the peers aren't aware of this comparison nor are affected by it. In a tournament only the best survive, however with autonomous selection the individual obtains the average fitness of its peers and then determines how far from this average its own fitness lies. This information it uses to determine its chances for reproducing and surviving. This greatly simplifies large-scale distributed genetic algorithms, especially those where it is difficult or impossible to distribute data between the different nodes, such as is the case with massively distributed projects.

A few years ago projects started to spring up that utilized spare CPU cycles on thousands of machines connected via the Internet. The first project that was distributed this way was Seti@Home.

The SETI project's goal was the search for extra-terrestrial intelligence and to accomplish this a lot of data was collected via radio-telescopes that continuously searched the skies for signs of life. However, as there is a lot of background noise generated the SETI project had to process the collected data to filter out the most promising pieces of data. The only problem was the vast amount of processing power required to accomplish this goal. Seti@Home was then started to allow everyone to participate by having their computers filter through the data when they would otherwise remain idle. The idea to use your desktop computer to search for extra-terrestrial life hooked the imaginations of many. Contests were started, people banded together and the SETI project had a massively distributed computer that rivaled supercomputers virtually for free.

Now the nature of Seti@Home, in spite of the massive amounts of data, was simple enough to be distributed easily. Each computing node received an amount of raw data and that node returned the analyzed results after it was done. The SETI team then only had to look at the filtered results, in the hope of finding something that stuck out from the rest. Many other projects were started up to solve problems that were as easy split up into subproblems.

For genetic algorithms and evolutionary computing in general there always remained the need to send single fitness evaluations between the distributed nodes. This limits the total size that the computations can be distributed. One example is the Newties project. The Newties project was started in 2005 and is a collaboration of a number of universities throughout Europe and is headed by the Vrije Universiteit in Amsterdam. The goals of the project are to simulate

evolution in such a way that entities become capable of learning, start using language and eventually form simple societies.

Currently the Newties project is still in the development process, however from the start it was clear that a vast amount of processing power would be required to run the simulations in order to accomplish the above goals. As large supercomputers are out of reach for the universities working on the project, it is clear that having the simulation be distributed over a massive amount of computing nodes is much more likely to give good results. We will not look into the details of the Newties project, for our purposes we will assume it can be simplified into a simple genetic algorithm. As stated above, the simple genetic algorithm has too many performance drawbacks to use in a massively distributed project, thus the use of autonomous selection is then to be preferred. Subpopulations could be spread over multiple nodes with only the average fitness sent between the nodes, while the subpopulations themselves only have to communicate their final results back to the main project server. This leads to a very flexible design and ensures that the computation required can be split over a vast number of nodes that are physically far apart, yet these nodes wouldn't have to wait for each other or communicate with each other on a regular basis if they weren't part of the same subpopulation. The nodes that would be part of the same subpopulation could be determined in a peer-to-peer manner using statistics like latency and reliability, however provisions would be required in the case that one of the nodes were to be disconnected. If the future distributed Newties project were to be set up in such a manner, then scaling the project would be vastly simplified compared to setting up the project by means of a traditional simple genetic algorithm.

### 1.2.2 Biological realism

Another advantage of autonomous selection is the increased realism of the simulation. When looking at Artificial Life simulations, or even real-world biological systems, it is unrealistic to assume that an individual compares its fitness with the whole of the population, or even a random subset. An individual would "look" locally, at the own subpopulation where it lives in, and determine how well it performs in comparison to its neighbors. In the real world an individual won't decide to spontaneously reproduce or commit suicide solely based on the fitness of its neighbors, yet outward signs of fitness exist for the purpose of being detected by both predators and peers.

### 1.2.3 Spatial structure

With autonomous selection, subpopulations are also able to evolve in more-or-less separate clusters of individuals, comparable to how populations evolved on distant continents and islands. Populations in the Americas for example evolved differently from those in Eurasia or Africa, while remote islands like the Galapagos Islands were the scene for many remarkable differences in flora and fauna. All in all, it helps in keeping an amount of diversity in the simulation

that would otherwise be flattened-out.

### 1.3 Drawbacks of Autonomous Selection

However autonomous selection has one major drawback: the population size can no longer remain constant. It is therefore required to find parameters that cause the population to remain relatively constant (ie. the population doesn't go extinct or explode in size) while there still is enough selection that better individuals have a far larger chance to survive and reproduce than their peers that don't have an acceptable fitness.

This in itself wouldn't have been a problem, except that these parameters in practice are very hard to find. Even though certain parameters do work reasonably well, how do we know that there aren't better parameters just around the corner? Following this reasoning, a period of 3 months was invested by the authors of [6] in coming up with efficient parameters. All in all, although the authors could cry victory, it is a sour one. Many man-hours were invested, yet the performance is still worse compared with a simple genetic algorithm.

The main problem with this algorithm is thus that calibrating the parameters in order to get a stable population is very hard, as stated in [6]. The challenge is thus not to only find parameters where the population remains fairly stable, but also to ensure for settings that lead to a fitness that is maximized while the number of fitness evaluations is minimized. Could autonomous selection be similar in efficiency compared to other forms of selection?

### 1.4 Calibration and Meta-Evolutionary Algorithms

Two approaches to solve the problems described above exist: either continue manual calibration (in the hope of finding better parameters that are much more efficient), or use automated techniques in order to discover better parameters. We believe that the authors of [6] did try very hard to find parameters that both perform well and maintain a stable population size, and the prospect of investing another 3 months of manual calibration doesn't sound very appealing. Instead, we will look at the use of automated means of parameter calibration, which we expect take less time and come up with better results.

A new problem arises, however. There are many methods for automated calibration, which method would give us the most optimal parameters? Unfortunately this question too remains unanswered. As we are looking at genetic algorithms, the approach in order to come up with a solution seems logical: Why not use an evolutionary algorithm to tune the genetic algorithm? The second topic then becomes finding and applying those so-called meta-evolutionary algorithms.

Two of these algorithms have been chosen. The first is the meta-genetic algorithm, which works like a simple genetic algorithm. The difference is that instead of optimizing a function, the meta-GA is optimizing a genetic algorithm, which in turn optimizes a function. The second choice is the REVAC-method

[23], an algorithm developed at the Vrije Universiteit which has previously been used, with success, for solving a similar problem.

## 1.5 Summary of Research Questions

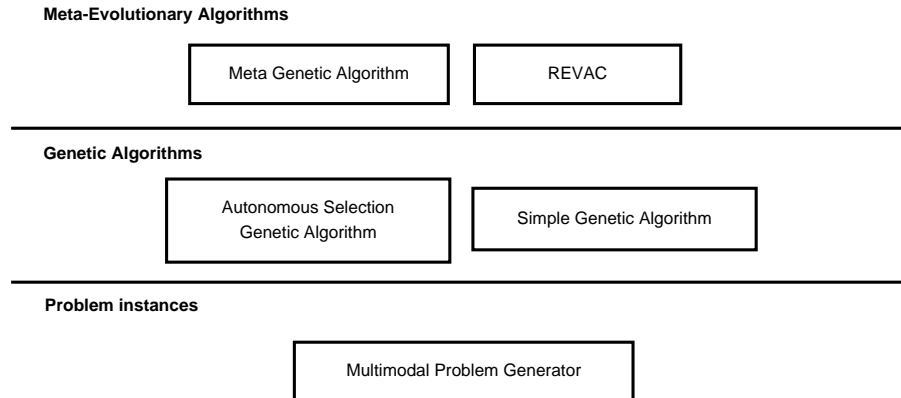
In the course of this thesis, the research questions in Table 1 will be answered.

Table 1: Summary of Research Questions

Research Questions	
1a	<b>Calibration</b> - Are there autonomous selection settings that perform better than in [6]?
1b	<b>Sensitivity and Stability</b> - How sensitive are the autonomous selection parameters?
1c	<b>Improvement</b> - How does a well-tuned autonomous selection genetic algorithm compare to a well-tuned simple genetic algorithm?
2a	<b>Comparing algorithms</b> - How well do different meta-evolutionary algorithms perform when calibrating autonomous selection?
2b	<b>Simple GA</b> - How well do different meta-evolutionary algorithms perform when calibrating a simple genetic algorithm?
2c	<b>Applicability</b> - How useful are meta-evolutionary algorithms?

## 1.6 Overview of the approach

From the questions to be answered in Table 1, it is clear that there are a number of different algorithms required. These algorithms will be explained extensively in the next sections, but for the clarity of the approach we will briefly describe the hierarchy of the algorithms and how they interact. The following model shows how the algorithms are organized.



- The lowest layer contains the problem landscapes or problem instances. The components in this layer grade how well a certain solution is, com-

pared to an unknown optimum state. We will discuss these shortly in **Chapter 5: Experimental Design**.

- The middle layer of algorithms are the algorithms which provide the lower-level algorithms with possible solutions. This layer gets a grade, a degree of fitness, for such a solution. Using this fitness it attempts to define better solutions. We will look at the algorithms used here in **Chapter 2: History of Evolutionary Computing**.
- The upper layer of algorithms are the meta algorithms. These algorithms in turn calibrate the middle-level algorithms by running each algorithm with a range of parameters. Similar to the connection between the lowest layer and the middle layer, each range of parameters provides the upper layer algorithm with a degree of fitness, which the upper layer algorithm uses to further calibrate the middle layer algorithm. These algorithms are discussed in **Chapter 3: History of Calibration and Meta-algorithms**.

## 1.7 Structure of this thesis

- **Chapter 2: History of Evolutionary Computing** - In order to get the reader acquainted with evolutionary computing and a number of the topics discussed in this thesis, this chapter provides a short history of the field of evolutionary computing. We also look at different types of selection, including autonomous selection
- **Chapter 3: History of Calibration and Meta-algorithms** - The field of parameter calibration and meta-algorithms is a highly dynamic one; In this chapter the reader will find a short background of these topics and descriptions of the meta-algorithms used
- **Chapter 4: Problem description** - In this chapter the research questions and hypotheses this thesis addresses are laid out
- **Chapter 5: Experimental Design** - We address the approach we take to support or reject our hypotheses
- **Chapter 6: Experiments** - The hypotheses in chapter 4 are examined one-by-one; We conduct experiments to determine the validity of our hypotheses
- **Chapter 7: Analysis** - Using the results gathered through experimentation we will analyze if our hypotheses are valid
- **Chapter 8: Conclusion** - We look back on this introduction, giving the answers to our questions. We also take a look at possible future research

## 2 History of Evolutionary Computing

This chapter contains a brief introduction to evolutionary computing and the various disciplines over its history.

For over 40 years, many researchers have been adapting techniques from evolutionary biology and using them as simulations and problem-solvers. Roughly, the field of Evolutionary Computing can be subdivided into 4 different categories, each with a different background. Each use different ideas and techniques for designing an algorithm that uses Darwinian-inspired trial-and-error and survival of the fittest in order to solve a wide variety of problems. Although in the rest of this thesis we will look primarily at genetic algorithms, a short introduction of the other 3 types of Evolutionary Algorithms (EA) is useful; it introduces a number of features that started off as being unique to each type of algorithm but have since been included by various types of genetic algorithms.

### 2.1 Evolutionary Algorithms in general

A typical evolutionary algorithm [31] consists of a **population**  $P$  of instances, where each instance is a possible solution to a certain problem. A solution instance is called a **chromosome**, with every single unit of each chromosome being one **gene**. A possible value of a gene is called an **allele**. Each instance can be graded by a **fitness function**, which is an indication of how well a certain solution is. Each generation  $t$  is evolved into generation  $t + 1$  by use of the fitness of each instance together with **crossover**, **mutation** and **selection**. The population size  $P$  normally remains constant. Initialization of the population tends to be random. The fitness function can be trivial or very complex, but in most evolutionary algorithms the fitness function tends to cost the most processor time. Selection is normally split into parent and survival selection, with the former selecting which parents are allowed to create offspring and the latter determining which instances of the population are allowed to be part of the next generation.

### 2.2 Genetic Algorithms

The simple (classic, normal) **Genetic Algorithm** was invented by J.H. Holland and is described in his 1975 book *Adaptation in Natural and Artificial Systems* [14]. With the Genetic Algorithm, solutions are typically represented by bit-strings, although other representations have also been used [31]. A population of such solutions, typically randomized at the start of the algorithm, undergoes mutation (a bit in an instance is flipped) and crossover (two instances create two new instances by splitting both parent bit-strings) operators each round, creating a collection of new solutions. The quality, or fitness, of each solution is determined by the fitness function which grades each solution, typically between 0.0 and 1.0.

Typical GA parent selection is fitness-proportional. Each parent produces zero or more children, with the rate being determined by the average fitness

**Algorithm 1** Evolutionary Algorithm [31]

---

```

1: procedure EA
2:    $t \leftarrow 0$ 
3:   initialize_population( $t$ )
4:   evaluate( $t$ )
5:   repeat
6:      $t \leftarrow t + 1$ 
7:     select_parents( $t$ )
8:     recombine( $t$ )
9:     mutate( $t$ )
10:    evaluate( $t$ )
11:    select_survivors( $t$ )
12:  until halt-condition reached
13: end procedure

```

---

of the population. Survival selection is generational, the offspring replaces the parent population at every round. The simple Genetic Algorithm is discussed extensively later on in this thesis and serves as a benchmark for the genetic algorithm that utilizes autonomous selection.

Genetic Algorithms are typically used as function optimizers: the whole purpose of the algorithm is to promote better solutions over worse ones. They are used to solve all kinds of problems from scheduling and planning [8] to evolving art [35], evolving hardware [22, 21], fashion design [18] and solving traveling salesman problems [13, 26, 38].

Calibration of Genetic Algorithms, as described in the previous section, typically constitutes of choosing efficient crossover and mutation rates along with the calibration of the population size. Depending on the type of selection other parameters can be calibrated as well, for instance the tournament size for genetic algorithms that use tournament selection. Calibration of genetic algorithms has been a topic of discussion for quite some time but most researchers tend to use best-practice parameters [7].

Table 2: Sketch of the simple GA [8]

Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional
Survival selection	Generational

### 2.3 Evolution Strategies

**Evolution Strategies** differ in a number of ways from Genetic Algorithms and were contrived separately. In the 1970s Rechenberg and Schwefel came up with

Evolution Strategies in order to solve an optimization problem. Rechenberg developed the first algorithm using Evolution Strategies in 1973 with selection, mutation and a population size of one. Schwefel expanded on the traditional algorithm in 1981 and introduced recombination and larger population sizes [31].

Evolution Strategies alleles tend to be real, floating-point values instead of bit-strings, these values are then optimized using mutation and (global) recombination. With Evolution Strategies, the mutation operator works by modifying one of the values using a normal distribution. The standard deviation for this distribution depends on the number of successful mutations in the past compared to unsuccessful mutations, where a mutation is successful if it produces offspring with a higher fitness. The standard deviation adapts if this number is too high or too low. This is a form of self-adaptation, the algorithm changes its actions depending on the degree of evolution. Self-adaptation is a form of online **parameter control**, which will be looked at in the following section, and differs from parameter calibration or **parameter tuning** in that with parameter control certain parameters change *during* the run, while with parameter tuning the parameters are changed *before* the run.

Recombination too is slightly different: either each gene is randomly chosen from one of the parents or the real values are averaged between both parents. Parent selection is done randomly, survivor selection is done by ranking all the individuals of the offspring, selecting the new generation is done according to the ranking of each individual.

Evolution Strategies have, like Genetic Algorithms, typically been used for optimization. The representation of Evolution Strategies does limit the usage somewhat, as each problem has to be described by one or more real values.

Table 3: Sketch of Evolution Strategies [8]

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survival selection	Best of children or Best of children + parents
Specialty	Self-adaptation of mutation step sizes

## 2.4 Evolutionary Programming

**Evolutionary Programming** (EP) is another branch of Evolutionary Computing, but as it is one that freely combines a number of traits there isn't a typical Evolutionary Programming algorithm. Fogel developed Evolutionary Programming in 1966 from the desire to generate machine intelligence, however EP is mostly used as an optimizer [31].

Representation is done using real values, similar to Evolution Strategies, or any other representation (lists, graphs) that is tailored to the problem domain.

One feature Evolutionary Programming frequently applies is the inclusion of mutation parameters within its representation. This is called meta-EP, however this type of self-adaptation differs from the approaches taken in this thesis. Mutation is done using a normal distribution, again similar to Evolution Strategies, however no recombination is used. As each individual creates a single offspring via mutation, no parent selection is required.

Survivor selection is done via a probabilistic tournament selection, where a number of individuals from the child and parent populations are evaluated against each other with the best being allowed to continue to the next generation. The best individual of each generation is always allowed to be part of the next generation, which is called Elitism.

Evolutionary Programming has the same limits as Evolution Strategies, and has also been applicable for a wide variety of problems. In Blondie24 [9], Fogel describes using Evolutionary Programming in order to calibrate a neural network for playing checkers. Eventually a checker-board evaluator was evolved that could play at a Master level in online tournaments against human players. The main achievement however was doing so without existing knowledge about the game; using Evolutionary Programming the evolved checker programs played against each other, where the best were allowed to create offspring.

Table 4: Sketch of Evolutionary Programming [8]

Representation	Real-valued vectors
Recombination	None
Mutation	Gaussian perturbation
Parent selection	Deterministic (each parent creates one offspring via mutation)
Survival selection	Probabilistic tournament from 2x offspring
Specialty	Self-adaptation of mutation step sizes (in meta-EP)

## 2.5 Genetic Programming

**Genetic Programming** differs from Genetic Algorithms, Evolution Strategies and Evolutionary programming in a number of ways, but most importantly due to the representation: Instead of using bit-strings or values, Genetic Programming has tree-structures as chromosomes. Instead of trying to optimize a particular fitness function Genetic Programming evaluates a tree by executing the nodes in order: each node represents an action, for example addition or multiplication, and each leaf contains a value or parameter. The fitness of such a tree then depends on how well the result of the execution is in comparison to the ideal value.

Mutation and crossover work differently due to the tree-representation. Mutation randomly selects a node or leaf in the tree and switches it with a different one, while during crossover two subtrees of two individuals are exchanged in order to create two offspring. Parent selection is done according to fitness and the

amount of offspring tend to be the population for the next generation.

Genetic Programming tends to be used for different applications than the other sub-domains of Evolutionary Computing. The algorithm gains fitness by modelling different trees while the question of how these trees are modelled is left entirely to evolution.

Table 5: Sketch of Genetic Programming [8]

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survival selection	Generational replacement

## 2.6 Simple Genetic Algorithm

The Simple Genetic Algorithm is the classic algorithm described by J.H. Holland in [14]. We use it primarily as a benchmark to compare the Autonomous Selection Genetic Algorithm.

A Simple Genetic Algorithm works by first randomly initializing a number of individual entities, with each individual being a possible solution to the problem we are trying to solve. Each round, a new generation of entities is constructed via use of mutation and crossover. At the end of each round, selection is applied to come up with the population of entities for the next round.

### 2.6.1 Representation

The representation of entities in a Simple Genetic Algorithm is typically a string of bits, which is the individual's genotype. In order to derive the fitness of an individual, the bit-string is translated into a form where it stands for a possible solution to the problem at hand. This form is called the phenotype of the individual.

By translation into phenotype's, it is possible to solve problems that require a different solution than that is expressed using bit-strings. Integer or real-value solutions can be expressed using Gray encoded bit-strings. More complicated types of solutions (trees, lists) require more elaborate encodings.

### 2.6.2 Mutation

Each round for each individual, there is a (typically small) chance for each bit to be flipped, ie. a 0 becomes a 1 and vice versa. The rationale for this is that mutation has a low probability to introduce either negative or positive changes to an individual which previously didn't exist in the population.

### 2.6.3 Crossover

Each round, the crossover operator is applied to the population in order to generate a different population. In the Simple Genetic Algorithm, two entities are randomly selected from the population. These parent entities generate two new entities, each new individual having (on average) half of its bits from each parent. The rationale for this is that traits that already exist in the population have a chance of spreading throughout the population.

Crossover is seen as the main search operator in Genetic Algorithms [31]. It allows partial solutions to be mixed-and-matched together in order to find better solutions, and works with the idea that better building blocks are kept in the population. As stated above mutation introduces previously non-existent changes, with only crossover the population would eventually become stagnant. The emphasis on crossover is typical of Genetic Algorithms, mutation being the driving force for both Evolutionary Programming and especially Evolution Strategies.

---

**Algorithm 2** Simple Genetic Algorithm

---

```
1: procedure SIMPLEGA
2:    $t \leftarrow 0$ 
3:   initialize_population( $t$ )
4:   repeat
5:      $t \leftarrow t + 1$ 
6:     select_parents_fitness_proportional( $t$ )
7:     recombine_one_point_crossover( $t$ )
8:     mutate( $t$ )
9:     select_survivors_generational( $t$ )
10:  until halt-condition reached
11: end procedure
```

---

## 2.7 Selection in Genetic Algorithms

In order to evaluate autonomous selection properly it is important to also look at the other more typical methods of selection in genetic algorithms. Selection is a combination of parent and survival selection, however it is not required to use the same types of selection for both.

### 2.7.1 Best Selection

Best selection is simply the taking of only the best individuals for survival, or having only the best individuals create offspring. This type of selection leads to hill-climbing behavior; the algorithm only takes the better individuals and throws away the rest. It tends to lead to premature convergence as the populations easily become stuck on a sub-optimal peak in the fitness landscape.

### 2.7.2 Fitness-proportional Selection

Fitness-proportional selection is the choice of parent selection for the simple genetic algorithm. It works as follows:

---

**Algorithm 3** Fitness-proportional selection

---

```
1: procedure FITPROPSELECT(population)
2:   total ← total fitness population
3:   for each individual  $i$  do
4:      $Prob_i \leftarrow total / f_i$ 
5:   end for
6:   select random individual using probability
7:   return individual
8: end procedure
```

---

Better individuals are selected more frequently and either create more offspring or have a better chance in surviving into the next generation.

There are a number of problems with fitness-proportional selection. When there are large differences in fitness between individuals, very fit individuals can quickly take over the population. This leads to premature convergence of the algorithm. On the other side, if there are very small differences between individuals, the selection pressure is absent; slightly better individuals will only ever so often be selected above their peers. Both these issues depend on the fitness values themselves. Should the fitness of an individual lie between 0.0 and 1.0 or 0 and 100? Such minor changes cause fitness-proportional selection to be a poor choice.

### 2.7.3 Rank-based Selection

Rank-based selection was invented to overcome this problem, it works as follows:

---

**Algorithm 4** Rank selection

---

```
1: procedure RANKSELECT(population)
2:   rank population using fitness
3:    $u \leftarrow size\ population$ 
4:    $s \leftarrow 1.0 < s \leq 2.0$ 
5:   for each individual  $i$  do
6:      $r \leftarrow rank\ i$ 
7:      $Prob_{in-rank}(i) = (\frac{2-s}{u} + \frac{2i(s-1)}{u(u-1)})$ 
8:   end for
9:   select random individual using probability
10:  return individual
11: end procedure
```

---

The above example is for linear ranking. Instead of emphasizing the absolute fitness values, it promotes individuals with a relative fitness higher than that of

the rest. However, rank-based selection does leave a chance for individuals with a lower fitness to be selected.

#### 2.7.4 Tournament Selection

Tournament selection works similar to best selection described above, but works on only a small portion of the population:

---

**Algorithm 5** Tournament selection

---

```
1: procedure TOURNAMENTSELECT(population)
2:   Randomly pick  $k$  individuals from the population
3:   best-individual  $\leftarrow$  1
4:   for each selected individual  $i$  do
5:     if  $i$  is better than best-individual then
6:       best-individual  $\leftarrow$   $i$ 
7:     end if
8:   end for
9:   return best-individual
10: end procedure
```

---

The tournament size  $k$  is typically a small number. Tournament selection is similar to rank-based selection as well; however it does so without having to evaluate every individual in the population. It is also possible to combine ranked-based selection with tournament selection by assigning probabilities to each of the individuals in the tournament, however this is uncommon.

#### 2.7.5 Age-Based Survival Selection

In the simple genetic algorithm, survival selection is absent: the amount of offspring created is equal to the number of parents, which take the place of their parents in the next generation.

It is also possible to keep a percentage of the parent population. The age of each individual is remembered, and individuals with a higher age have a lower probability of being part of the next generation.

#### 2.7.6 Elitist Survival Selection

Elitist selection is sometimes used in combination with another form of selection. With Elitist selection, the entities with the highest fitness from both the old and the new population become the population for the next generation.

The rationale for this is that better entities are allowed to survive and reproduce, while worse entities are removed from the population. As such, the algorithm always favors better solutions, which hopefully leads the algorithm to the best possible solution.

## 2.8 Autonomous Selection Genetic Algorithm

What then is an Autonomous Selection Genetic Algorithm (ASGA)? Instead of having a population-wide algorithm determine which entities in a Genetic Algorithm should survive and populate subsequent generations, under autonomous selection “individuals should select and deselect themselves individually from others” [6]. Such an individual would estimate the fitness of the population and determine how well its own fitness is compared with the rest of the population. If its fitness is good enough, it has a chance to reproduce (parent selection). On the other hand, if it isn’t good at all the individual is removed from the population (survivor selection). The rest of the algorithm performs similarly to the Simple Genetic Algorithm described above. The main difference is that the size of the population isn’t controlled, selection doesn’t determine that there is a fixed number of individuals.

### 2.8.1 Algorithm Description

Each individual collects the fitness from a certain number of individuals around him. This knowledge is used to determine how far off the individual is from the mean or median of the population. The chance then for either reproduction or survival is determined by the use of a sigmoid function.

$$\text{sig}(x) = \frac{1}{1 + e^{-m(x-s)}} \quad (1)$$

As is clear from the formula, there are two other parameters required in order to determine  $\text{sig}(x)$ , namely the multiplier ( $m$ ) and the shift ( $s$ ). The shift parameter either shifts the sigmoid towards the left, increasing the chances for the whole population to reproduce or survive, or it shifts the sigmoid towards the right, decreasing these same chances. The multiplier defines the steepness of the slope. A high multiplier will cause a sharp transition between good and bad individuals around the center, a low multiplier will cause a more gradual slope, allowing a higher percentage of bad individuals to reproduce or survive.

This method is thus used for both parent selection and survival, this we actually have two identical formulas. One for parent selection, or fertility:

$$Pf(x) = \text{sig}_{m_f, s_f}(\Delta f) = \frac{1}{1 + e^{-m_f(\Delta f - s_f)}} \quad (2)$$

And one for survival:

$$Ps(x) = \text{sig}_{m_s, s_s}(\Delta f) = \frac{1}{1 + e^{-m_s(\Delta f - s_s)}} \quad (3)$$

## 2.9 Distributed Genetic Algorithms

Although autonomous selection in itself is a new form of selection, there have naturally been attempts to overcome the same problems autonomous selection overcomes. Distributed Genetic Algorithms (DGA) is not a new topic; Tanese

**Algorithm 6** Autonomous Selection Genetic Algorithm

---

```
1: procedure ASGA
2:    $t \leftarrow 0$ 
3:   initialize_population( $t$ )
4:   repeat
5:      $t \leftarrow t + 1$ 
6:     determine_avg_fitness( $t$ )
7:     select_parents_autonomously( $t$ )
8:     recombine_one_point_crossover( $t$ )
9:     mutate( $t$ )
10:    select_survivors_autonomously( $t$ )
11:  until halt-condition reached
12: end procedure
```

---

[34] proposed a method of distributing a genetic algorithm across multiple processors and so was able to achieve near-linear speedup with a genetic algorithm using 64 processors. Belding [2] expanded on her work.

Their class of DGA, and most attempts of distributing GAs, is that of splitting the population of individuals over the nodes. These island-GAs allow a proportion of the subpopulations to be selected and sent to other islands. The island-GA worked very well for Tanese, and Belding found that the performance on certain problems (in the Royal Road class) was on average equal to the performance of a canonical genetic algorithm. Performance of their DGA varied however; certain problems with certain degrees of migration caused the performance to drop sharply.

Despite all the research on island-GAs, using them for massively distributing a GA over vast numbers of nodes will be overly complicated. Island-GAs send instances between nodes, doing so over thousands of nodes would require a lot of bookkeeping and communication.

Looking at the previous section, autonomous selection offers a major improvement here compared to current methods of distributing a genetic algorithm. Instead of sending individuals, it is more efficient to only communicate the average fitness of a population. In a large distributed system using a genetic algorithm with autonomous selection, nodes of which the population die out (the individuals were on average less fit than its neighbors) can simply be re-populated. Surviving populations can send their best individuals, after a certain interval, back to the main project servers. In turn, the main servers can use the collected individuals to re-populate extinct or stagnant nodes. Communication of a node between peers and with the main servers are then minimized, making it possible to scale the algorithm virtually without problems linearly over a large number of separate nodes.

### 3 History of Calibration and Meta-algorithms

In this chapter we will look at the history of genetic algorithms and various approaches in finding efficient parameters. We describe the history of the meta-algorithms we have found promising enough.

#### 3.1 Parameter Calibration of Genetic Algorithms

Ever since the birth of genetic algorithms by Holland [14], much of the research in this field has been on determining, controlling and calibrating the various parameters which a GA depends on. Traditionally, the parameters most looked at are the rates for crossover and mutation, although other parameters for various selection strategies and the population size have also been studied frequently [7], [8]. However, from [7] it is also clear that there aren't any single parameters that will be effective for all problem domains. What works for certain numerical problems [16] might be grossly ineffective for scheduling or a traveling-salesman problem.

Even so, most genetic algorithms have parameters that have been selected by trial and error or have been adapted from values gained by best practices in the selected problem domain. Both strategies are suboptimal, both in the time spent by the researchers involved and the resulting algorithm, of which only a vanishingly small parameter search space has been experimented with. Although problems might seem similar, results can be, due to the No Free Lunch theorem, a lot worse.

There have been researchers trying to overcome these issues. A recent example is [27], where a framework has been constructed for parameter sweeping across the search space using a cluster of machines. Such tactics, although useful for automating a process that is still done manually very often, can cover only a small portion of the entire search space due to the granularity and the need to artificially tone down the amount of parameters over which the framework sweeps. This approach simply doesn't scale for increasing amounts of parameters due to the exponential increase in complexity. Another problem is that parameter tuning (setting the parameters to a fixed position throughout the run of the algorithm) doesn't lead to an optimal algorithm, as the optimum will change during the course of the run of the GA.

#### 3.2 Adaptation and Parameter control

Many researchers have looked into adaptation and self-adaptation in order to overcome the suboptimal fixed parameters [7] gained through the brute-force method. The difference between these techniques is that adaptive parameter control adapts the parameters using the current state of the GA as a whole, while with self-adaptive parameter control the parameters are encoded into the individual chromosomes. Either way, for both approaches there are new issues to deal with as the GA has to be fundamentally changed in order to make

the adaptation of parameters possible [25, 33, 7]. Self-adaptation has an additional drawback; because the parameters themselves must be tuned too, the search space for the genetic algorithm increases dramatically. A GA using self-adaptation simply takes longer to reach an optimum on static problems.

The use of adaptation is more similar as a meta algorithm than that self-adaptation is. With adaptation, the algorithm parameters are controlled during runtime according to an external mechanism, which stands outside the genetic algorithm itself. With a meta algorithm, the parameters are instead calibrated before runtime. With both approaches the parameters are modified by an external mechanism, unlike with self-adaptation.

### 3.3 Meta Genetic Algorithms

The irony of the problem of GA parameter calibration is that these are the types of problems that can be handled very well by a GA: there is a vast search space with a high degree of complexity. Using a GA for calibrating the parameters of a GA was first done by Mercer and Sampson in 1978. In their research they used a meta-GA for determining the best crossover and mutation operator probabilities. Their research was however limited due to the large computation costs at that time and based on a single run with a limited search space.

A more elaborate experimentation was conducted by Grefenstette [12] and his approach has since been dubbed “Meta Genetic Algorithm”, or meta-GA. The main difference with his approach is that the second GA, the “lower-level” algorithm of which the parameters are being calibrated, does not have to be changed for the application of a meta-GA.

The search space for Grefenstette’s meta genetic algorithm consisted of  $2^{18}$  parameter combinations. The results consisted of an optimization of 3% using slightly better parameters than previously known. However, only 20 meta-generations were evolved and only 2000 lower-level evaluations were performed. Nowadays this could be completed in a matter of minutes. It is therefore our opinion that the use of meta genetic algorithms is worthy for reconsideration.

Research however didn’t stop after these first steps. In the early nineties a number of studies were conducted on meta-GAs. Shahookar and Mazumder used a meta-GA approach in [29] for solving a cell placement problem for industrial circuits, which resulted in a decrease of 20 to 50 times in the number of configurations which had to be evaluated. Lee and Takagi [19] used a meta-GA similar to that of Grefenstette in studying the effects of a dynamic adaptive population size, crossover and mutation rates on the De Jong [16] set of test problems. Their results confirmed recent theoretical results on optimal mutation rates [10].

More recent research on meta-GAs has been infrequent and results have been varying at best. In [4] the conclusion was that although meta-GA are able to produce reasonable parameters for various problems, these parameters are far from optimal. The meta-GA approach was: “Jack of all trades, master of none”. Other researchers [36, 37] however have found more positive results

for this type of black-box optimization. In [5], a meta-GA was used for time-series forecasting. The results, when compared to existing techniques, showed an all-round improvement on existing forecasting packages and similar results to an algorithm specifically designed for the domain. The researchers remarked that the strengths of the meta-GA approach is clearly seen for problems with a complex non-linear behavior.

In a recent study [28], Samsonovich and De Jong took a numerical approach to the “free lunch” that meta algorithms and meta-GAs seemed to offer. They attempted to calibrate GAs on three problems (2D optimization, the eight puzzle and binary dendritic tree reconstruction) and concluded that meta-evolution “may improve the performance of an evolutionary algorithm, if the fitness is appropriately defined at the meta-level(s)”. They also remarked that the search space that the meta algorithms look at must be chosen carefully, and that more levels of evolution could be useful, as long as fitness improved.

A different use of a meta-GA is to use it for dynamic fitness landscapes; ie. problems that change over time. In [32] Stanhope and Daida used a similar meta-GA as Grefenstette for determining the mutation and crossover rates for a dynamic fitness function. When the required mutation and crossover rates were found the fitness remained constant. They do note that it would have been possible to use self-adaptation (as discussed above) instead, but they regarded self-adaptation as not fitting for their particular experiment, as they were trying to achieve a baseline and insights for future research.

### 3.3.1 The “Grefenstette” Meta-GA

The classic meta genetic algorithm works by optimizing parameter-values. Like a normal genetic algorithm, these values are encoded using a binary representation. We will use Gray encoding for this in our experiments in order to minimize the distance between mutations.

For every set of parameter-values the meta genetic algorithm sees a single individual. In order to evaluate the fitness of this individual the lower-level genetic algorithm is executed with the set parameter-values that belong to the individual. The value the lower-level algorithm returns (most often the highest fitness reached) is the fitness with which the meta genetic algorithm will determine if the individual will be selected for the next generation and/or the possibilities of the individual to reproduce.

When being selected for the following generation, there is a chance that the individual will undergo either mutation or crossover. These two operators respectively either flip a bit in the binary representation of a parameter value or combines the bit-string with that of a different individual in order to produce two new child-individuals with each containing half of the values of their parents.

## 3.4 Relevance Estimation and Value Calibration

In [23] a similar approach to a meta-GA is used in calibrating the parameters for energy-policy multi-agent simulations. In [24] this approach to calibrate the

**Algorithm 7** Pseudo-code for the Meta Genetic Algorithm

---

```

1: procedure METAGA
2:    $t \leftarrow 0$ 
3:   initialize_meta_population( $t$ )
4:   repeat
5:      $t \leftarrow t + 1$ 
6:     determine_fitness_entities( $t$ )
7:     select_parents_fitness_proportional( $t$ )
8:     recombine_one_point_crossover( $t$ )
9:     mutate( $t$ )
10:    select_survivors_generational( $t$ )
11:  until halt-condition reached
12:  function DETERMINE_FITNESS_ENTITIES( $t$ )
13:    for entity  $i$  in generation  $t$  do
14:      parameters  $\leftarrow$  convert_to_parameters( $i$ )
15:      fitness  $\leftarrow$  do_genetic_algorithm(parameters)
16:    end for
17:    return best-fitness
18:  end function
19: end procedure

```

---

parameters is further developed and tested on a number of abstract and real benchmark problems.

The method for relevance estimation and value calibration uses search methods based on information theory for optimization. By estimating the model complexity using the Shannon entropy from the distribution of the results, the less-relevant parameters of the model are discarded. For the remaining parameters a search algorithm similar to a meta-GA is used for calibrating the parameters (where a model is a list of parameters for the simulation).

### 3.4.1 Estimation of Distribution Algorithms

The REVAC method is a type of Estimation of Distribution Algorithm (or EDA). EDAs are a relatively new search algorithms that maintain a population of possible solutions and as such are similar to evolutionary algorithms, however the population is used to “estimate a probability distribution over the search space that reflects what are considered to be important characteristics of the population” [40]. A new population is generated by sampling this distribution, which replaces traditional crossover and mutation in genetic algorithms.

EDAs can be seen in the light of the principle of maximum entropy. Without losing what is known, the algorithm by this principle should maximize the information entropy. The population of an EDA is adjusted in order to learn as much as possible about the fitness landscape.

In [30] one of the main problems with EDAs is investigated, namely the lack of diversity over time. Like a conventional evolutionary algorithm without

mutation, EDAs are prone to stagnation; they will then not be able to find any better solution, even if it exists. The amount of diversity within the population appears to be a major problem for EDAs.

### 3.4.2 Workings of REVAC

EDAs work through the use of probability functions, and in such are grounded on existing information theory for optimization. By estimating the model complexity using the Shannon entropy from the distribution of the results, REVAC finds and discards the less-relevant parameters of the model. For the remaining parameters an EDA search algorithm was used for calibrating the parameters (where a model is a list of parameters for the simulation).

The main difference between the Relevance Estimation and Value Calibration method used in [23] and a traditional genetic algorithm is thus the lack of separate crossover and mutation (in [23] termed as imitation and innovation). Instead of these two operators, the new parameters are selected by first selecting an individual randomly from the population. From this individual the two closest neighbors are determined. The new parameter-value becomes a value between the parameter of both of these neighbors. This is done for every parameter, thus the end result is that REVAC combines  $x$  parents and mutates the result. As such, this operator is similar in nature to a combination of multi-parent crossover and mutation; each offspring has a different parent for each parameter. The REVAC-method solely uses this instead of the traditional crossover and mutation operators in genetic algorithms, however the question remains if such an approach for parameter calibration is more successful than a meta-GA (or other methods) in finding effective parameters for a GA.

The REVAC method only has a few parameters with which it can be tuned, namely: the size of the initial population, the number of entities to replace at each round and the amount of smoothing. The smoothing parameter is the distance of the two neighbors which are chosen in the REVAC\_operator function in Table 8. A smoothing of 1 means that the two nearest neighbors on both side are chosen, while a smoothing of 4 means that two neighbors are chosen that have 3 intermediary entities.

---

**Algorithm 8** Pseudo-code for Relevance Estimation and Value Calibration

---

```
1: procedure REVAC
2:    $t \leftarrow 0$ 
3:   initialize_meta_population( $t$ )
4:   repeat
5:      $t \leftarrow t + 1$ 
6:     determine_fitness_entities( $t$ )
7:     parents  $\leftarrow$  select_best_100( $t$ )
8:     create_offspring(parents,  $t$ )
9:     replace_1_oldest( $t$ )
10:  until halt-condition reached
11:  function DETERMINE_FITNESS_ENTITIES( $t$ )
12:    for entity  $i$  in generation  $t$  do
13:      parameters  $\leftarrow$  convert_to_parameters( $i$ )
14:      fitness = do_genetic_algorithm(parameters)
15:    end for
16:    return best-fitness
17:  end function
18:  function CREATE_OFFSPRING(parents,  $t$ )
19:    for each parameter do
20:       $x \leftarrow$  random parent from parents
21:      offspring_parameter  $\leftarrow$  REVAC_operator( $x$ )
22:    end for
23:    return offspring
24:  end function
25:  function REVAC_OPERATOR( $x$ )
26:    select two neighbors of  $x$ 
27:    return random value from neighbor interval
28:  end function
29: end procedure
```

---

## 4 Problem description

In letting the fixed-size population of genetic algorithms go and using autonomous selection, there is one major drawback: how can you reach a stable population without interfering with the local determining of survival and reproduction? In [6], it is made clear that coming up with efficient parameters isn't an easy task, even for a relatively simple problem instances of the Spears multi-modal problem generator, which is defined in [31].

The approach used in the original paper [6] was composed of good old trial-and-error. After a lot of trials, there were some settings that did give good results: ie, the population didn't go extinct right away. The drawback of the best settings found, which were described in [6], was that the birth- and death-rates were very low, thus the results weren't very good. Thus the search goes on in coming up with both working and efficient parameters for the autonomous selection approach. This leads to the first research question.

### 4.1 Research Questions: Autonomous Selection

Research Question 1a: Are there parameters of the autonomous selection genetic algorithm to be found which haven't yet been discovered and display a better performance than the parameters which are known to date?

In [6] where autonomous selection was first introduced the algorithm and the parameters found were only used on instances of the Spears multi-modal problem generator. The question remains how stable the discovered parameters actually are, and how well the algorithm works on other problem instances.

Research Question 1b: How sensitive and stable are the parameters of the autonomous selection genetic algorithm? Are parameters that worked well for one problem instance equally effective when solving other problem instances, or is it more effective to use different parameters when solving different problem instances?

Following from these questions we have to come up with a final judgement on autonomous selection. Is it possible to have both an algorithm that has the advantages of autonomous selection and to also have results that are comparable to other genetic algorithms?

Research Question 1c: How does a well-tuned autonomous selection genetic algorithm compare to a well-tuned simple genetic algorithm? In which cases is it advisable to use a genetic algorithm that uses autonomous selection instead of other forms of selection?

### 4.2 Research Questions: Meta-Evolutionary Algorithms

There is an alternative to trial-and-error or exhaustive search, namely to use a second algorithm to find parameters for the first algorithm: the meta-algorithms

discussed in the previous chapter. A number of different approaches are available and in order to answer the above research questions we will use these meta-algorithm techniques in order to optimally calibrate our algorithms.

Research Question 2a: How well do different meta-evolutionary algorithms perform when trying to find parameters for the autonomous selection genetic algorithm? Are the parameters found better than those found previously by manual calibration?

After we have answered the above research question, it is then possible to judge the usefulness of these meta-evolutionary algorithms. As it is not wise to make this judgement merely on the calibration of a single algorithm, we will also try to calibrate the simple genetic algorithm as a benchmark.

Research Question 2b: How well do different meta-evolutionary algorithms perform when trying to find parameters for a simple genetic algorithm? Are the parameters found better than those found previously?

With the answers to these two questions we can make a more weighed judgement on the usefulness of meta-evolutionary algorithms compared to manual calibration.

Research Question 2c: How useful are meta-evolutionary algorithms in calibrating evolutionary algorithms in general? Is it better to have an algorithm search for parameters than to use best-practice parameters?

### 4.3 Hypotheses

Now the research questions are defined, however the methods on how to answer these questions are still not known. In this section we will expand our research questions into falsifiable hypotheses.

#### 4.3.1 Autonomous selection calibration

Research Question 1a: Are there parameters of the autonomous selection genetic algorithm to be found which haven't yet been discovered and display a better performance than the parameters which are known to date?

Although a lot of time has already been invested in searching for working and efficient parameters for autonomous selection, our opinion is that only a very small subsection of the overall search space has been uncovered. We expect that not only are there better parameters waiting to be found, but also that finding these parameters will give a better understanding of how autonomous selection

works. We suspect that finding new parameters will lead to more insight on how to improve autonomous selection.

In the original autonomous selection publication, the mean best fitness and success rate were used to compare the results. We will do the same as doing so will make our findings comparable.

Hypothesis 1a: It is possible to find better parameters for the autonomous selection genetic algorithm when comparing the mean best fitness and success rate.

#### **4.3.2 Sensitivity of parameters**

Research Question 1b: How sensitive are the parameters of the autonomous selection genetic algorithm? Are parameters that worked well for one problem instance equally effective when solving other problem instances, or is it more effective to use different parameters when solving different problem instances?

We suspect that small changes in the fitness landscape will cause changes in the working of autonomous selection. The balance of population in autonomous selection appears to be very sensitive, so we expect that different problem instances will require other parameters for autonomous selection to work properly.

The main method of measurement here is to look at the variance of the results.

Hypothesis 1b: Parameters of the autonomous selection genetic algorithm are sensitive, in that small changes to either the parameters of autonomous selection or the underlying fitness landscape of the problem instance can lead to an ineffective algorithm.

#### **4.3.3 Performance**

Research Question 1c: How does a well-tuned autonomous selection genetic algorithm compare to a well-tuned simple genetic algorithm? In which cases is it advisable to use a genetic algorithm that uses autonomous selection instead of other forms of selection?

In spite of the drawback mentioned above, we do expect that autonomous selection can work properly, in which we mean that, if the algorithm is calibrated properly, it can come up with results equal in overall fitness to a simple genetic algorithm. We don't see any reason why this would not be the case: the autonomous selection algorithm does have more parameters, yet if everything is set up properly and the population remains constant the autonomous selection algorithm should have an efficiency comparable to that of a simple genetic algorithm.

Hypothesis 1c: A genetic algorithm using autonomous selection, with well-tuned parameters, can give results equal to or better than

a well-tuned simple genetic algorithm when looking at the mean best fitness and success rate.

#### 4.3.4 Comparable parameters via meta-evolutionary algorithms

Research Question 2a: How well do different meta-evolutionary algorithms perform when trying to find parameters for the autonomous selection genetic algorithm? Are the parameters found better than those found previously by manual calibration?

Although a lot of time and effort has been put into manual calibration, we expect that a meta-evolutionary algorithm could come up with similar or better results. Maybe certain best-practice parameters have been decided upon by more than just raw performance, but in that case a meta-algorithm could make use of more than just the best fitness in deciding how well a parameter vector should be scored. Naturally, certain meta-algorithms would be expected to perform better than others.

Hypothesis 2a: Parameters found using meta-evolutionary algorithms on autonomous selection will have a higher mean best fitness and success rate than when using either parameters found using manual calibration or best-practice parameters.

#### 4.3.5 Meta-evolutionary and simple genetic algorithms

Research Question 2b: How well do different meta-evolutionary algorithms perform when trying to find parameters for a simple genetic algorithm? Are the parameters found better than those found previously?

The simple genetic algorithm has a lot less parameters and those parameters have been determined after decades, not weeks, of experimentation. We thus don't expect any radical improvements compared to the parameters already known.

Again, we will compare the mean best fitness and success rates of the algorithms in order to get a valid measure of performance.

Hypothesis 2b: Parameters found using meta-evolutionary algorithms will not cause a significantly higher mean best fitness than when best-practice parameters are used with the simple genetic algorithm.

#### 4.3.6 Applicability

Research Question 2c: How useful are meta-evolutionary algorithms in calibrating evolutionary algorithms in general? Is it better to have an algorithm search for parameters than to use best-practice parameters?

We do expect the above question to be true: although a meta-evolutionary algorithm will not beat decades of previous experimentation and calibration, We do think that the amount of time a researcher has to invest in determining the parameters of an algorithm can be reduced drastically. We also think that the amount of CPU time will be decreased: exhaustive search for parameters can be compared to blind search while manual calibration is often not much more than hill-climbing.

Hypothesis 2c: Using meta-evolutionary algorithms in finding parameters instead of exhaustive search or manual calibration will show a drastic reduction of time required. Meta-evolutionary algorithms should be considered in all cases where best-practice parameters are not directly applicable, as they will give better results.

Summarizing, we have two classes of hypotheses: The first being those related to autonomous selection, the second being those related to meta-evolutionary algorithms. Each of these classes are subdivided into three separate hypotheses which we will examine in the next chapter.

#### 4.4 Summary of Hypotheses

Table 6: Summary of Hypotheses

---

Hypotheses	
1a	<b>Calibration</b> - Possible to find better autonomous selection parameters
1b	<b>Sensitivity and Stability</b> - Parameters of the autonomous selection genetic algorithm are sensitive
1c	<b>Improvement</b> - A well-tuned autonomous selection genetic algorithm, gives at least comparable performance
2a	<b>Comparing algorithms</b> - Autonomous selection parameters found using meta-evolutionary algorithms will have a higher mean best fitness and success rate compared with previously-known parameters.
2b	<b>Simple GA</b> - Simple GA parameters found using meta-evolutionary algorithms will not cause a significantly higher mean best fitness.
2c	<b>Applicability</b> - Meta-evolutionary algorithms should be considered in all cases where best-practice parameters are not directly applicable.

---

## 5 Experimental Design

In this chapter we will describe the approach chosen to answer the hypotheses. First, we will look at screening the problem domain to determine which search space our algorithms will look at. Second, we will look at verifying the results of previous experiments in order to make sure the algorithms perform as described.

### 5.1 Experimental Design: objectives

Our approach follows the *design of experiments* (DOE) concept, which is a framework for planning experiments. For more information about DOE in general and within evolutionary computation see [1].

#### 5.1.1 Optimality

Experimenting to reach an optimal condition for a problem has been discussed by Hotelling [15], Friedman and Savage [11] and Box and Wilson [3]. Box and Wilson, a chemist and statistician, looked at determining the optimal conditions of chemical processes, but in “On the experimental attainment of optimum conditions” they generalized their methods for all domains requiring empirical experimentation. They define the *experimental region*, the search space the experimenter investigates.

The problem is to find, in the smallest number of experiments, the point within the experimental region at which the response is a maximum or a minimum. In our field, yield, purity or cost of product are the responses which have to be maximized (or minimized in the case of cost). The factors affecting these responses are variables such as temperature, pressure, time of reaction, proportions of the reactants. [3]

They state that a sure way of finding optimum conditions would be to explore the whole experimental region. A grid of experiments would have to be constructed that extend through the experimental region. This is possible for grids with a small density, however if the complexity of the problem rises (more variables are added or the granularity increases) then the grid quickly contains far too many points to be investigated. However if the experimental error is small and the experiments can be conducted sequentially, then shorter methods are possible.

Box and Wilson continue by looking at only a few sub-regions and use their results to look further around the most promising regions. They admit that their hill-climbing method might get stuck on sub-optimal peaks and miss “a higher ultimate maximum”, yet they take a pragmatic stance and are satisfied with the result if further experimentation is not possible.

#### 5.1.2 Robustness

Finding good results is the goal of a genetic algorithm, and thankfully our algorithms are frequently able to find optimal solutions where hill-climbing doesn't.

Frequently however a researcher wants to have his algorithm to also find solutions for other, similar problems. Erroneous input or other noise happens in real-world situations, and it is useful to make sure that an algorithm's performance doesn't suffer because of it. Research has been done in order to find out how an algorithm can be both optimal and robust.

### 5.1.3 Statistical Significance

Any experiment can contain noise. The task of the researcher is to determine if his results are statistically significant in spite of the possibility that his results could be invalid due to random noise.

In our experiments we will be comparing a number of results empirically in order to determine which algorithm is "better", however, without properly defining when one result is better than another such research can hardly be considered scientific. A solid statistical backing is therefore required when judging our hypotheses.

In our analysis we will be conducting our statistical hypothesis testing using the two-sided t-test. Using a reference distribution, we can determine if the difference between results is statistically significant within a certain probability. From [20] we learn that a probability of 1% is commonly used in empirical research. If our measured difference lies in this region we can conclude that our null-hypothesis (the results don't show any difference, the means of the two results are equal) must be rejected. Formalized, the null and alternative hypothesis become:

- $H_0 : \mu_1 - \mu_2 = 0$
- $H_1 : \mu_1 - \mu_2 \neq 0$

Using the two-sided t-test, we determine which of the above is correct:

$$S_1 = \frac{1}{n-1} \sum (u - X)^2 \quad (4)$$

$$S_p^2 = \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2} \quad (5)$$

$$t_0 = \frac{\bar{y}_1 - \bar{y}_2}{S_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \quad (6)$$

In equation 4 we determine the standard error over the results for  $S_1$ . We use this value in equation 5 to determine the probability value, adjusted if the amount of results differ. In 6 we determine the difference between the two mean results and divide this by the probability value in order to get the t-distribution. With this t-distribution we can determine if our results are statistically significant.

#### 5.1.4 No Free Lunch?

There is a mathematical basis called the **No Free Lunch theorem** that no algorithm or algorithm settings are optimal for solving all kinds of problems [39], with the condition that the algorithm in question doesn't incorporate problem-specific knowledge, which is usually the case with genetic algorithms.

In particular, if an algorithm performs better than random search on some class of problems then it must perform worse than random search on the remaining problems. Thus comparisons reporting the performance of a particular algorithm with particular parameter settings on a few sample problems are of limited utility. While such results do indicate behavior on the narrow range of problems considered, one should be very wary of trying to generalize those results to other problems [39].

Is it then completely useless to empirically compare one algorithm with another? This is naturally not the case. One can generalize over a broad range of problems belonging to the same class, but only when those problems have characteristics that are well-defined. [31] describes two important characteristics, namely **epistasis** and **multi-modality**. On epistasis:

A system (problem) has a low epistasis if the optimal allele for any locus (single location in a chromosome) depends on a small number of alleles at other loci [31].

Likewise, if the optimal allele depends on a large number of other alleles then the epistasis is high. An algorithm or parameter that is effective on problems with low epistasis will likely be inefficient on problems with a high epistasis.

The characteristic of multi-modality describes how many "false peaks" the algorithm can get stuck on. In a fitness landscape there are often multiple peaks where the algorithm can obtain a higher fitness. Most of the time only one of those peaks is truly optimal; the other peaks lead to a sub-optimal solution and thus are termed false peaks. If a problem has a high degree of multi-modality, then there are many peaks that can throw the algorithm off course.

In [17], De Jong et. al. looked at the confusion that surrounds comparisons between EAs, which arose because many studies were comparing apples and oranges. Researchers previously based their results on experiments performed on a few test-problems, and many contradictory studies and discussions resulted because of this. By creating problem generators that target a specific problem domain comparisons become more systematic and useful. Using the characteristics of epistasis and multi-modality, Spears created a multi-modal problem generator based on Boolean Satisfiability problems that creates a class of problems. We will use one of his problem generators to test our algorithms, ensuring that the algorithms chosen will also work on related problems.

## 5.2 Problem instances

### 5.2.1 Spears multi-modal problem generator

The main problem instance the initial research focussed on was done on the same problem instance for the autonomous selection genetic algorithm as in [6], which were instances of the Spears multi-modal problem generator [31]. The advantage of using a problem generator is that it is fairly easy to scale up from simple problems to more complex ones and so get an idea how well an algorithm performs on a series of problem instances. Also, the problem instances generated with the Spears problem generator have a high degree of controllable multi-modality. Because of this the problem instances can become increasingly difficult.

As stated in [31], one of the weaknesses of studying new algorithms empirically is the question of what to use as a benchmark problem. How robust is an algorithm when similar problems need to be solved? The algorithm can be carefully tuned and can outperform all other algorithms, however what happens if the problem instances which were used for tuning are far off from the new problem instances? These questions were addressed by Spears in order to determine the usefulness of mutation and recombination in genetic algorithms, however the same questions apply to our algorithms.

The generated problem instances are actually very simple. They consist of a fitness instance defined by an arbitrary number of peaks, which increase in size and have a minimum of 0.5 and a maximum of 1.0. Each peak is a random bit-string, and the fitness of each individual depends on the Hamming distance of it to the nearest peak and is scaled by the size of the peak. As such, each possible solution has a fitness between 0.0 and 1.0.

$$f(c) = \frac{1}{L} \max_{i=1}^P \{L - \text{Hamming}(c, \text{Peak}_i)\} \quad (7)$$

The consequence of having the problem instances as described above is that they closely resemble a Boolean Satisfiability (SAT) problem which, when used with a minimum of 3 literals, is a NP-complete problem. It must be noted however that problem instances of Spears multi-modal problem generator are a SAT-simplification and are not automatically NP-complete. Also, the problem instances are in disjunctive normal form (DNF) while satisfiability problems are in conjunctive normal form (CNF). Problems in CNF can be converted to DNF and vice versa however.

How then is the characteristic of epistasis related to this problem generator? With a higher number of peaks, the epistasis of the problem becomes stronger. In [17] De Jong, Potter and Spears compared the use of crossover and mutation of a GA using this problem generator on 1 and 500-peak problems. The GAs with crossover needed more evaluations to reach a fitness comparable to the GA with only mutation, which agrees with their hypothesis that GAs using crossover have more trouble with problems with a high-epistasis.

### 5.3 Screening: Manual calibration and framework development

Before applying any meta-algorithms, we first tried to determine efficient autonomous selection genetic algorithm parameters by hand. The reason to do this was because the problem really had to be hard: If efficient results would have been easily obtained then calibrating the algorithm wouldn't be required. This also gave some results on how the Autonomous Selection Genetic Algorithm performed with various settings. In the end, the results we obtained were not any more efficient than those from [6].

After manual calibration, we constructed an optimization-framework which consisted of both the Meta-Genetic Algorithm and REVAC methods. By importing a fitness-module, which contains the code required to evaluate the parameters, the framework makes it easy to use meta-algorithms. The Python-code for the framework is freely available via the authors. More details about the framework can be found in Appendix A.

#### 5.3.1 Determining search space and granularity

The main problem when using meta-algorithms in the past has been determining the range within to search and the granularity with which to search this range. Together, these choices are the search space and determine if the algorithm can find good results at all within a certain time-frame. If the range is too small, then there is a chance of missing good results completely, if the range is too large then the algorithm might take too long to find any good results. The granularity of the search works the other way around; a large granularity causes the search algorithm to skip possible good results, while a small granularity can cause the algorithm to take too long to complete. We chose fixed values for both the size of the search space and the granularity, however in further meta-algorithm research it might be interesting to look at adaptive granularity.

The four parameters determining the chance of selection for the autonomous selection genetic algorithm can be any real value, so there had to be a range within the meta-algorithm would remain. This problem didn't exist for the mutation or crossover rates, as these were normal probabilities between 0.0 and 1.0.

The experiments we will run consist of optimizing the 4 parameters for selection together with (depending on the experiment) the mutation and crossover rates and population size. Each parameter is defined by a string of 16 bits, thus for each parameter there are  $2^{16}$  possible values. These values are scaled to between 0.0 and 1.0 for both the mutation and crossover rates. For the 4 autonomous selection parameters the values are scaled to between 0.0 and either  $2^{16}/10$  or  $2^{16}/1000$  for the four selection parameters. The range of the autonomous selection parameters has been determined by the initial trial & error described earlier. We make sure that the granularity is small enough so that we will find (near-)optimal values.

## 5.4 Verification

### 5.4.1 Verifying Autonomous Selection results

The autonomous selection genetic algorithm is fairly new and the initial results have not been widely distributed. The first task therefore is to verify the findings of the original authors in [6]. The results of the authors weren't published and therefore haven't been available for review, thus it is important to verify the results they obtained.

For the autonomous selection genetic algorithm the parameters required are shown in Table 7.

Table 7: Best manual ASGA settings according to [6].

Parameter	Parameter Setting
Crossover	0.2
Mutation	0.05
Start Population	100
Parent selection, Shift	0.12
Parent selection, Multiplier	60.0
Survivor selection, Shift	-0.14
Survivor selection, Multiplier	1000.0

Using these parameters we run the ASGA algorithm 10 times. The results of the verification are shown in Table 8. For a more thorough comparison in later experiments we also record other variables about the runs besides the mean best fitness.

Table 8: Best manual ASGA results.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Population Diversity.

Peaks	ASGA / original		ASGA / verified			
	Mean Best Fitness	Success Rate	Mean Best Fitness	Success Rate	Evaluations	Population Diversity
1	0.997	76%	0.92	0%	10000	0.0073
10	0.998	84%	0.919	0%	10000	0.0072
100	0.982	28%	0.908	0%	10000	0.0072

As is clear from Table 8, the results differ from the values published. This discrepancy has yet to be explained, however the parameters used were the same as which were known to be the best. As we are also interested in keeping the population size stable we will plot the population size and the variance of the verification runs in the next section.

## 5.4.2 ASGA Verification Plots

Figure 1: 10-run ASGA Verification: Mean Best Fitness, 1 peak

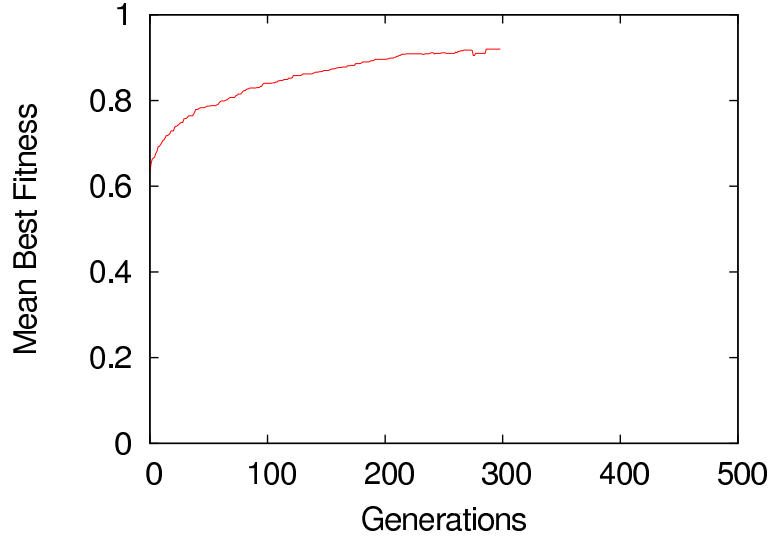


Figure 2: 10-run ASGA Verification: Mean Population Size, 1 peak

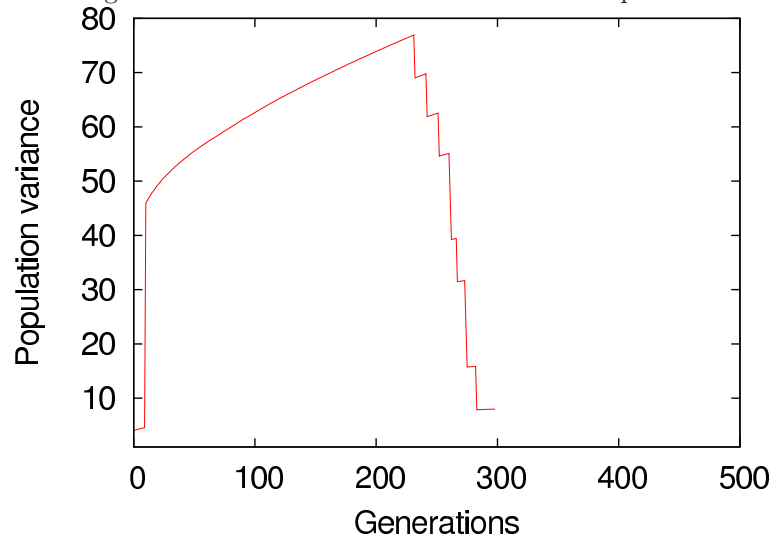


Figure 3: 10-run ASGA Verification: Population variance, 1 peak

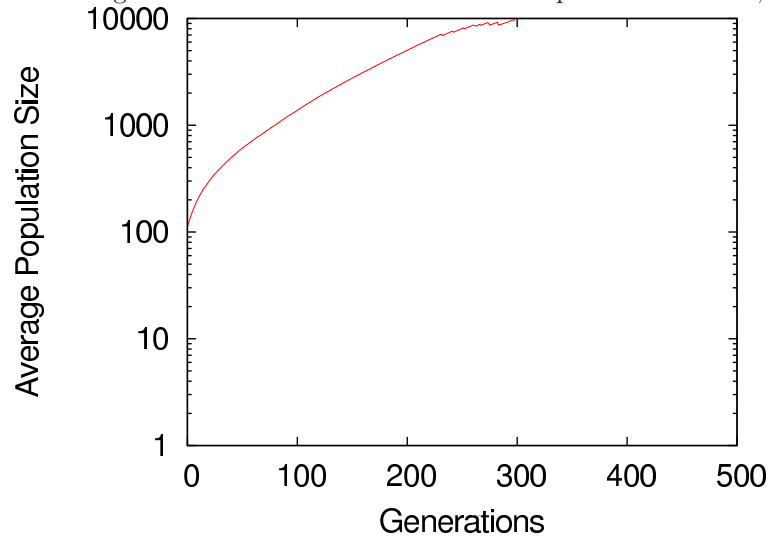


Figure 4: 10-run ASGA Verification: Mean Best Fitness, 10 peak

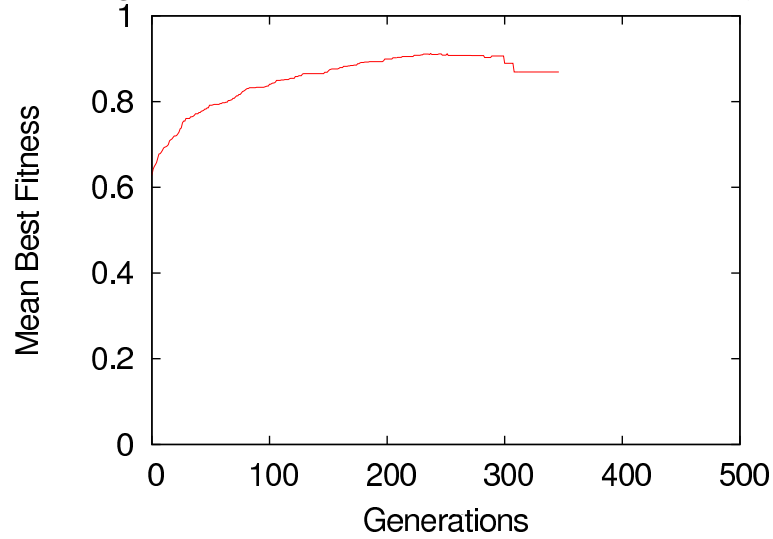


Figure 5: 10-run ASGA Verification: Mean Population Size, 10 peak

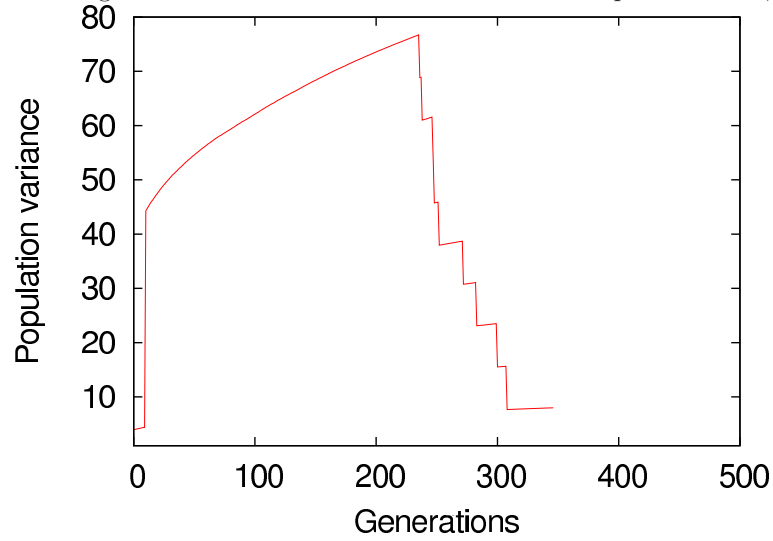


Figure 6: 10-run ASGA Verification: Population variance, 10 peak

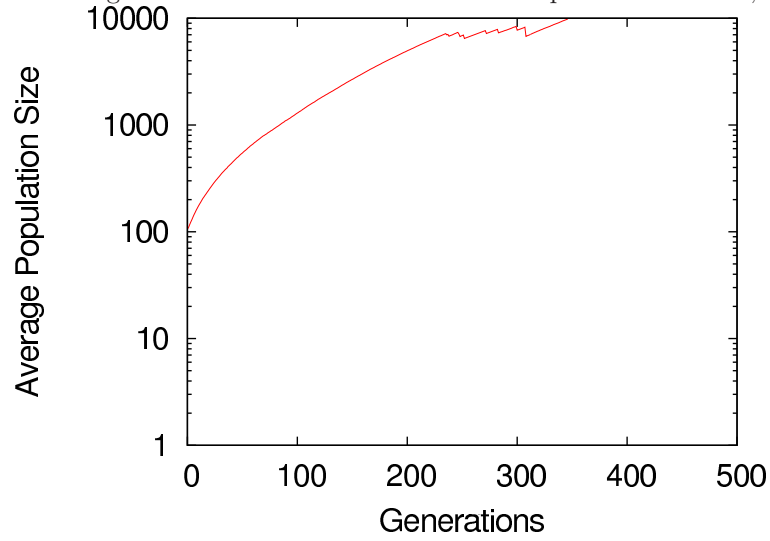


Figure 7: 10-run ASGA Verification: Mean Best Fitness, 100 peak

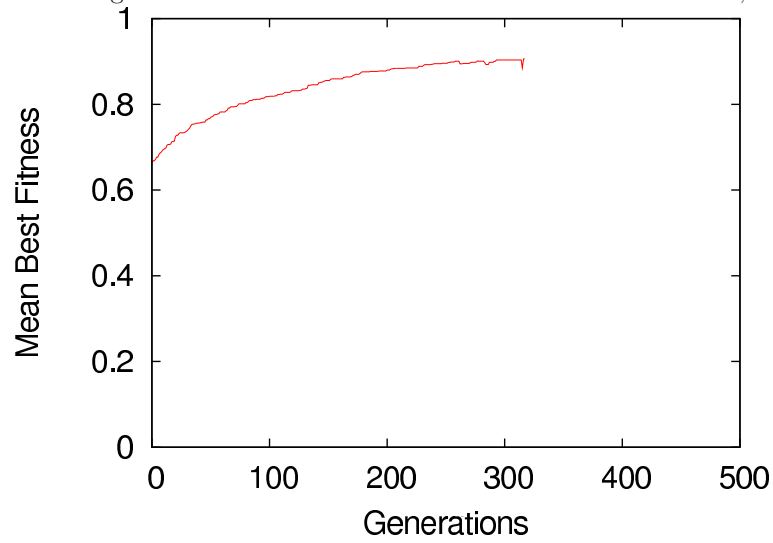


Figure 8: 10-run ASGA Verification: Mean Population Size, 100 peak

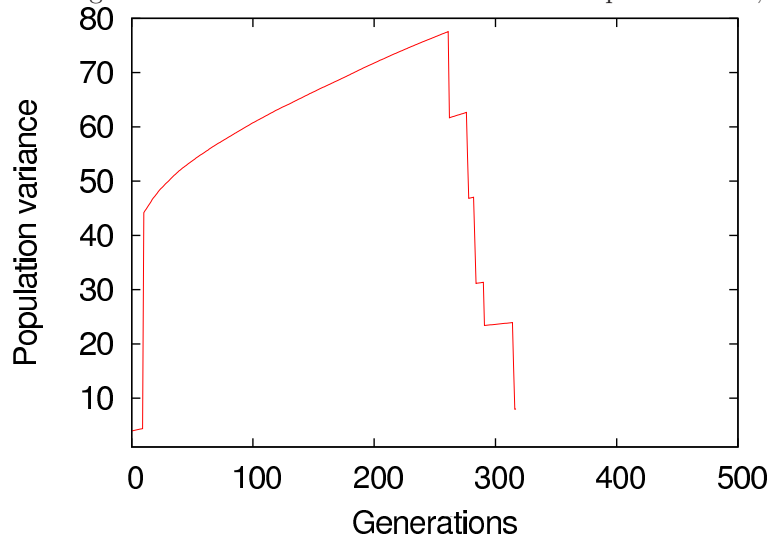
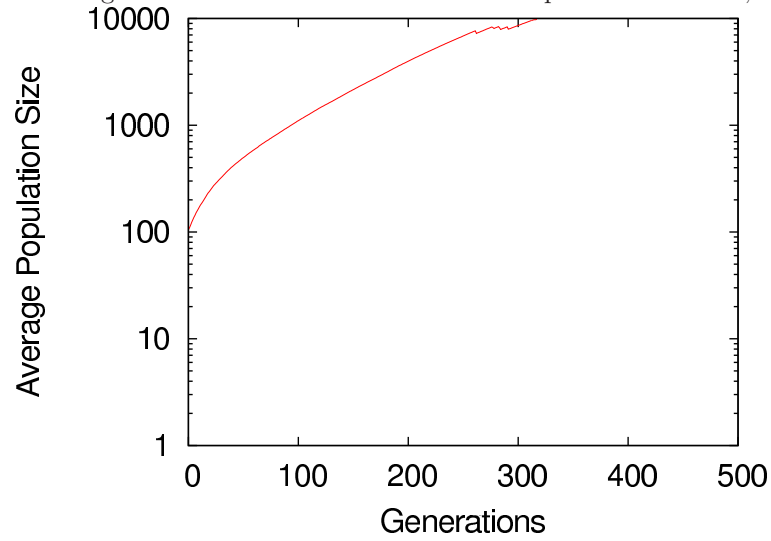


Figure 9: 10-run ASGA Verification: Population variance, 100 peak



### 5.4.3 Verifying Simple Genetic Algorithm results

The next step is to verify the results from the Simple Genetic Algorithm against the Spears problem instances in the ASGA paper. This is important as the SGA will be used as a benchmark with which to compare the ASGA later on in the experiments.

Table 9: SGA settings according to [6]

Parameter	Parameter Setting
Crossover	0.2
Mutation	0.05
Population	100
Parent selection	Tournament, size 2
Survival selection	Replace worst
Termination condition	10000 evaluations

What is surprising in 9 is that the type of selection isn't what is expected; a Simple GA would instead use Fitness-proportional parent selection and Generational survival selection. Thus, it can be stated that the 'Simple GA' with which the ASGA was compared isn't a Simple GA at all.

The results of the verification using the settings in Table 9 are shown in Table 10.

Table 10: GA-with-Tournament results.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success

Peaks	SGA / paper		GA-with-tournament / verified		
	Mean Best Fitness	Success Rate	Mean Best Fitness	Success Rate	Evaluations
1	1.0	100%	1.0	100%	5868
10	0.994	90%	1.0	100%	5688
100	0.988	26%	0.991	60%	8031

Interestingly, the verified GA with tournament selection performed better than expected, however the values in Table 12 don't deviate much from the results previously known. What does seem odd is the Success Rate of the 100-peak problem instance and the Average number of Evaluations for Success of the 10-peak problem instance, however these results show that the algorithms are comparable.

Now the question remains: how would a real Simple GA perform on the selected Spears problem instances? Spears himself in [31] used a GA very similar to the Simple GA as described previously. The results are vastly different from

those in 10. Using the original settings of Spears and a real Simple GA, we should be able to get more realistic results.

Table 11: SGA settings according to Spears in [31]

Parameter	Parameter Setting
Crossover	0.6
Mutation	0.001
Population	100
Parent selection	Fitness-proportional
Survival selection	Generational
Termination condition	10000 evaluations

Table 12: SGA results.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success

Peaks	SGA / verified		
	Mean Best Fitness	Success Rate	Evaluations
1	0.868	0%	10000
10	0.855	0%	10000
100	0.856	0%	10000

The results in Table 12 are similar to the results in [31] and are a more reasonable comparison for testing the ASGA against.

## 6 Experiments

Using the information obtained in the previous chapter from the screening and verification processes we can now calibrate the autonomous selection genetic algorithm.

### 6.1 Optimization: Autonomous Selection Calibration

**Hypothesis 2a:**

**Comparing algorithms** - Autonomous selection parameters found using meta-evolutionary algorithms will have a higher mean best fitness and success rate compared with previously-known parameters.

**Method:**

Calibrate the Autonomous Selection Genetic Algorithm using both the Meta-GA and REVAC methods, then determine if either provide a better mean best fitness compared to the verified results from the previous chapter.

We will first look into calibrating only the 4 ASGA-specific parameters using the Meta-GA and REVAC algorithms. After that we will also look into calibrating the ASGA with other parameters.

#### 6.1.1 Calibration using Meta Genetic Algorithm

Grefenstette in [12] used a population of 50, a crossover rate of 0.6 and a mutation rate of 0.001. We will use these same values for our Meta-GA, except that we double the population size to 100. The reason for this is that De Jong [16] determined that the population size should be between 50 and 100 and we assume Grefenstette chose 50 in order to limit the amount of evaluations and thus CPU-time required. In comparison to using only fitness-proportional survival selection Grefenstette also used Elitism. We will do the same.

Table 13: Meta-GA settings

Meta-GA Parameter	Meta-GA Setting
Crossover	0.6
Mutation	0.001
Population	100
Parent selection	Generational
Survival selection	Fitness-proportional + Elitism

As we have screened the search space manually beforehand, we have a guideline as to what degree of granularity is required. For the Multiplier-parameters, we picked a much smaller granularity than for the Shift-parameters.

For our first experiments we choose to only calibrate these AS-specific parameters, in later experiments we will also add more generic genetic algorithm parameters like crossover rate, mutation rate and the initial population size (which, as explained previously, varies after the first generation). For these parameters we will currently use the same settings as those used in the best-known ASGA.

Table 14: ASGA settings: 4 parameters are selected to be calibrated, the remaining parameters take the best-known value from [6]. The problem instances chosen are the same as in previous experiments.

Parameter to be calibrated	Parameter range	Granularity	Search space
Parent selection, Shift	0.0 to 6553.6	0.1	16-bit
Parent selection, Multiplier	-65.536 to 0.0	0.001	16-bit
Survivor selection, Shift	0.0 to 6553.6	0.1	16-bit
Survivor selection, Multiplier	-65.536 to 0.0	0.001	16-bit
Parameter not to be calibrated	Parameter setting		
Crossover	0.2		
Mutation	0.05		
Start Population	100		
Problem instance: Spears multi-modal problem generator			
Parameter	Problem instance setting		
Bit-string size	100		
Number of peaks	1, 10, 100		

In order to obtain comparable results to the previous experiments and verifications we will use the same problem instances, namely 3 different settings using the Spears problem generator.

The last step before running is to determine the termination conditions. Again, in order to obtain comparable results, we will use the same termination conditions as previously used for each of the ASGA runs, which were either 1000 generations or 10000 evaluations, depending on which happens first. 4 of these ASGA runs will have their best fitness averaged and returned to the Meta-GA; this then becomes the result of a single evaluation on the upper-level of the meta algorithm.

For the Meta-GA we will have to determine the termination conditions on our own. Grefenstette in [12] chose 20 meta-generations due to limited CPU resources, but 20 years of progress allows us to let our Meta-GA run as long as necessary. Still, we chose to keep the number of meta-evaluations to 3000 in order to limit the computation time required. The results obtained are shown in Table 15.

The last task is to run the ASGA using the parameters in Table 15 10 times in order to determine the average performance of these newly-found parameters. The results can be found in Table 16

Table 15: ASGA calibration using Meta-GA

Calibration Results			
Peaks	Max Fitness	Generations	Evaluations
1	0.988	29	3000
10	0.945	29	3000
100	0.988	29	3000
Best parameters found			
Peaks	Parent Selection - Shift Survival Selection - Shift	Parent Selection - Multiply Survival Selection - Multiply	
1		2603.6	-47.873
		1890.6	-0.004
10		188.7	-60.872
		3417.5	-0.008
100		3652.9	-15.051
		1703.5	-0.004

Table 16: Calibrated ASGA results using Meta-GA: 10 run average using best-found parameters.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Diversity of Population, Summed Population Variance.

Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.986	30%	4287	1.066	0.391
10	0.964	0%	10000	0.0472	0.630
100	0.960	10%	3697	1.633	0.368

### 6.1.2 Calibration using REVAC

As with the Meta-GA, a number of parameters had to be set for REVAC. For our experiments we use a slightly improved version of the original REVAC code, also used in [23]. However, the parameters used are thus also slightly adjusted and can be found in Table 17.

Table 17: Our initial REVAC settings

Parameter	Parameter Setting
Pool size	300
Best size	200
Smoothing	2

For the ASGA, we will use the same parameter ranges for the parameters we want to calibrate as in Table 14, as well as the parameter values for those parameters we are not calibrating. The problem instances we calibrate on are the same 1, 10 and 100 peak Spears problem instances as in Table 14. The results of the ASGA calibration using REVAC is shown in Table 18.

Table 18: ASGA calibration using REVAC

Calibration results			
Peaks	Max Fitness	Generations	Evaluations
1	0.880	29	3000
10	0.860	29	3000
100	0.840	29	3000
Best parameters found			
Peaks	Parent Selection - Shift Survival Selection - Shift	Parent Selection - Multiply Survival Selection - Multiply	
1		5022.0	-54.951
		4753.9	-0.0131
10		1361.1	-7.274
		4182.5	-0.0131
100		5208.8	-56.944
		4656.3	-0.0131

Again, the parameters found are put to the test. The results are shown in Table 19.

We have now calibrated the ASGA using both meta-evolutionary algorithms. We have also verified our own results by taking the average of 10 ASGA runs using the calibrated parameters. In **Chapter 7: Analysis** we will combine our results with the results obtained using the best known parameters and determine if our hypothesis is correct.

Table 19: Calibrated ASGA results using REVAC: 10-run average using best-found parameters.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Diversity of Population, Summed Population Variance.

Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.879	0%	10000	0.0455	0.843
10	0.896	0%	10000	0.0335	0.810
100	0.867	0%	10000	0.0307	0.790

## 6.1.3 ASGA Calibration Plots, Spears 1 Peak

Figure 10: Meta-GA calibrating ASGA, 1 peak. Best and Average Fitness

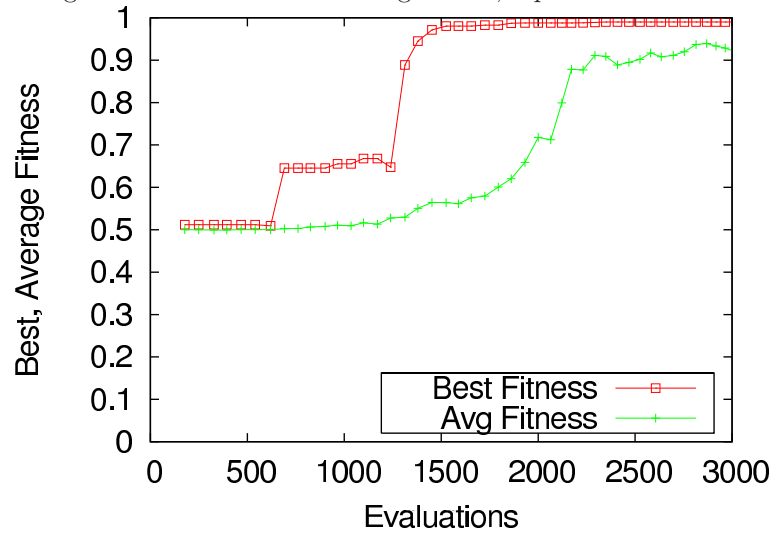


Figure 11: REVAC calibrating ASGA, 1 peak. Best and Average Fitness

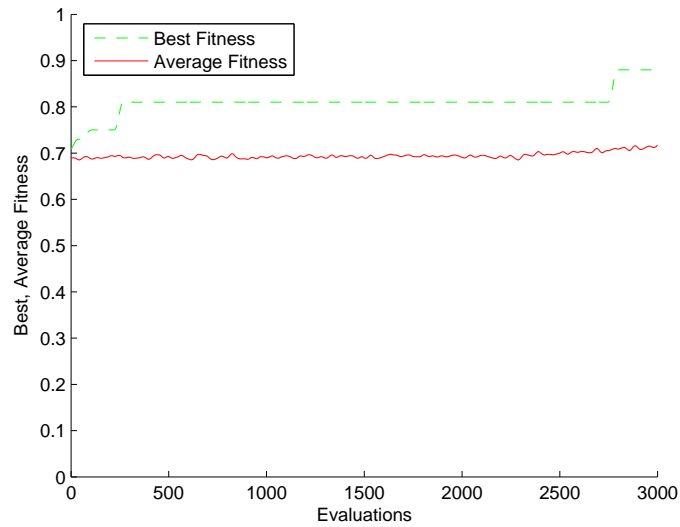


Figure 12: 10-run Meta-GA calibrated ASGA: Mean Best Fitness, 1 peak

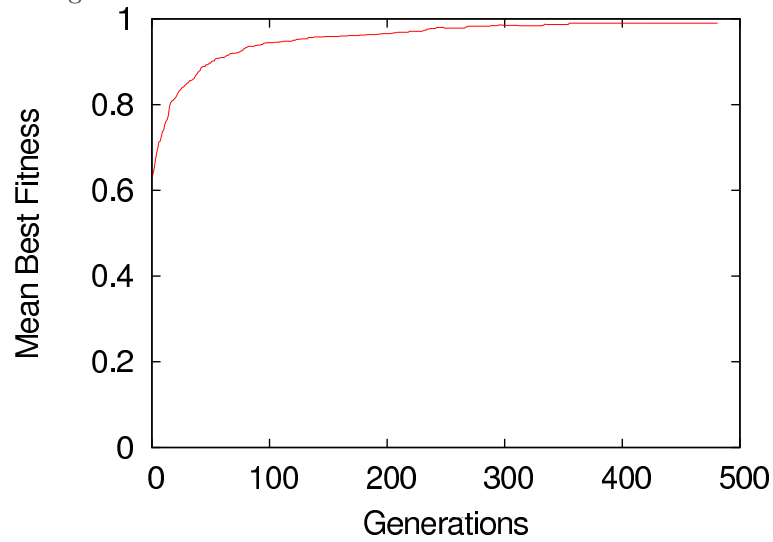


Figure 13: 10-run REVAC calibrated ASGA: Mean Best Fitness, 1 peak. Population explosion causes premature termination.

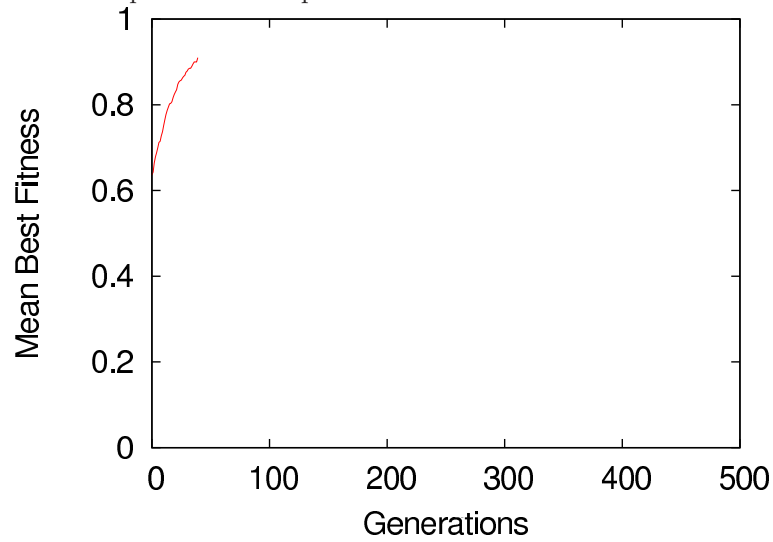


Figure 14: 10-run Meta-GA calibrated ASGA: Mean Population Size, 1 peak

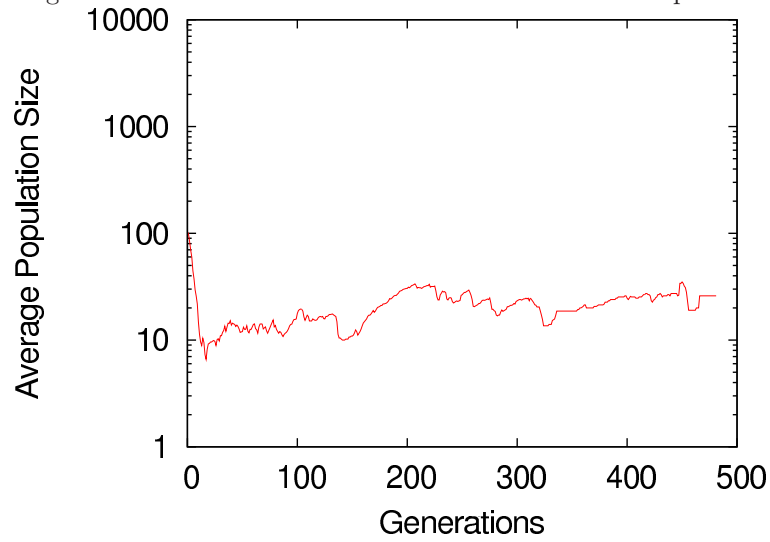


Figure 15: 10-run REVAC calibrated ASGA: Mean Population Size, 1 peak. Population explosion causes premature termination.

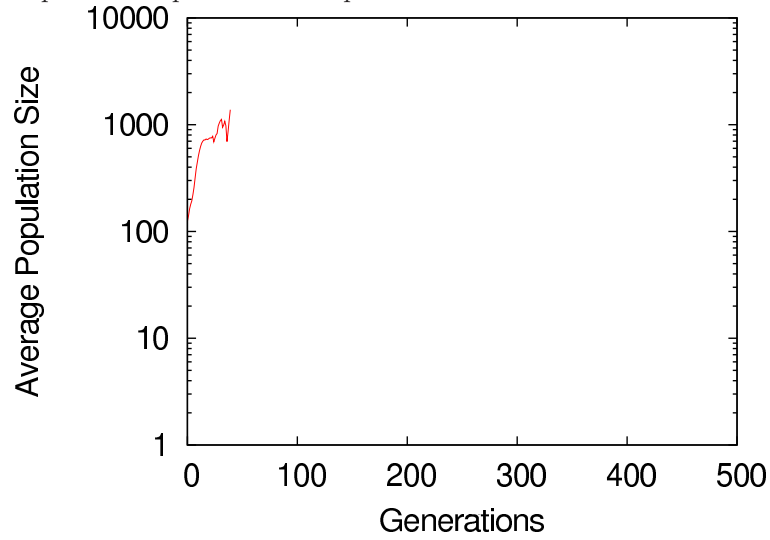


Figure 16: 10-run Meta-GA calibrated ASGA: Population variance, 1 peak

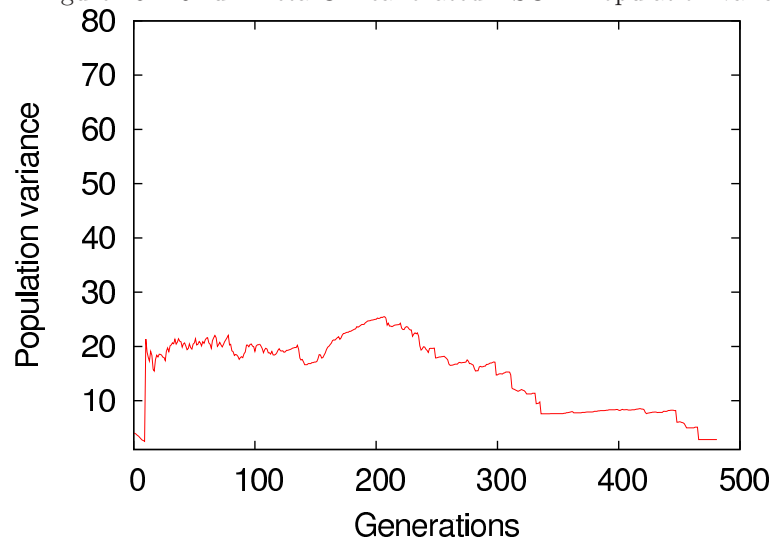


Figure 17: 10-run REVAC calibrated ASGA: Population variance, 1 peak. Population explosion causes premature termination.

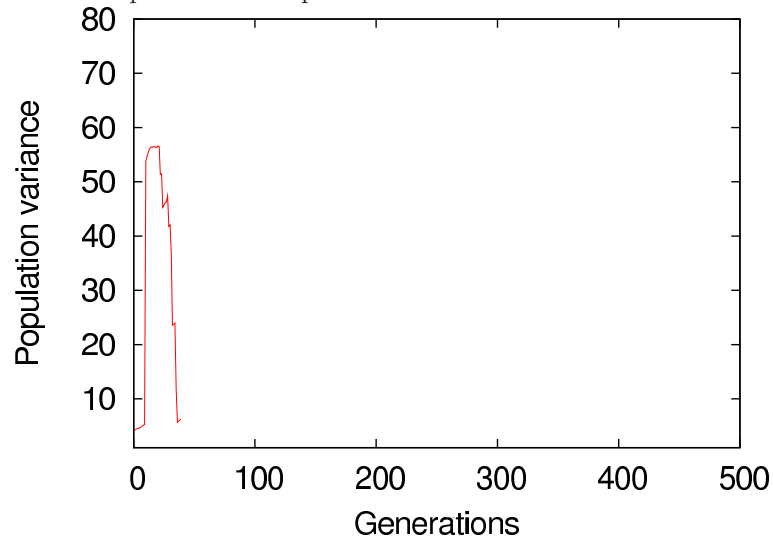


Figure 18: Meta-GA calibrating ASGA, 10 peak. Best and Average Fitness

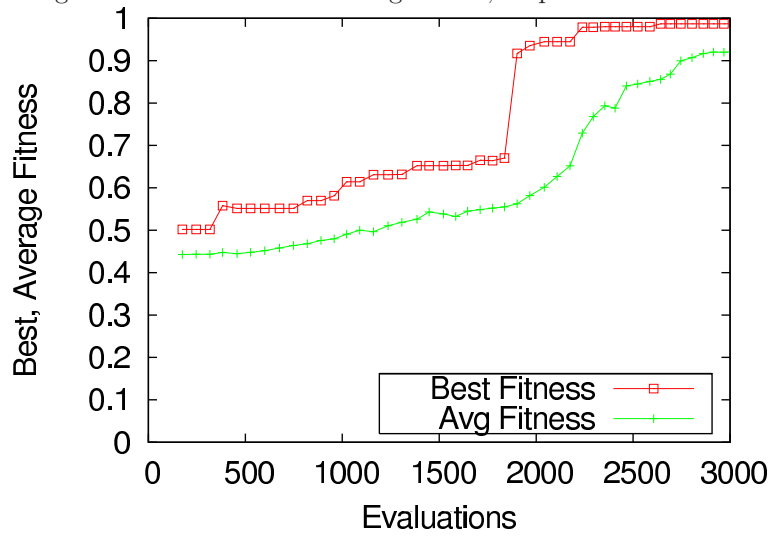


Figure 19: REVAC calibrating ASGA, 10 peak. Best and Average Fitness

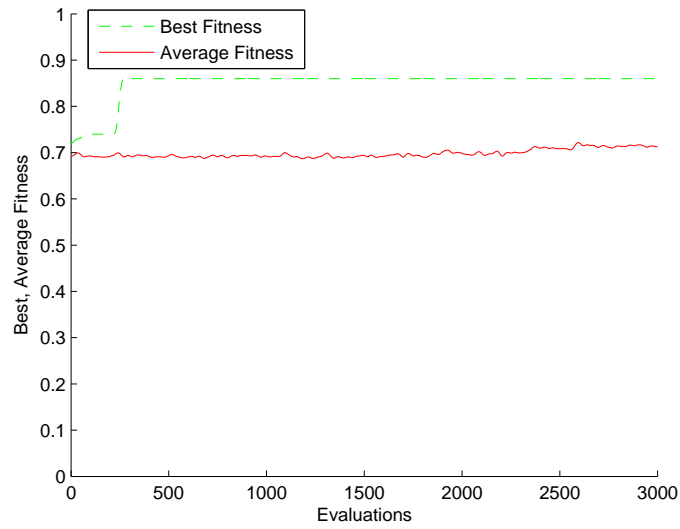


Figure 20: 10-run Meta-GA calibrated ASGA: Mean Best Fitness, 10 peak. Higher population causes premature termination.

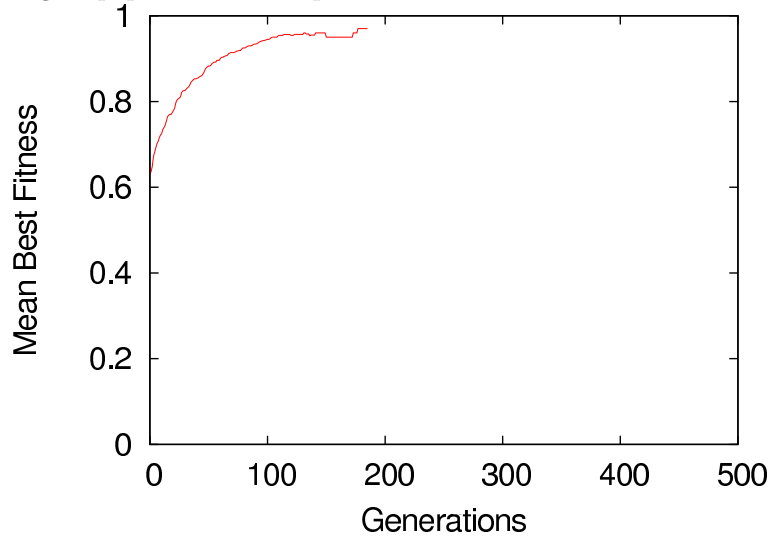


Figure 21: 10-run REVAC calibrated ASGA: Mean Best Fitness, 10 peak. Population explosion causes premature termination.

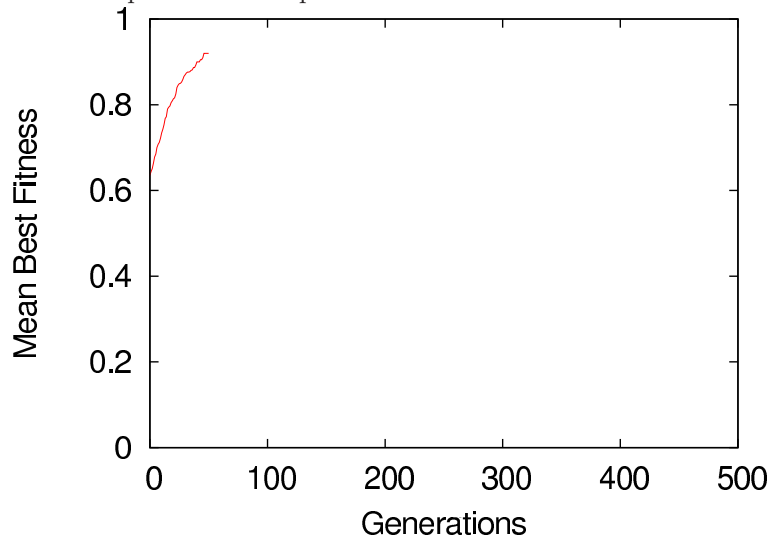


Figure 22: 10-run Meta-GA calibrated ASGA: Mean Population Size, 10 peak. Higher population causes premature termination

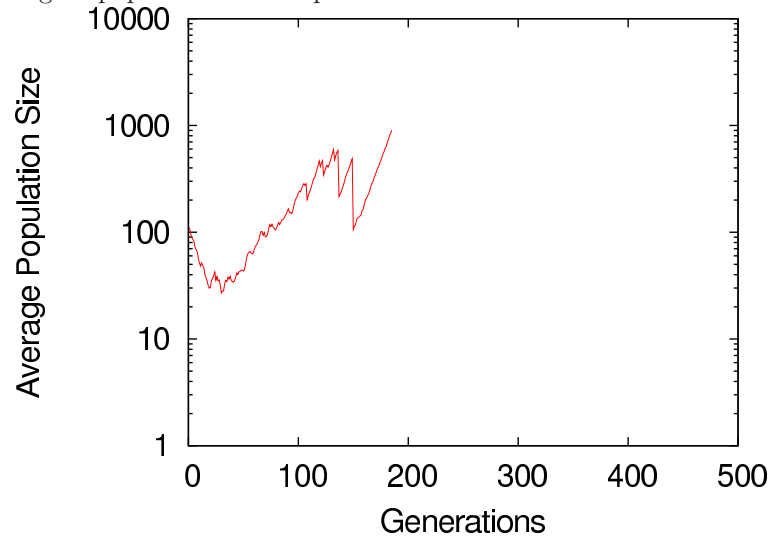


Figure 23: 10-run REVAC calibrated ASGA: Mean Population Size, 10 peak. Population explosion causes premature termination.

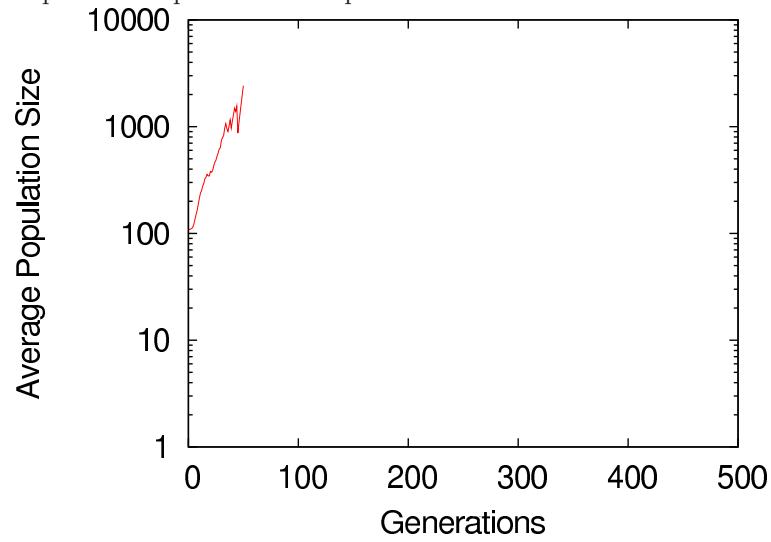


Figure 24: 10-run Meta-GA calibrated ASGA: Population variance, 10 peak. Higher population causes premature termination.

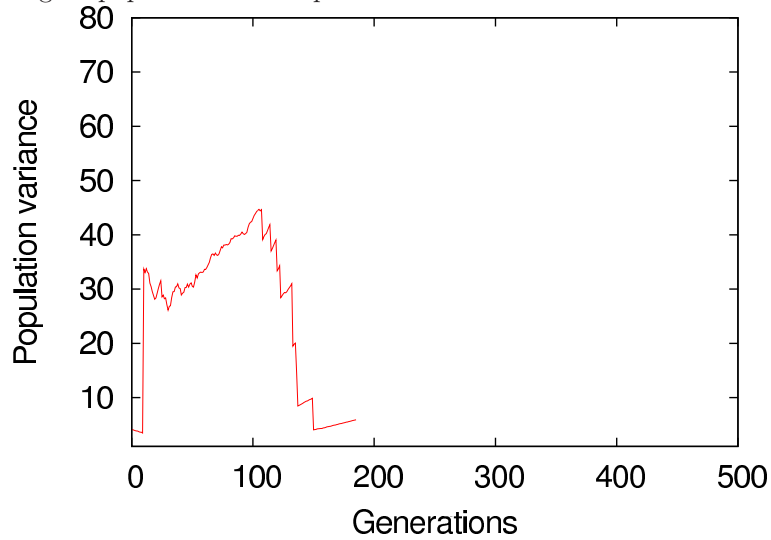


Figure 25: 10-run REVAC calibrated ASGA: Population variance, 10 peak. Population explosion causes premature termination.

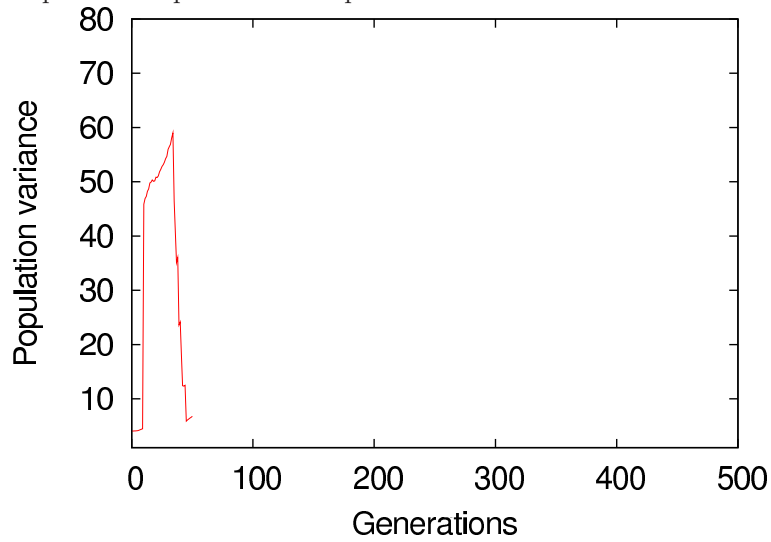


Figure 26: Meta-GA calibrating ASGA, 100 peak. Best and Average Fitness

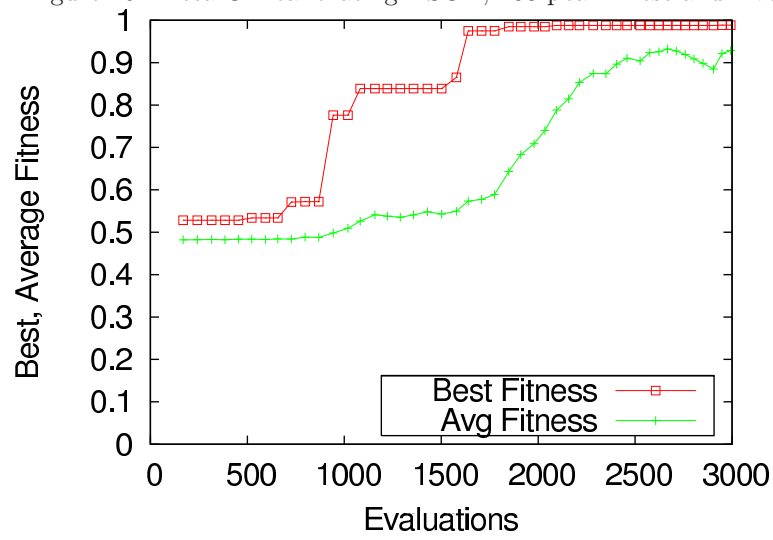


Figure 27: REVAC calibrating ASGA, 100 peak. Best and Average Fitness

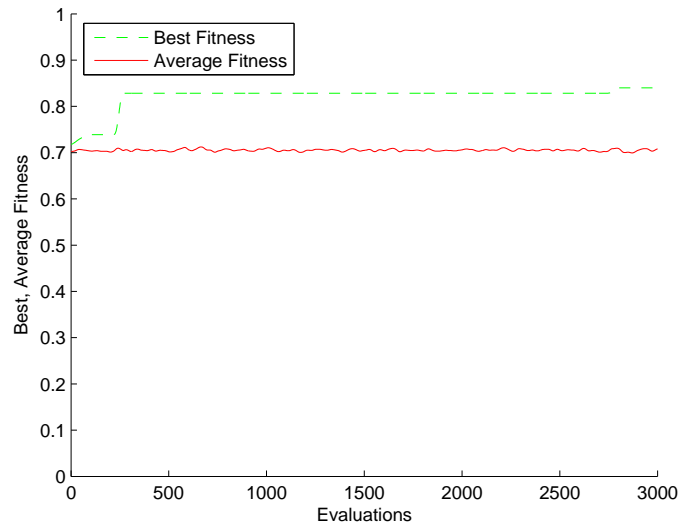


Figure 28: 10-run Meta-GA calibrated ASGA: Mean Best Fitness, 100 peak

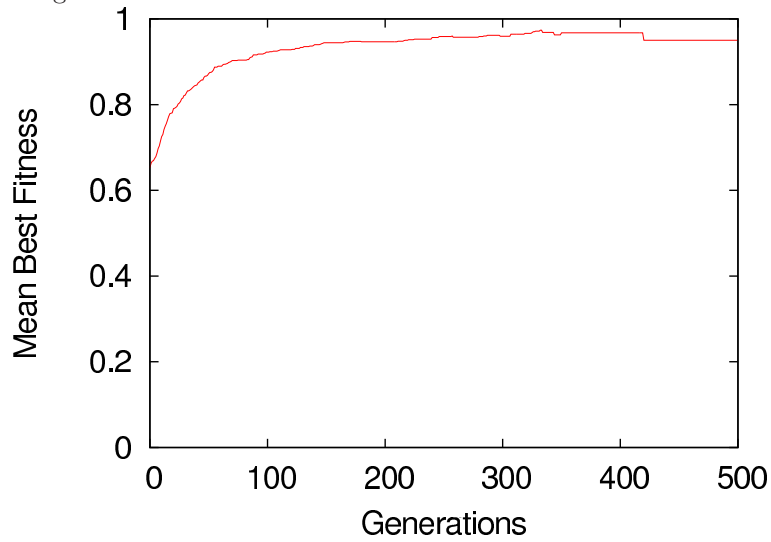


Figure 29: 10-run REVAC calibrated ASGA: Mean Best Fitness, 100 peak. Population explosion causes premature termination.

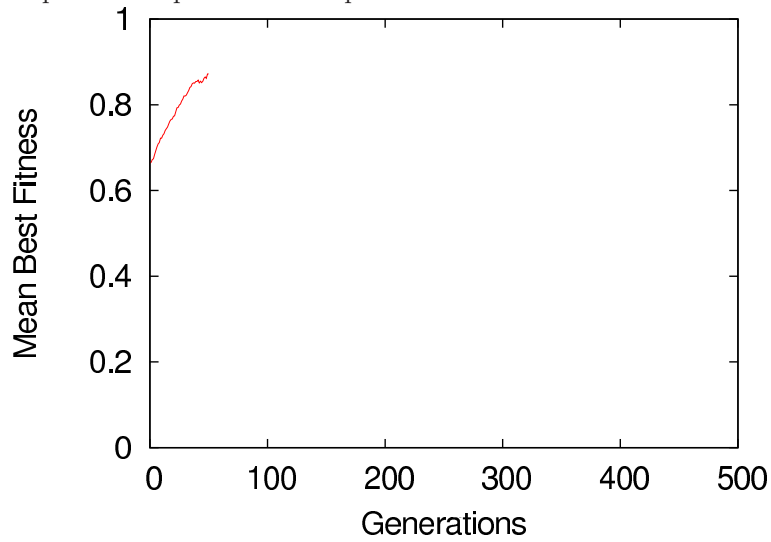


Figure 30: 10-run Meta-GA calibrated ASGA: Mean Population Size, 100 peak

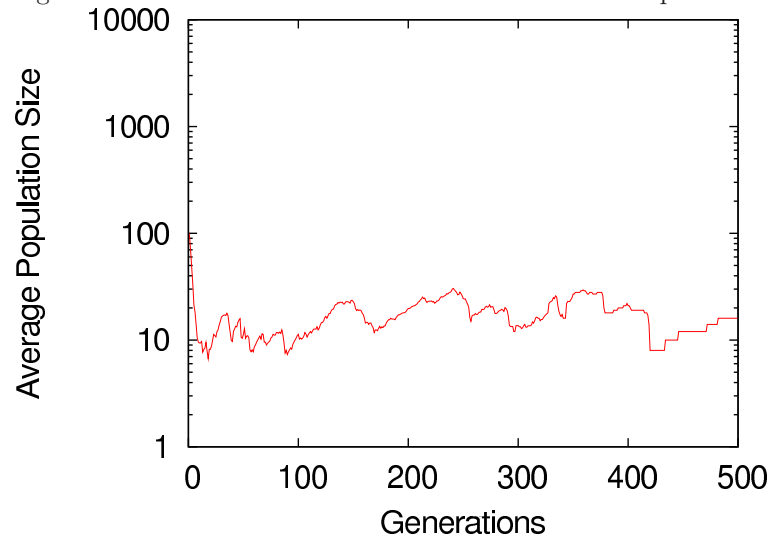


Figure 31: 10-run REVAC calibrated ASGA: Mean Population Size, 100 peak. Population explosion causes premature termination.

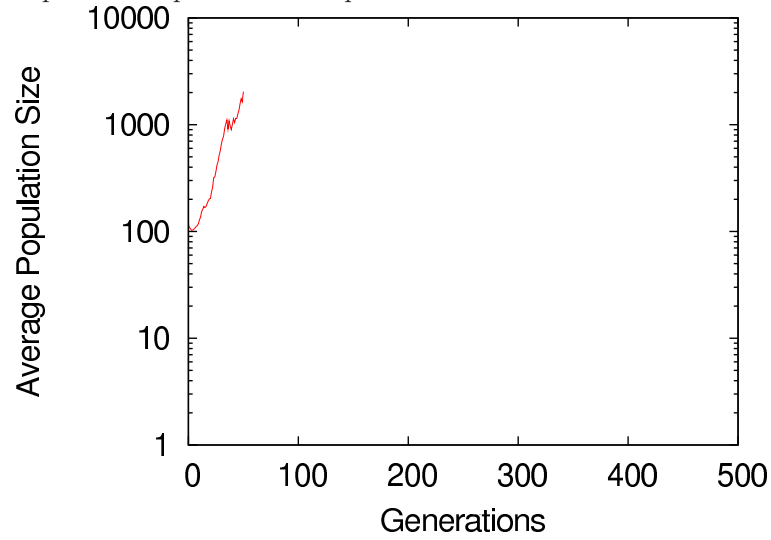


Figure 32: 10-run Meta-GA calibrated ASGA: Population variance, 100 peak

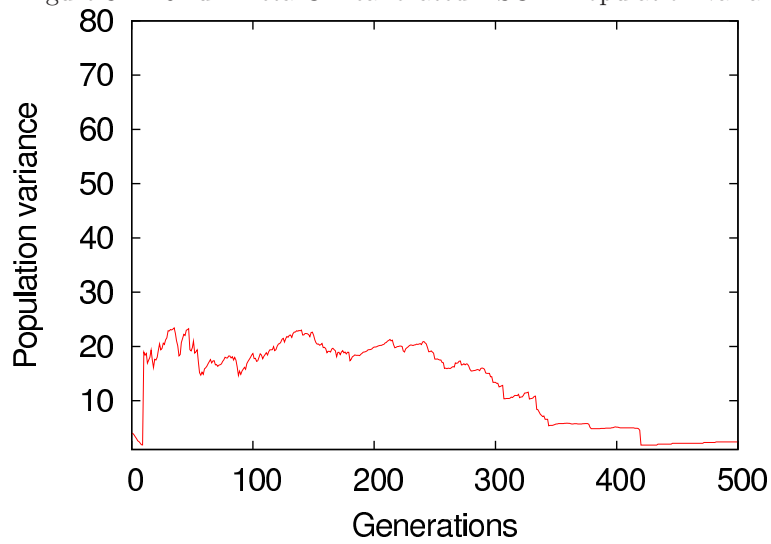
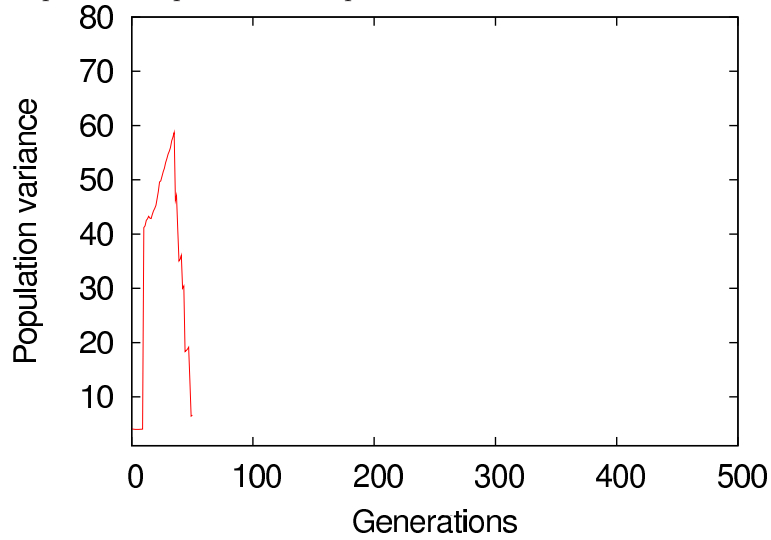


Figure 33: 10-run REVAC calibrated ASGA: Population variance, 100 peak. Population explosion causes premature termination.



### 6.1.4 Calibration using REVAC, second parameter combination

Compared to the Meta-GA results, the results from the first ASGA calibrations by REVAC weren't very good. However, during the experiments a second combination of parameters was found. These parameters are described in [24] and, according to the author, these parameters should provide better performance. This second combination of parameters is shown in Table 20.

Table 20: REVAC settings according to [24]

Parameter	Parameter Setting
Pool size	100
Best size	50
Smoothing	5

We run the same experiments as in the previous section, without any other modifications. The results are shown in Table 21.

Table 21: ASGA calibration using REVAC, second parameter combination

Calibration results			
Peaks	Max Fitness	Generations	Evaluations
1	0.990	29	3000
10	0.840	29	3000
100	0.980	29	3000
Best parameters found			
Peaks	Parent Selection - Shift Survival Selection - Shift	Parent Selection - Multiply Survival Selection - Multiply	
1		244.4	-11.390
		5943.5	-0.000
10		4568.5	-51.839
		5474.9	-0.0131
100		1740.0	-1.173
		855.9	-0.007

By the looks of it, the calibration went very well. The results of the ASGA 10-run averages are shown in Table 22.

The results in Table 22 are odd; the 1- and 100-peak runs are worse than we would expect from the initial calibration. We have rerun these results to make sure that they are correct, however all show the same behavior. One possible explanation is that REVAC easily gets 'lucky' using the second parameter combination. One ASGA run gives a REVAC-entity a high fitness, which REVAC then retains, however the parameters of that entity on average only give poor results. For our further experiments we will however stay with the original REVAC parameters, as they provide more consistent results.

Table 22: Calibrated ASGA results using REVAC with second parameter combination: 10-run average using best-found parameters.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Diversity of Population, Summed Population Variance.

Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.712	0%	10000	0.0	0.609
10	0.897	0%	10000	0.0355	0.814
100	0.693	0%	10000	0.0	0.612

### 6.1.5 7-Parameter ASGA Calibration using Meta-GA

Although calibrating the 4 parameters in Table 14 is the main goal, it is also interesting to also look at the other 3 parameters we haven't calibrated so far. The crossover rate, mutation rate and population size will be looked at when discussing calibrating the Simple GA in the next section, however the Autonomous Selection Genetic Algorithm also has these parameters (with the exception that the population size determines the size at the start of the run).

The 3 additional parameters to be calibrated are shown in Table 23, bringing the total number of parameters which we calibrate to 7. For this experiment we will only be using a single problem instance of the Spears multi-modal problem generator.

Table 23: ASGA settings: All 7 parameters are selected to be calibrated. This time we only use one problem instance.

Parameter to be calibrated	Parameter range	Granularity	Search space
Parent selection, Shift	0.0 to 6553.6	0.1	16-bit
Parent selection, Multiplier	-65.536 to 0.0	0.001	16-bit
Survivor selection, Shift	0.0 to 6553.6	0.1	16-bit
Survivor selection, Multiplier	-65.536 to 0.0	0.001	16-bit
Crossover	0.0 to 1.0	0.001	10-bit
Mutation	0.0 to 1.0	0.001	10-bit
Start Population	1 to 1024	1	10-bit
Problem instance: Spears multi-modal problem generator			
Parameter	Problem instance setting		
Bit-string size	100		
Number of peaks	1		

The results from the calibration using the Meta-GA are shown in Table 24.

It is clear from the statistics in Table 24 that the Meta-GA isn't calibrating properly, as it shows a very high mutation rate. Even so, we ran 10 runs using these parameters and obtained the results in Table 25.

Table 24: 7-Parameter ASGA calibration using Meta-GA

Calibration results			
Peaks	Max Fitness	Generations	Evaluations
1	0.760	29	3000
Best parameters found			
Peaks	Parent Selection - Shift Survival Selection - Shift	Parent Selection - Multiply Survival Selection - Multiply	
1		6100.9 5634.4	-1.483 -4.201
Best parameters found			
Peaks	Crossover rate	Mutation rate	Start Population
1	0.401	0.659	337

Table 25: 7-Parameter Calibrated ASGA results using Meta-GA: 10-run average using best-found parameters.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Diversity of Population, Summed Population Variance.

Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.683	0%	10000	0.0093	0.940

### 6.1.6 7-Parameter ASGA Calibration using REVAC

The results from the Meta-GA weren't good at all. Can REVAC perform better? Again, we calibrate the ASGA using the settings in Table 23 using REVAC. The results of the calibration are shown in 26.

Table 26: 7-Parameter ASGA calibration using REVAC

Calibration results			
Peaks	Max Fitness	Generations	Evaluations
1	0.870	29	3000
Best parameters found			
Peaks	Parent Selection - Shift Survival Selection - Shift	Parent Selection - Multiply Survival Selection - Multiply	
1		1433.9 6539.8	-58.098 -0.013
Best parameters found			
Peaks	Crossover rate	Mutation rate	Start Population
1	0.767	0.965	400

Again the calibration seems to have backfired, looking at the impossibly high mutation rate. Even so, we try the parameters on a 10-run ASGA average, with

the result shown in Table 27.

Table 27: 7-Parameter Calibrated ASGA results using REVAC: 10-run average using best-found parameters.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Diversity of Population, Summed Population Variance.

Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.713	0%	10000	3.073	0.612

It is clear that, within the current limits of 3000 meta-evaluations, both REVAC and the Meta-GA can't properly calibrate the ASGA with the increased search space. The most likely reason for this is that the search space has become too large for the meta-algorithm to sufficiently cope with.

## 6.1.7 7-Parameter ASGA Calibration Plots

Figure 34: Meta-GA calibrating 7-parameter ASGA, 1 peak. Best and Average Fitness

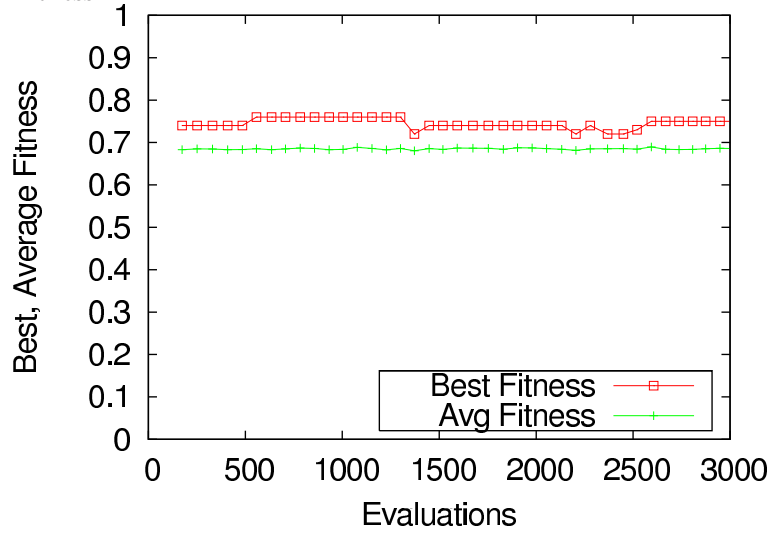


Figure 35: REVAC calibrating 7-parameter ASGA, 1 peak. Best and Average Fitness

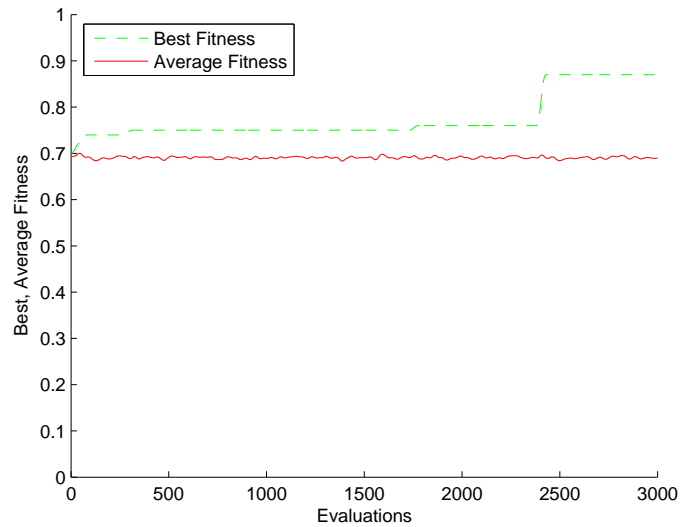


Figure 36: 10-run Meta-GA calibrated 7-parameter ASGA: Mean Best Fitness, 1 peak. Population explosion causes premature termination (after 5 generations)

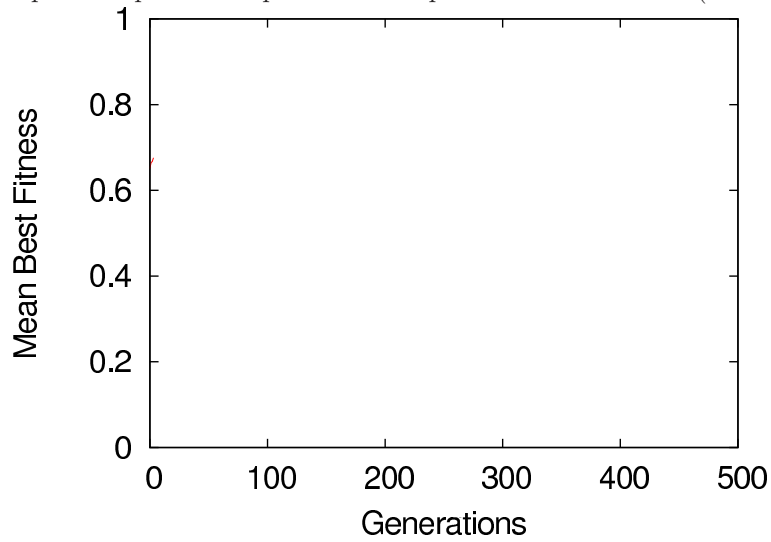
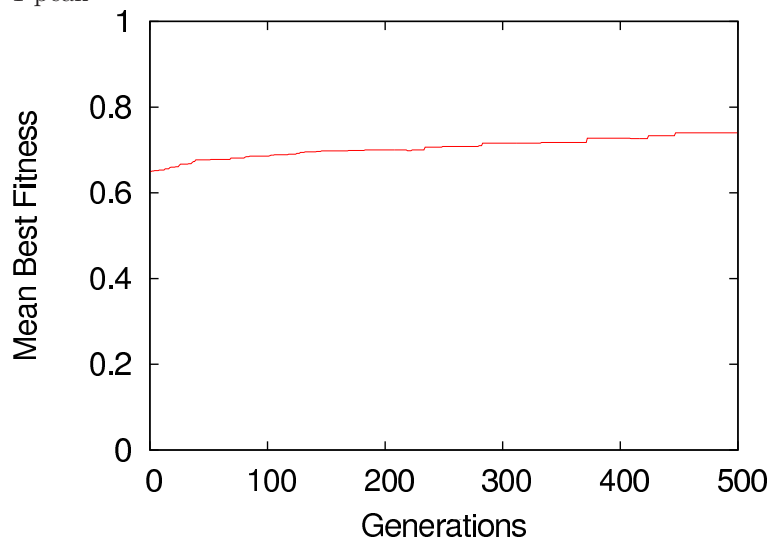


Figure 37: 10-run REVAC calibrated 7-parameter ASGA: Mean Best Fitness, 1 peak



## 6.2 Optimization: Calibrating SGA

### Hypothesis 2b:

**Simple GA** - Parameters found using meta-evolutionary algorithms will not cause a significantly higher mean best fitness than when best-practice parameters are used with the simple genetic algorithm.

### Method:

Calibrate a Simple Genetic Algorithm using both the Meta-GA and REVAC methods.

The next experimentation consisted of calibrating the Simple Genetic Algorithm instead of the Autonomous Selection Genetic Algorithm, using problem instances generated by Spears multi-modal problem generator. By doing so we are trying to find the most optimal parameters for crossover, mutation and population size for the Simple Genetic Algorithm. Again, we will do experiments using both the Meta-GA and the REVAC methods.

### 6.2.1 Calibration using Meta Genetic Algorithm

The settings of the Meta-GA will be the same as in Table 13.

The search space for the Simple GA within we will calibrate is a lot easier to determine, compared to the ASGA search space. Crossover and mutation are both between 0.0 and 1.0. The population size, theoretically being a value anywhere from 1 to inf, is slightly more difficult, however we will limit this to a reasonable range.

Table 28: Simple GA settings: We calibrate the crossover, mutation and population parameters, while keeping the methods of selection constant

Parameter to be calibrated	Parameter range	Granularity	Search space
Crossover	0.0 to 1.0	0.0000153	16-bit
Mutation	0.0 to 1.0	0.0000153	16-bit
Population	1 to 1024	1	10-bit
Parameter not to be calibrated	Parameter setting		
Parent selection	Generational		
Survival selection	Fitness-proportional		
Problem instance: Spears multi-modal problem generator			
Parameter	Problem instance setting		
Bit-string size	100		
Number of peaks	1, 10, 100		

Although many types of selection are possible, the Simple GA uses the two types of selection that the Meta-GA also uses, namely generational parent selection and fitness-proportional survival selection. Elitism isn't used in the Simple GA, however. The chosen problem instances are again the same problem instances from the previous experiment.

The same termination conditions were used as with the ASGA experiments above. The Simple GA has either 1000 generations or 10000 evaluations in order to find the optimal solution for the Spears problem, while the Meta-GA has 3000 evaluations to find the best Simple GA. The results of the calibrations and the parameters found are shown in Table 29. The 10-run averages using the obtained parameters is shown in Table 30.

Table 29: SGA calibration using Meta-GA

Calibration results			
Peaks	Max Fitness	Evaluations	
1	0.999	2953	
10	0.857	2920	
100	0.999	2984	
Best Parameters found			
Peaks	Crossover	Mutation	Population size
1	0.61	0.00043	38
10	0.60	0.005	120
100	0.43	0.00060	36

Table 30: Calibrated SGA results using Meta-GA: 10 run average using best-found parameters.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success

Peaks	Mean Best Fitness	Success Rate	Evaluations
1	0.957	10%	5272
10	0.782	0%	10000
100	0.941	0%	6722

### 6.2.2 Calibration using REVAC

Again we will also calibrate the Simple GA using REVAC. The same REVAC-parameters will be used as for the ASGA calibration, defined in Table 17. The problem instances specified in the previous section in Table 28 remain the same, as do the termination conditions for the algorithms. The results of the calibration are shown in Table 31, the results of the 10-run averages using the found parameters is shown in Table 32.

We have now calibrated the Simple GA using both meta-evolutionary algorithms. Again, we have also verified our own results by taking the average of

Table 31: SGA calibration using REVAC

Calibration results			
Peaks	Max Fitness	Evaluations	
1	1.0	2625	
10	0.985	2926	
100	0.985	2428	
Best parameters found			
Peaks	Crossover	Mutation	Population size
1	0.51	0.0004	55
10	0.30	0.0007	30
100	0.45	0.0003	70

Table 32: Calibrated SGA results using REVAC: 10 run average using best-found parameters.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success

Peaks	Mean Best Fitness	Success Rate	Evaluations
1	0.969	10%	8775
10	0.922	0%	5997
100	0.944	10%	9533

10 Simple GA runs using the calibrated parameters. In **Chapter 7: Analysis** we will combine our results with the results obtained using the best known parameters and determine if our hypothesis is correct.

## 6.2.3 SGA Calibration Plots

Figure 38: Meta-GA calibrating SGA, 1 peak. Best and Average Fitness

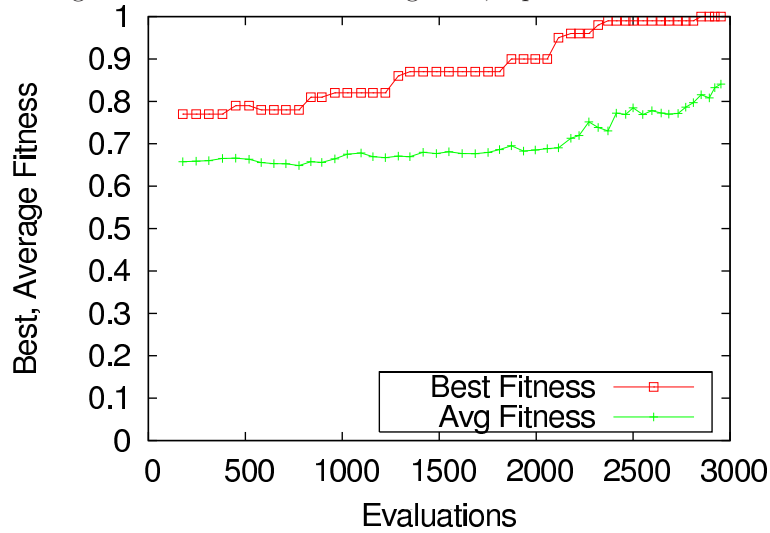


Figure 39: REVAC calibrating SGA, 1 peak. Best and Average Fitness

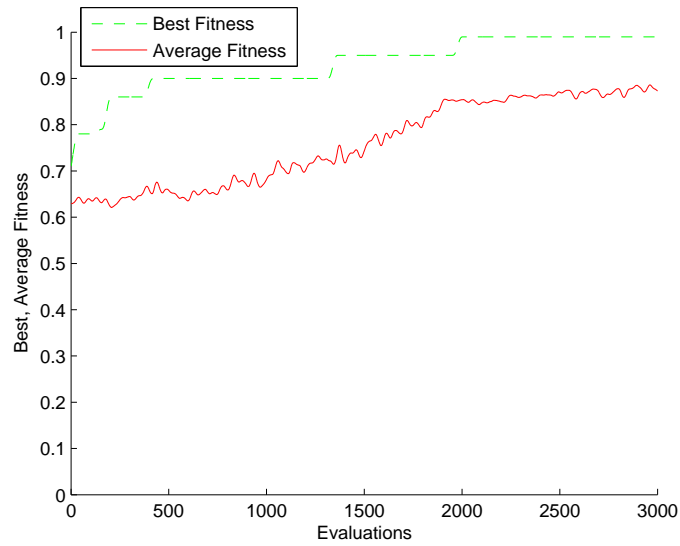


Figure 40: Meta-GA calibrating SGA, 10 peak. Best and Average Fitness

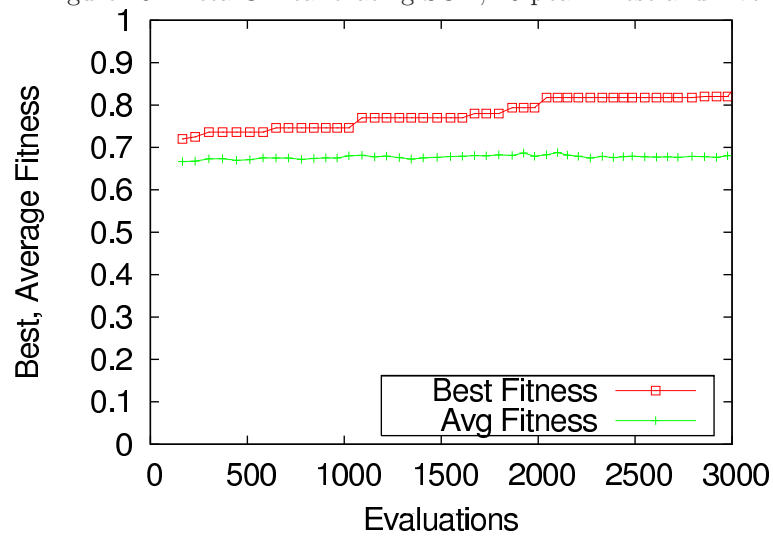


Figure 41: REVAC calibrating SGA, 10 peak. Best and Average Fitness

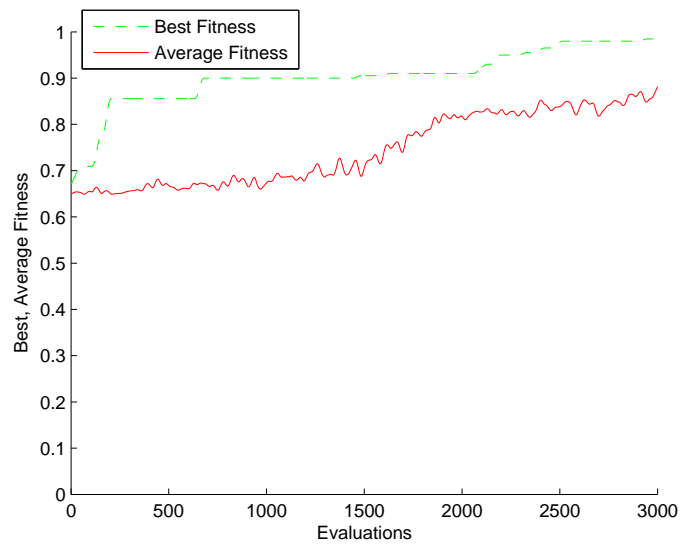


Figure 42: Meta-GA calibrating SGA, 100 peak. Best and Average Fitness

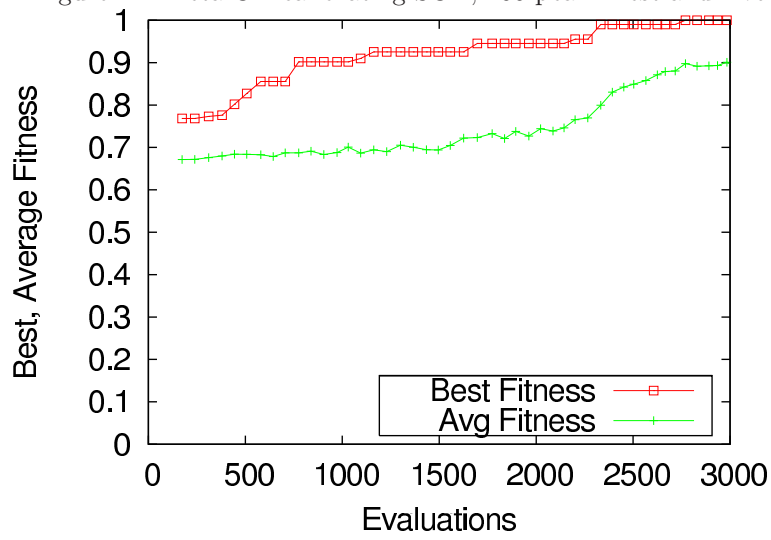
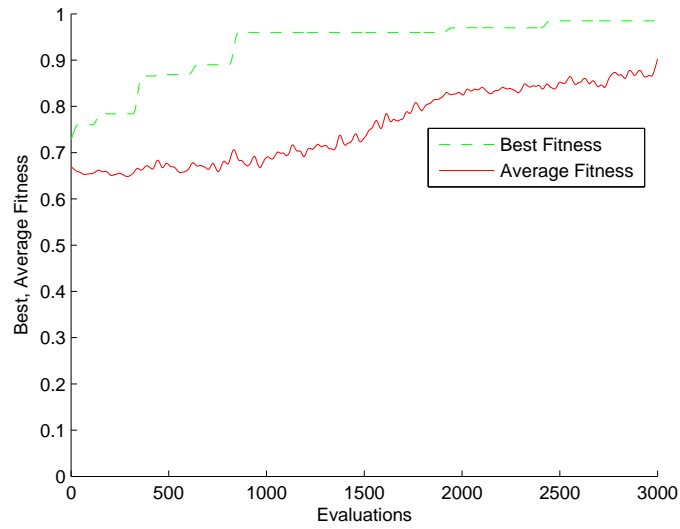


Figure 43: REVAC calibrating SGA, 100 peak. Best and Average Fitness



### 6.3 Robustness: ASGA and Stability

Although having a high fitness is the prime goal of our experiments, we must also look at the stability of these results. How do our results fare against slightly different problem instances?

**Hypothesis 1b:**

**Sensitivity and Stability** - Parameters of the autonomous selection genetic algorithm are sensitive, in that small changes to either the parameters of autonomous selection or the underlying fitness landscape of the problem instance can lead to an ineffective algorithm.

**Method:**

Perform ASGA runs using optimized parameters on different problem instances.

For each of the best parameter-combinations found using both meta-algorithms we will perform experiments with the other peaks and with the amount of peaks multiplied by five. The performance will give us an indication how stable the parameters we found are.

Table 33: Adjusted problem instances

Problem instance: Spears multi-modal problem generator	
Spears Parameter	Problem instance setting
Bit-string size	100
Number of peaks	1, 5, 10, 50, 100, 500

The ASGA parameters used can be found in Table 15 and Table 18. As with the previous experiments we use a 10-run average. The results for the Meta-GA are shown in Table 34 and those for REVAC are shown in Table 35.

Table 34: ASGA using parameters calibrated by Meta-GA on other problem instances.

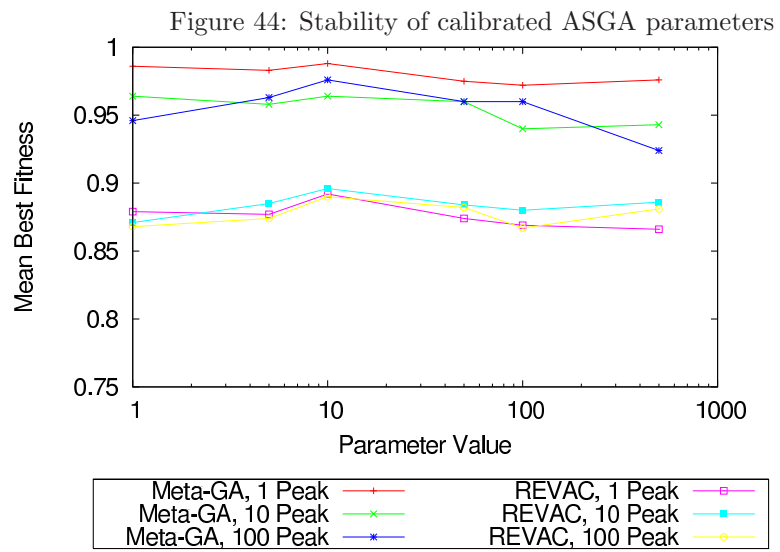
Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Diversity of Population, Summed Population Variance.

Results using Meta-GA-calibrated parameters from 1 peak calibration, on ASGA					
Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.986	30%	4287	1.066	0.391
5	0.983	20%	4169	1.226	0.371
10	0.988	40%	4669	1.147	0.387
50	0.975	10%	4792	1.141	0.388
100	0.972	0%	4267	1.554	0.372
500	0.976	0%	4022	1.379	0.363
Average	0.980	17%	4368	1.252	0.379
Results using Meta-GA-calibrated parameters from 10 peak calibration, on ASGA					
Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.964	0%	10000	0.057	0.650
5	0.958	0%	10000	0.054	0.637
10	0.964	0%	10000	0.047	0.630
50	0.960	0%	10000	0.047	0.617
100	0.940	0%	10000	0.044	0.640
500	0.943	0%	10000	0.050	0.630
Average	0.955	0%	10000	0.050	0.634
Results using Meta-GA-calibrated parameters from 100 peak calibration, on ASGA					
Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.946	0%	2600	0.931	0.364
5	0.963	0%	3517	1.315	0.375
10	0.976	20%	3226	1.036	0.346
50	0.960	10%	3134	1.358	0.351
100	0.960	10%	3697	1.633	0.368
500	0.924	0%	2294	0.907	0.330
Average	0.955	7%	3078.0	1.197	0.356

Table 35: ASGA using parameters calibrated by REVAC on other problem instances.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Diversity of Population, Summed Population Variance.

Results using REVAC-calibrated parameters from 1 peak calibration, on ASGA					
Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.879	0%	10000	0.046	0.842
5	0.877	0%	10000	0.035	0.785
10	0.892	0%	10000	0.033	0.776
50	0.874	0%	10000	0.034	0.752
100	0.869	0%	10000	0.031	0.754
500	0.866	0%	10000	0.035	0.750
Average	0.876	0%	10000	0.036	0.777
Results using REVAC-calibrated parameters from 10 peak calibration, on ASGA					
Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.871	0%	10000	0.049	0.804
5	0.885	0%	10000	0.035	0.792
10	0.896	0%	10000	0.034	0.810
50	0.884	0%	10000	0.035	0.753
100	0.880	0%	10000	0.030	0.749
500	0.886	0%	10000	0.034	0.735
Average	0.884	0%	10000	0.036	0.774
Results using REVAC-calibrated parameters from 100 peak calibration, on ASGA					
Peaks	Mean Best Fitness	Success Rate	Evaluations	Population Diversity	Population Variance
1	0.868	0%	10000	0.047	0.807
5	0.874	0%	10000	0.042	0.797
10	0.890	0%	10000	0.033	0.774
50	0.882	0%	10000	0.032	0.751
100	0.867	0%	10000	0.031	0.790
500	0.881	0%	10000	0.032	0.737
Average	0.877	0%	10000	0.036	0.776



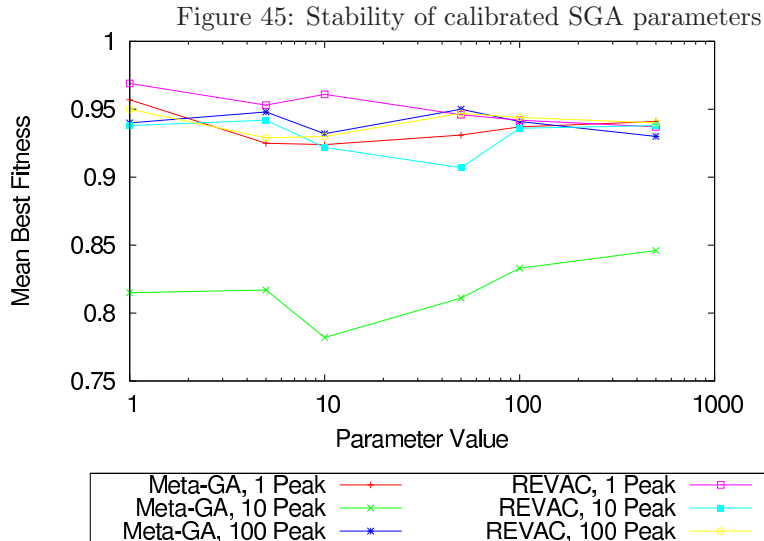
## 6.4 Robustness: SGA and Stability

For consistency, we also look at the stability of our SGA results. For this experiment we don't have a hypothesis to support or disprove. We will however determine how the calibrated SGA parameters work on the same problem instances as in the previous section.

Table 36: Adjusted problem instances for Simple Genetic Algorithm

Problem instance: Spears multi-modal problem generator	
Spears Parameter	Problem instance setting
Bit-string size	100
Number of peaks	1, 5, 10, 50, 100, 500

The SGA parameters used can be found in Table 29 and Table 31. Again, we use 10-run averages. The results for the Meta-GA are shown in Table 37 and those for REVAC are shown in Table 38.



In this section we have obtained generalized results using the parameters obtained by our two meta-evolutionary algorithms. In the next chapter, **Chapter 7: Analysis**, we will continue with these results in order to determine if our hypotheses are correct.

Table 37: SGA using parameters calibrated by Meta-GA on other problem instances.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Diversity of Population, Summed Population Variance.

Results using Meta-GA-calibrated parameters from 1 peak calibration, on SGA			
Peaks	Mean Best Fitness	Success Rate	Evaluations
1	0.957	10%	5272
5	0.925	0%	5211
10	0.924	0%	5279
50	0.931	0%	4762
100	0.937	0%	5154
500	0.941	0%	4999
Average	0.936	2%	5113
Results using Meta-GA-calibrated parameters from 10 peak calibration, on SGA			
Peaks	Mean Best Fitness	Success Rate	Evaluations
1	0.815	0%	10000
5	0.817	0%	10000
10	0.782	0%	10000
50	0.811	0%	10000
100	0.833	0%	10000
500	0.846	0%	10000
Average	0.817	0%	10000
Results using Meta-GA-calibrated parameters from 100 peak calibration, on SGA			
Peaks	Mean Best Fitness	Success Rate	Evaluations
1	0.940	0%	7000
5	0.948	0%	6670
10	0.932	0%	6605
50	0.950	0%	6689
100	0.941	0%	6722
500	0.930	0%	6731
Average	0.940	0%	6736

Table 38: SGA using parameters calibrated by REVAC on other problem instances.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Diversity of Population, Summed Population Variance.

Results using REVAC-calibrated parameters from 1 peak calibration, on SGA			
Peaks	Mean Best Fitness	Success Rate	Evaluations
1	0.969	10%	8775
5	0.953	0%	8769
10	0.961	0%	8569
50	0.946	0%	8508
100	0.942	10%	8877
500	0.937	0%	8835
Average	0.951	3 %	8722
Results using REVAC-calibrated parameters from 10 peak calibration, on SGA			
Peaks	Mean Best Fitness	Success Rate	Evaluations
1	0.938	0%	5990
5	0.942	0%	5923
10	0.922	0%	5997
50	0.907	0%	5795
100	0.936	0%	5760
500	0.938	0%	5772
Average	0.931	0%	5873
Results using REVAC-calibrated parameters from 100 peak calibration, SGA			
Peaks	Mean Best Fitness	Success Rate	Evaluations
1	0.950	0%	9765
5	0.929	0%	9351
10	0.930	0%	9468
50	0.947	0%	9785
100	0.944	10%	9533
500	0.940	0%	9726
Average	0.940	2% 9605	

## 7 Analysis & Discussion

In this chapter we will analyze and discuss the implications of the simulations done in the previous chapter. This information will be used in **Chapter 7: Conclusion** in order to answer our research questions.

We will now look at our data and observe whether or not our hypotheses are valid. As we have done our experiments to verify the second group of hypotheses first, we will analyze these three hypotheses first.

### 7.1 Comparison between meta-algorithms on ASGA

#### Hypothesis 2a:

**Comparing algorithms** - Parameters found using meta-evolutionary algorithms on autonomous selection will have a higher mean best fitness and success rate than when using either parameters found using manual calibration or best-practice parameters.

From the results in Table 39 a major problem is obvious, which has been remarked about previously: there is a major difference between the original ASGA results and those verified using the same best-known parameters. As of yet the reason of this discrepancy is unknown.

If one were to look only at the verified results, it is clear that both meta-evolutionary algorithms do a reasonable job, however the Meta-GA results proved to be consistently above the verified results. The REVAC results are worse than the verified results for for the 1, 10 and 100 peak problem instances, however the differences aren't very large.

#### 7.1.1 Diversity of Population

In the results in Table 39 the diversity of the population is also taken into account. In all cases the Meta-GA performs much better than the competition, meaning that the Meta-GA tends to find parameters with a higher diversity. A high degree of population diversity seems to be correlated with a high fitness.

#### 7.1.2 Statistical Significance

The primary means of comparison is that of the mean best fitness. Having done all runs 10 times and determining the mean best fitness and the variance of the results, it is possible to conclude which of these results are indeed statistically significant using a two-sided t-test and a probability of 1% that we are incorrect. For every comparison, the null-hypothesis is that the algorithms don't show a significant difference, the alternative hypothesis being that one of the two algorithms performs statistically better than the other.

From the results in Table 40 we can conclude that the results found are indeed better compared with the previously-known ASGA parameters. Our results support our hypothesis, but only for the Meta-GA. REVAC performs

Table 39: ASGA results: a comparison.  
 Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success, Population Diversity

Problem Instance: Spears 1 peak				
	Manual (original)	Manual (verified)	Meta-GA	REVAC
Mean Best Fitness	0.997	0.92	0.986	0.879
Success Rate	76%	0%	30%	0%
Evaluations	-	10000	4287	10000
Population Diversity	-	0.0073	1.066	0.0455
Problem Instance: Spears, 10 peaks				
	Manual (original)	Manual (verified)	Meta-GA	REVAC
Mean Best Fitness	0.998	0.919	0.964	0.896
Success Rate	84%	0%	0%	0%
Evaluations	-	10000	10000	10000
Population Diversity	-	0.0072	0.0472	0.0335
Problem Instance: Spears, 100 peaks				
	Manual (original)	Manual (verified)	Meta-GA	REVAC
Mean Best Fitness	0.982	0.908	0.960	0.867
Success Rate	28%	0%	10%	0%
Evaluations	-	10000	3697	10000
Population Diversity	-	0.0072	1.633	0.0307

worse in all cases compared to the verified or the Meta-GA results. The Meta-GA outperforms the verified results in all three cases, and beats the REVAC results by a large margin in all three cases.

Table 40: ASGA statistical significance: we use a 3-way analysis and we consider a probability of  $P_{m_0=m_1|t} < 1\%$  as significant proof that the hypothesis are not the same.

# Peaks	Manual	Mean Best Fitness	Variance	Meta-GA	Mean Best Fitness	Variance	$P_{m_0=m_1}$	Better Algorithm
1		0.92	0.00009		0.986	0.00025	$\ll 1\%$	Meta-GA
10		0.919	0.00029		0.964	0.00009	$\ll 1\%$	Meta-GA
100		0.908	0.00054		0.960	0.00129	$\ll 1\%$	Meta-GA
# Peaks	Manual	Mean Best Fitness	Variance	REVAC	Mean Best Fitness	Variance	$P_{m_0=m_1}$	Better Algorithm
1		0.92	0.00009		0.879	0.00050	$\ll 1\%$	Manual
10		0.919	0.00029		0.896	0.00014	$\ll 1\%$	Manual
100		0.908	0.00054		0.867	0.00005	$\ll 1\%$	Manual
# Peaks	Meta-GA	Mean Best Fitness	Variance	REVAC	Mean Best Fitness	Variance	$P_{m_0=m_1}$	Better Algorithm
1		0.986	0.00025		0.879	0.00050	$\ll 1\%$	Meta-GA
10		0.964	0.00009		0.896	0.00014	$\ll 1\%$	Meta-GA
100		0.960	0.00129		0.867	0.00005	$\ll 1\%$	Meta-GA

## 7.2 Simple GA analysis

**Hypothesis 2b:**

**Simple GA** - Parameters found using meta-evolutionary algorithms will not cause a significantly higher mean best fitness than when best-practice parameters are used with the simple genetic algorithm.

Table 41: SGA results: a comparison.

Performance values: Mean Best Fitness, Success Rate, Average number of Evaluations to Success

Problem Instance: Spears, 1 peak				
	Manual (original)	Manual (verified)	Meta-GA	REVAC
Mean Best Fitness	1.0	0.866	0.957	0.969
Success Rate	100%	0%	10%	10%
Evaluations	-	10000	5272	8775
Problem Instance: Spears, 10 peaks				
	Manual (original)	Manual (verified)	Meta-GA	REVAC
Mean Best Fitness	0.994	0.855	0.782	0.922
Success Rate	90%	0%	0%	0%
Evaluations	-	10000	10000	5997
Problem Instance: Spears, 100 peaks				
	Manual (original)	Manual (verified)	Meta-GA	REVAC
Mean Best Fitness	0.988	0.856	0.941	0.944
Success Rate	26%	0%	0%	10%
Evaluations	-	10000	6722	9533

As discussed previously, the reason that the original values are so high is because of the non-standard use of tournament selection. In our analysis we will therefore look primarily at the results using the Simple GA in the traditional sense: with fitness-proportional parent selection and generational survival selection.

Looking at the mean best fitness-results in Table 41, we see that REVAC performs remarkably well. In all cases it beats both the verified and the Meta-GA results, although the latter only with a minor amount with the 1 and 100-peak problem instances.

### 7.2.1 Statistical Significance

Again we must determine if the differences found are statistically significant. As with the ASGA results, we use a two-sided t-test and a probability of 1% to weigh the SGA results. The results of this analysis is shown in Table 42.

Table 42: SGA statistical significance: we use a 3-way analysis and we consider a probability of  $P_{m_0=m_1|t} < 1\%$  as significant proof that the hypothesis are not the same.

# Peaks	Manual		Variance	Meta-GA		Variance	$P_{m_0=m_1}$	Better Algorithm
	Best Fitness	Mean Best Fitness		Best Fitness	Mean Best Fitness			
1		0.866	0.00036		0.957	0.00067	$\ll 1\%$	Meta-GA
10		0.855	0.00107		0.782	0.00089	$\ll 1\%$	Manual
100		0.856	0.00094		0.941	0.00086	$\ll 1\%$	Meta-GA
# Peaks	Manual		Variance	REVAC		Variance	$P_{m_0=m_1}$	Better Algorithm
	Best Fitness	Mean Best Fitness		Best Fitness	Mean Best Fitness			
1		0.866	0.00036		0.969	0.00063	$\ll 1\%$	REVAC
10		0.855	0.00107		0.922	0.00077	$\ll 1\%$	REVAC
100		0.856	0.00094		0.944	0.00097	$\ll 1\%$	REVAC
# Peaks	Meta-GA		Variance	REVAC		Variance	$P_{m_0=m_1}$	Better Algorithm
	Best Fitness	Mean Best Fitness		Best Fitness	Mean Best Fitness			
1		0.957	0.00067		0.969	0.00063	$\ll 1\%$	REVAC
10		0.782	0.00089		0.922	0.00077	$\ll 1\%$	REVAC
100		0.941	0.00086		0.944	0.00097	$\ll 1\%$	REVAC

In spite of the small margins, REVAC maintains a statistically significant lead on the Meta-GA results. The Meta-GA performs reasonably well on the 1 and 100-peak, but it seems to have failed on the 10-peak problem instances.

Even so, it is clear that using meta-evolutionary algorithms to calibrate and algorithm provides a significant improvement compared to using best-practice parameters.

### 7.3 Ease of applying methods compared to trial and error

**Hypothesis 2c:**

**Applicability** - Using meta-evolutionary algorithms in finding parameters instead of exhaustive search or manual calibration will show a drastic reduction of time required. Meta-evolutionary algorithms should be considered in all cases where best-practice parameters are not directly applicable, as they will give better results.

Compared with the previous two hypotheses, this hypothesis does not have hard data to support it. However, we can look at the time required to find suitable ASGA parameters.

The original authors of the ASGA in [6] took about 12 weeks to find the best parameters known. From personal correspondence we know that although this team didn't work full-time on the problem, they did work on the problem with a team of 6. Assuming they worked on average one day a week on this particular issue, the total amount of researcher time is 72 days.

Now to look at the amount of time our team invested. It is possible to look at this in two ways: the amount of time required to run the calibrations or the amount of time to develop the calibration framework and run the calibrations. For the former, the calibration runs we performed rarely took more than two days. From this point of view the use of meta-evolutionary algorithms saved us 70 days of manual calibration. However, the development of the framework took a considerable amount of time (primarily because we intended it to be as flexible as possible): about 16 weeks with on average three days a week. This includes experimenting and optimizing the framework where necessary. The total amount of time spent then becomes 50 days: still a saving of 22 days of work.

Considering the results of the analysis of the previous two hypotheses and the amount of time invested, we can conclude that applying meta-evolutionary algorithms for parameter calibration is certainly worthwhile compared to manual calibration. Even if best-practice parameters are available, it can be rewarding in terms of better performance. If calibration is required, we can only conclude that using meta-evolutionary algorithms should be seriously considered.

### 7.4 Quality of the autonomous selection solutions discovered

**Hypothesis 1a:**

**Calibration** - It is possible to find better parameters for the autonomous selection genetic algorithm when comparing the mean best fitness and success rate.

Considering the data in Table 39 and Table 40 above we can conclude that it is indeed possible to find better parameters for the autonomous selection genetic algorithm. Although the new results don't outperform the original results in [6], they do outperform the verified results.

## 7.5 Stability of results

### Hypothesis 1b:

**Sensitivity and Stability** - Parameters of the autonomous selection genetic algorithm are sensitive, in that small changes to either the parameters of autonomous selection or the underlying fitness landscape of the problem instance can lead to an ineffective algorithm.

From the experiments it is clear that minor changes to the ASGA parameters cause the performance of the algorithm to drop significantly. Especially the value for the “survival selection - multiply” parameter needs to be within a very small range: outside of the -0.004 to -0.008 region the performance of the algorithm drops sharply.

On the other hand, the parameters are stable for other fitness landscapes than those which were optimized on. Parameters work comparably on fitness landscapes with a much higher or lower number of peaks than that the algorithm was calibrated on.

This hypothesis is thus partly true: small changes to the parameters of autonomous selection cause a sharp drop in performance, while a different fitness landscape is much less a problem for the algorithm.

However we must not forget to also look at the population dynamics of the verified and calibrated ASGA runs. We can see from the population plots in the previous chapter that the population size remains relatively constant for the duration of most runs, the population variance remains low after the first few generations of the algorithm.

## 7.6 Comparable performance?

### Hypothesis 1c:

**Improvement** - A genetic algorithm using autonomous selection, with well-tuned parameters, can give results equal to or better than a well-tuned simple genetic algorithm when looking at the mean best fitness and success rate.

For the above hypothesis we need to combine the best results from both the ASGA and SGA calibrations. With the ASGA, the Meta-GA performed the best overall, while with the SGA REVAC proved to obtain the highest fitness.

From the results in Table 43 we see that ASGA has a substantial lead on all SGA results. The calibrated ASGA results outperform the calibrated SGA results, therefore we can conclude that an algorithm using autonomous selection can indeed provide results better than than a simple genetic algorithm.

Table 43: ASGA versus SGA.

Performance values: Mean Best Fitness, Success Rate

Problem Instance: Spears, 1 peak				
Result	ASGA, manual	ASGA, Meta-GA	SGA, manual	SGA, REVAC
Mean Best Fitness	0.92	0.986	0.866	0.969
Success Rate	0%	30%	0%	10%
Problem Instance: Spears, 10 peaks				
Result	ASGA, manual	ASGA, Meta-GA	SGA, manual	SGA, REVAC
Mean Best Fitness	0.919	0.964	0.855	0.922
Success Rate	0%	0%	0%	0%
Problem Instance: Spears, 100 peaks				
Result	ASGA, manual	ASGA, Meta-GA	SGA, manual	SGA, REVAC
Mean Best Fitness	0.908	0.960	0.856	0.944
Success Rate	0%	10%	0%	10%

## 8 Conclusion

Our conclusions:

- REVAC finds better parameters when calibrating Simple GA than Meta-GA
- Meta-GA finds better parameters when calibrating ASGA than REVAC
- Results are statistically significant
- Calibrated ASGA results are better than results from previously known parameters
- Population size remains constant in ASGA when using properly calibrated parameters
- Usage of meta-evolutionary algorithms is efficient

REVAC works very well on a smooth-search landscape, as is clear from the calibration of the Simple GA. It maintains less diversity in its population, which is a problem for EDAs in general. The reason why REVAC doesn't work well on the ASGA is that the gradient is only significant in a very small region, and this region is too small for the ASGA.

A Meta-GA works better on a landscape where must-have-but-hard-to-find features exist. It maintains a much higher degree of population diversity, allowing it to find many niches and converge on these niches quickly. The ASGA has such a landscape, the Meta-GA finds the gradient mentioned previously and thus calibrates the ASGA properly. On the down-side, Meta-GA in its current form works less well for local optimization and thus the SGA results aren't as good as those found with REVAC.

As we have calculated in the previous chapter, we have determined that the results found are statistically significant.

Those results show that a calibrated ASGA algorithm performs better than an ASGA algorithm that uses the previously best-known parameters.

What is clear from the population plots is that the calibrated ASGA algorithm also maintains a more stable population size. The parameters found during calibration prevent the algorithm from the population going extinct or exploding, which leads to more robust results over multiple problem instances, even those which the algorithm hasn't been calibrated on.

What we can also conclude is that the usage of meta genetic algorithms for calibrating genetic algorithms can be a very useful approach for complex problem instances. Setting up and running the meta algorithms took little over a day and found parameters that were thus-far unknown, yet performed better for the autonomous selection genetic algorithm than the parameters found by months of manual trial & error.

## 8.1 Future Work

Although one might think that the choice for a meta-algorithm is essentially the same problem as the choice for good parameters, this is not the case. As is clear from the analysis chapter, both meta-algorithms find good results within a certain amount of evaluations. However, we must also conclude that certain meta-algorithms are more effective for calibration than others. It is reasonable to presume that better results would be found if we had allowed the meta-algorithms to continue for a longer amount of time.

For our experiments we have only looked at instances from Spears' multimodal problem generator. Although we have assumed that we can limit ourselves to this type of problem instance, it would be interesting to see if differences in the type of problem instance results in different results at the calibration level.

We could also look at other real-world problems to calibrate. As there are many algorithms used on a daily basis in many fields, it would be interesting to calibrate other algorithms using the meta-algorithms in this thesis. Further comparisons with manually calibrated parameters would support our conclusion that meta-algorithms are more effective for parameter calibration.

On autonomous selection, we do believe that there is room for improvement. Experiments done on a distributed system could provide more validation for the use of autonomous selection. It would also be useful to compare autonomous selection on a distributed system with tournament selection or even a simple genetic algorithm on a distributed system. By extrapolating the results obtained, it would then be possible to determine how large a distributed system would have to be in order to validate the use of autonomous selection above other types of selection. On a distributed system communication simply can't be avoided, there always has to be a balance between the communication and the overall performance of such a system.

Looking at the results of both meta-evolutionary algorithms, we see that both are effective, even with a limited amount of evaluations. REVAC however is fairly new, and more research could be done in order to make it more effective.

## A GAMP

In this appendix we will describe the GAMP library constructed in order to do our experiments.

### A.1 GAMP Overview

The Genetic Algorithm Module for Python, or GAMP, is a library that easily allows deployment of a (meta) genetic algorithm. We chose to develop a new library instead of using an existing one for two reasons. First, because existing modules don't offer the features required for our experiments or would have required considerable rewriting. Second, because developing such a module from scratch allow us to keep to the basics required for our research, instead of having to deal with large complicated APIs which might do unexpected things under the hood.

In keeping with the scientific tradition of transparency, all code used in conducting the experiments is available from the GAMP website (<http://www.alextrreme.org/projects/gamp>). Documentation is also available for others wanting to use GAMP. The GAMP framework was written with a large degree of flexibility in mind, and where the framework isn't flexible enough the licensing (GNU Lesser General Public License, LGPL) allows others to modify GAMP to suit their needs.

GAMP was written in Python ([www.python.org](http://www.python.org)), a general-purpose object-oriented programming language also used by Google, NASA and the NYSE. The reason for choosing Python was due to the clear syntax, powerful semantics, extensive libraries and high degree of maintainability.

### A.2 Features

#### A.2.1 Representation

GAMP currently has direct support for discrete alleles, and has mappings from bit-strings to integers and floating-point numbers. An arbitrary number of alleles may be used.

#### A.2.2 Recombination

GAMP supports one-point uniform crossover with a user-defined degree of crossover probability. No research on other forms of recombination were conducted, so no other types of recombination have been implemented.

#### A.2.3 Mutation

GAMP supports single, allele-flipping mutation with a user-defined degree of mutation probability. Mutation can both be done together with recombination in providing the offspring for selection or be applied after survival-selection on the next generation.

#### A.2.4 Parent selection

As selection has been one of the main topics, GAMP supports a range of parent selection methods, all described extensively in the algorithm sections of this thesis:

- Fitness-proportional selection
- Linear rank-based selection
- Tournament selection
- Best selection
- Random selection

GAMP has support for an arbitrary tournament size. The above methods are used for determining which entity in the population is allowed to reproduce.

#### A.2.5 Survival selection

As with parent selection, GAMP also sports a number of different types of survival selection methods:

- Fitness-proportional selection
- Generational selection
- Generational/Fitness-proportional selection
- Tournament selection
- Best selection
- Random selection
- Elitist selection (combined with any of the above)

#### A.2.6 And more...

GAMP also has a number of non-GA-specific optional features, like:

- Evaluation caching, to avoid duplicate evaluations
- Machine-readable logging of overall statistics, entity data
- Automatic backups after each generation
- Various user-specified halt-conditions
- Easy integration of existing applications

### A.3 Conclusion

Although GAMP has been designed primarily for our experiments in (meta) genetic algorithms, we think it has enough merit to be used by others both inside and outside the research community. Even so, GAMP is still in active development; feedback to the authors is very welcome.

---

## References

- [1] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation, The New Experimentalism*. Springer Berlin Heidelberg, 2006.
- [2] T. C. Belding. The distributed genetic algorithm revisited. In D. Eschelmann, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1995.
- [3] G. Box and K. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society, Series B*, 13(1):1–45, 1951.
- [4] J. Clune, S. Goings, B. Punch, and E. Goodman. Investigations in metagases: panaceas or pipe dreams? In F. Rothlauf, editor, *GECCO 2005: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 235–241, New York, NY, USA, 2005. ACM Press.
- [5] P. Cortez, M. Rocha, and J. Neves. A meta-genetic algorithm for time series forecasting. In L. Torgo, editor, *Proceedings of Workshop on Artificial Intelligence Techniques for Financial Time Series Analysis (AIFTS-01), 10th Portuguese Conference on Artificial Intelligence (EPIA'01)*, pages 21–31, Oporto, Portugal, Dec. 2001.
- [6] A. Eiben et al. The possibilities of autonomous selection. Project report, 2006.
- [7] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.
- [8] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2003.
- [9] D. B. Fogel. *Blondie24*. Academic Press, 2002.
- [10] B. Freisleben. *Advanced Techniques in Evolutionary Computation; Metaevolutionary approaches*, chapter 7.2, pages 212–223. Oxford University Press, 1997.
- [11] Friedman, Milton, and Savage. *Techniques of Statistical Analysis*, chapter 13, Planning Experiments Seeking Maxima. McGraw-Hill, 1947.
- [12] J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. Syst. Man Cybern.*, 16(1):122–128, 1986.
- [13] J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. V. Gucht. Genetic algorithms for the traveling salesman problem. In J. J. Grefenstette, editor, *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 160–168, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.

- 
- [14] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, 1975.
- [15] H. Hotelling. Experimental determination of the maximum of a function. *Annals of Mathematical Statistics*, 12(1):20–45, 1941.
- [16] K. A. D. Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University Michigan, 1975.
- [17] K. A. D. Jong, M. A. Potter, and W. M. Spears. Using problem generators to explore the effects of epistasis. In T. Bäck, editor, *Proceedings of The Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.
- [18] H.-S. Kim and S.-B. Cho. Application of interactive genetic algorithm to fashion design. *Engineering Applications of Artificial Intelligence*, 13(6):635–644, 2000.
- [19] M. Lee and H. Takagi. Integrating design stage of fuzzy systems using genetic algorithms. In IEEE, editor, *Second IEEE International Conference on Fuzzy Systems*, volume 1, pages 612–617. IEEE Computer Society Press, 1993.
- [20] R. Levin and D. Rubin. *Applied Elementary Statistics*. Prentice-Hall, 1980.
- [21] H. H. Lund and O. Miglino. Evolving and breeding robots. In P. Husbands and J.-A. Meyer, editors, *Evolutionary Robotics, First European Workshop*, pages 192–210. Springer, 1998.
- [22] M. Mataric and D. Cliff. Challenges in evolving controllers for physical robots. *”Evolutional Robotics”, special issue of Robotics and Autonomous Systems*, 19(1):67–83, 1995.
- [23] V. Nannen and A. E. Eiben. A Method for Parameter Calibration and Relevance Estimation in Evolutionary Algorithms. In M. Keijzer et al., editors, *GECCO 2006: Genetic and Evolutionary Computation Conference*, pages 183–190, New York, 2006. ACM.
- [24] V. Nannen and A. E. Eiben. Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Jan. 2007. To appear.
- [25] Y. M. Naoki. Agent oriented self adaptive genetic algorithm. In M. H. Hamza, editor, *Communications and Computer Networks*, 2002.
- [26] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 224–230, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

- 
- [27] M. E. Samples, J. M. Daida, M. Byom, and M. Pizzimenti. Parameter sweeps for exploring GP parameters. In H.-G. Beyer et al., editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1791–1792, Washington DC, USA, 25–29 June 2005. ACM Press.
- [28] A. V. Samsonovich and K. A. D. Jong. Pricing the ‘free lunch’ of meta-evolution. In H.-G. Beyer and U.-M. O’Reilly, editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1355–1362, New York, NY, USA, 2005. ACM Press.
- [29] K. Shahookar and P. Mazumder. VLSI cell placement techniques. *ACM Comput. Surv.*, 23(2):143–220, 1991.
- [30] J. L. Shapiro. Diversity loss in general estimation of distribution algorithms. In T. P. Runarsson, H.-G. Beyer, E. K. Burke, J. J. M. Guervós, L. D. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, number 4193 in LNCS, pages 92–101. Springer, 2006.
- [31] W. M. Spears. *Evolutionary Algorithms; The role of mutation and recombination*. Springer Berlin Heidelberg, 2000.
- [32] S. A. Stanhope and J. M. Daida. Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment. In V. Porto, N. Saravanan, D. Waagen, and A. Eiben, editors, *Evolutionary Programming VII*, number 1447 in LNCS, pages 693–702. Springer, 1998.
- [33] E. Takashima, Y. Murata, N. Shibata, and M. Ito. Self adaptive island ga. In IEEE, editor, *Congress on Evolutionary Computation*, pages 1072–1079. IEEE Computer Society Press, 2003.
- [34] R. Tanese. Parallel genetic algorithms for a hypercube. In J. J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 177–183. Lawrence Erlbaum Associates, Publishers, 1987.
- [35] J. van Hemert and A. Eiben. Mondriaan art by evolution. In E. Postma and M. Gyssens, editors, *Proceedings of the Eleventh Belgium/Netherlands Conference on Artificial Intelligence*, pages 291–292. CWI, Amsterdam, 1999.
- [36] G. Wang, T. Dexter, E. Goodman, and W. Punch. Optimization Of a GA and Within the GA for a 2-Dimensional Layout Problem. In E. Goodman, editor, *First International Conference on Evolutionary Computation and Its Applications*. Presidium of the Russian Academy of Sciences, 1996.
- [37] G. Wang, E. Goodman, and W. Punch. On the optimization of a class of blackbox optimization algorithms. Technical report, GARAGE/Case Center, Michigan State University, 1997.

- [38] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: the genetic edge recombination. In J. D. Schaffer, editor, *Proceedings of the third international conference on Genetic algorithms*, pages 133–140, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.
- [39] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.
- [40] A. Wright, R. Poli, C. Stephens, W. Langdom, and S. Pulavarty. An estimation of distribution algorithm based on maximum entropy. In K. Deb et al., editors, *GECCO 2004: Genetic and Evolutionary Computation Conference*, pages 343–354. Springer, 2004.