Parameter Tuning a Real-Valued Evolutionary Algorithm: An Autonomous Driving Case Study

Oliver Funk
Department of Computer Science
University of Cape Town
South Africa

November 2015

Abstract

Evolutionary Algorithms (EAs) have many parameters that may be altered and the optimization of those parameters can lead to a substantial performance gain and improvement in overall solution quality, which merits the effort required in doing so. However, in practice, this is rarely done as the effects of the parameters is not well understood. In this paper we present an empirical study on the effect various parameters have on the performance of a real-valued EA. Three main components of the EA were considered: parent selection, genetic crossover and genetic mutation, with each having three different operators that were tested. It was found that the selection and mutation operators had a significant impact on the performance of the EA, a mutation rate that was too high or too low resulted in sub-optimal convergence and the effect of selection pressure could not be fully quantified, however, a moderate selection pressure worked best.

Keywords: Evolutionary Algorithm, real-valued, parameter tuning, selection, crossover, mutation

1 Introduction

Evolutionary Algorithms (EAs) are a class of general purpose search algorithms modelled on the mechanics of evolution (natural selection, survival of the fittest and genetic mutations [7]). They have proven to be effective in finding 'good' solutions to optimization problems that cannot be solved easily using other methods [16]. Due to the effectiveness of EAs, they have been applied to optimization problems from a wide variety of fields (Big Data, Engineering Design, Robotics, Finance)

[19], thus their performance[1] is becoming a critical aspect of their use. It is now not merely about finding good solutions, but to find them quickly and reliably.

The encoding scheme used to represent the problem (for example, binary [5], permutation [28], real valued [31]) dictates the kinds operations that are performed to search for solutions to the problem and thus the operators available for the algorithm to use. The parameters used by an EA will have an effect of its evolutionary search and thus its performance, which is coupled to the algorithms ability to search a region of possibly good solutions [2]. Good parameter values are essential for good performance, in practice however, little time and effort is spent tuning them and the values used are generally based on conventions chosen on an ad-hoc basis [12].

Eiben and Smit [12] state that finding appropriate parameter values for evolutionary algorithms (EA) is one of the persisting grand challenges of the evolutionary computing (EC) field. This reluctance may be attributed to the difficulty of finding good parameter values for an EA, given the large number of values that may be changed and the limited knowledge about the effect these values have on the performance of the EA [12]. Therefore, gaining ground in the understanding of these effects is important.

In this paper we conducted an empirical study on the effect different parameter values had on the performance of a real-valued EA. The EA was used to optimize the placement of sensors on a simulated autonomous car, using a real-valued encoding for their positions. Two rounds of experiments were conducted. In the first, nine different operators were tested to determine which yielded the best performing EA and in the second, the values of the parameters used by the best performing operators were modified to determine the effect those values have. Both rounds were conducted in order to test three hypotheses:

1. The optimal mutation rate is $1/l$ (l being the the number of genes in the genotype), as suggested by [6].
2. High selection pressure will result in early convergence on non-optimal solutions, which is based on the theoretical understanding of selection pressure.
3. The selection operator has the biggest impact on the performance of the EA, as suggested by Nannen et. al. [26].

The main purpose of this paper is to find the set of parameters values that yield an EA with the best search performance and by doing so, attempt to understand the effects those parameters value have and to test the hypotheses stated above.

The paper is organized as follows. In section 2, we present the workings and theory behind EAs and discuss the terminology used to unambiguously describe EA parameters. In section 3, the methodology is described detailing the tested operators, the EA used and the experiments conducted. In section 4, the results of

[1]Performance (or search performance in this is case) is a measure of the ability of an EA to find good solutions to a given problem in a set amount of time.

the experiments are given and a discussion is presented about the trends observed in the results. Finally, we conclude the paper in section 5, providing a summary of what was found and stating possible future work.

2 Literature Review

2.1 Theoretical Background

The Evolutionary Algorithm is a branch of optimization algorithms [24]. These algorithms are stochastic and non-deterministic, one cannot know if a solution found is the 'best' solution to the problem, only that it is, at most, a very good solution. EAs are thus approximation algorithms, which are best applied to problems where analytical methods (such as constructive heuristics) cannot be applied or to problems with the extremely large search spaces, using other methods would be too slow (such as the Travelling Salesman Problem).

The EA is modelled on Darwin's Theory of Evolution [7]. He saw the great diversity found in nature and the remarkable adaption species have to their environments and offered an explanation for this phenomenon, known as survival of the fittest through natural selection. Darwin noted that desirable phenotypic traits were propagated into subsequent generations via offspring. These desirable qualities made individuals fitter, that is, better suited to their environment, allowing them to survive and reproduce. Phenotypic traits are encoded in an individuals genes and when two individuals reproduce, their genes are mixed in a certain way to produce offspring and as such the offspring will share some common phenotypic traits with both parents. However, sometimes during the reproduction process small, random mutations may occur, giving offspring new random traits. This is the mechanism by which genetic diversity is introduced into a population and may lead to new beneficial adaptations thereby making individuals better suited to their environment. This incremental evolutionary process is replicated by EAs to find good solutions (that is, solutions with high fitness) to any problem that can be encoded in a way an EA can use, by searching through the problem's search space.

In the nineteen seventies, Holland [21] begun formalizing the Evolutionary Algorithm which used ideas about fitness and suitably to an environment, mating and mutation to create robust, adaptive systems (algorithms), cable of dealing with an uncertain and changing environment [8]. Thereafter, Goldberg [14], [15], De Jong [22] and others ([23], [4], [18]) then began applying these principles to more complex problems, such classifier systems, machine learning, evolving programs, artificial life) [8].

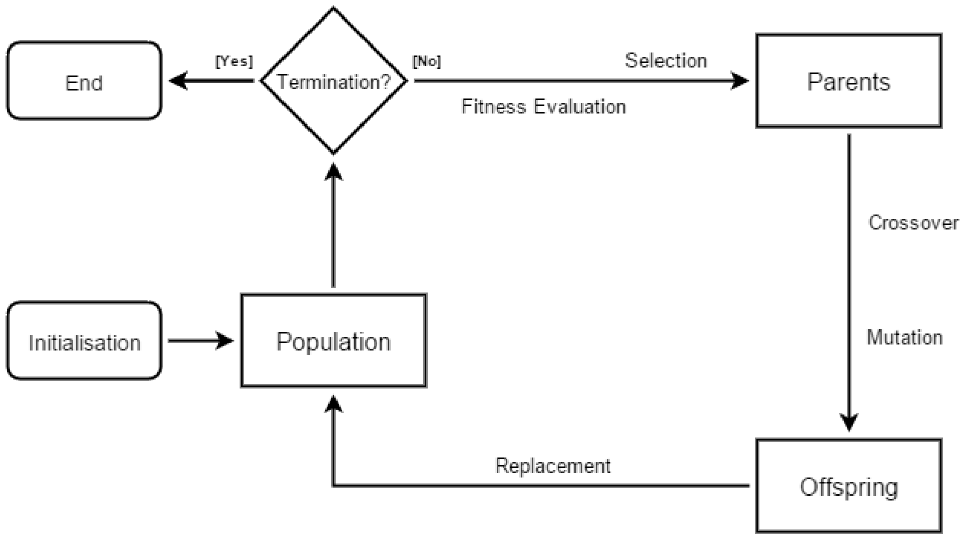

Figure 1: A common scheme of an EA showing the genetic operations performed as a flow-chart

2.2 Evolutionary Algorithms and Their Parameters

The basic workings of an EA are as a follows (see figure 1): a population of candidate solutions (individuals) evolve through the use of genetic operators, which are modeled on the mechanics of evolution and the way species evolve to suit their environment. The fitness, or quality, of each solution is evaluated by placing individuals in the environment and allowing enough time to pass so their fitness may be determined (this may be as simple evaluating a mathematical function with a candidate solution). The individuals that are better suited to their environment will have a higher fitness value and thus their genes will more likely be passed onto the next generation. As time passes, the population as a whole begins to adapt to the fitness landscape of the problem [13].

A candidate solution (or individual) is, at the most fundamental level, encoded by its genotype. The genotype is used to derive the phenotype of the individual and the phenotype is the real world representation of the individual. As Eiben and Smith [13] state “The fundamental observation from genetics is that each individual is a dual entity: its phenotypic properties (outside) are represented at a low genotypic level (inside)”. Genes are the units of inheritance and encode the phenotypic traits for an individual. A genotype is the collection of genes and a phenotype is the collection of phenotypic traits an individual has. The mapping between the genotype and phenotype may not always be one-to-one, as one gene may affect various phenotypic traits.

The genetic operators of EAs are: selection, crossover and mutation, as well as schemes that affect the way the evolution works (such as the Elitism replacement scheme, where a number of the best individuals are put into subsequent generations without change).

The selection operator chooses individuals from a population that will be combined to produce offspring. Nannen et. at. [26] suggest the choice of selection operator has the strongest impact on the performance of the EA. A selection operator that heavily basis fitter individuals is said to have high selection pressure. Selection pressure is a measure of the probability with which individuals of higher fitness will be chosen for further genetic processing, compared to those with lower fitness [3]. If the selection pressure is too high, the average fitness of a population tends to grow quickly, but generally results in sub-optimal convergence and genetic stagnation as a result [17].

The mutation operator randomly changes (mutates) offspring in some way, which can allow for an escape from a local optima. Mutations may counter genetic stagnation by introducing new genetic material into the population, however, it may also cause the loss of some genes that would have produced good results. Hesser and Mnner [20] suggest that mutation is useful as it can compensate for the "loss of building blocks due to errors of the selection process". There have been multiple studies on the optimal rate of mutation [20], [6], [27] which mostly coincide on the suggestion that optimal mutation rate depend mainly on $1/l$ (the reciprocal of the genotype length). However, Deb and Agarawal [10] suggest that the rates are largely dependent on the function being optimized and one cannot stipulate a best value.

The crossover operator (also called the recombination operator) uses selected individuals and combines their genes to produce offspring. Crossover operators will exhibit a high fitness correlation between parents and their offspring, meaning parents with low fitness will produce low fitness offspring, a crossover operator cannot be expect to produce high fitness offspring from low fitness parents, however, it should produce high fitness offspring from high fitness parents [9]. Due to the structured manner in which crossover occurs (individuals must be selected by the selection operator first), it acts as the primary driving force behind the evolution and creation of new candidate solutions. Ortiz-Boyer [29] and Picek et. al. [30] argue that the crossover (recombination) operator is the most important operator as there are "significant differences between crossover operators" [30], which implies that a crossover operator will have the biggest impact on the performance of the EA.

2.3 Parameter Tuning

A formalization of the terminology used to describe the parameters of an EA is necessary in order to be clear about what distinguishes different EAs and for building a conceptual framework for how parameter tuning will be done. There are two general types of parameters, which are distinguished by their domains. A qualitative parameter has a finite domain with no sensible distance metric or ordering and a quantitative parameter is a subset of R, which may be bound [12]. This difference between the two types of parameters changes the way one searches for the best value for them. For the quantitative parameter, a heuristic search and optimization methods can be deployed to find better values, however, this is not possible for the qualitative parameter as it has no explorable domain and two values are not linked in anyway, thus each value must be sampled. A value for the qualitative parameter is a specific operator instance and each operator instance may require a number of its own quantitative parameters. Two EAs are said to be different when their qualitative parameters differ, implying that an EA is a partially specified algorithm. When all the parameters have been specified, we obtain a particular instance of an EA [12]. For example, the qualitative parameter 'crossover operator' may be set to the crossover operator instance 'one-point crossover', with its quantitative parameter 'crossover rate' set to '0.5', meaning the one-point crossover operator will be used and crossover will occur 50% of the time. For simplicity, from here onwards, a qualitative parameter will be referred to as an 'operator', the value of which be referred to as the 'instance' or 'operator instance' and the quantitative parameters used by the operator instance (if any) will be referred to as the 'instance parameters'.

There are two approaches one can use to find good parameters for an EA: parameter control and parameter tuning [11]. In the parameter control approach, the operator instances are set (that is, to specify what EA will be used) and the values of the instance parameters change during the run[2] of the EA, using some control strategy to change them, which may be deterministic, adaptive or self-adaptive. In the parameter tuning approach, the qualitative and quantitative parameters are specified at the start of the EA and do not change during its execution, making changes to the parameters only when the run has completed. In this paper, the parameter tuning approach is taken.

The act of tuning the parameters of an EA entails finding the set of parameters that deliver the best task performance, meaning. In general, task performance is a measure of solution quality (in terms of fitness) versus computational effort (how long it took to find). However, the definition of performance is problem specific. In some cases, finding a very good solution is far more important than the computational effort required to do so and in cases such as those, an EA needs to be 'reliable' and not necessarily 'quick'. In this paper, we seek to find a comprise between reliability and speed, meaning, we wanted to find a set of parameters that yields an EA instance which, on average, finds individuals with higher fitness than other EA instances, in a set amount of time.

[2]A run is defined as allowing the EA to execute until some stopping condition has been met (for example a maximum number of generation has been reach, or a maximum/required fitness has been reached)

3 Methodology

An EA was built and used to optimize the placement of the sensors on a vehicle, in order to find the most general sensor configuration, which is the configuration that detects the most number of unique obstacles. A sensor was a simple ray-casting device. A sensor configuration was defined to be a collection of 10 sensors. The genotype was defined to be an array of 10 two dimensional vectors, each containing the spherical co-ordinates for a sensor (the first component being the rotation about the Y-axis and the second being the rotation about the X-axis). The phenotype was defined to be the array that stored the configuration of the sensors in the game's world space. An individual was defined to be a vector that held the phenotype (the configuration) and the fitness accumulated by that phenotype. Figure 2 shows the genotype to phenotype mapping for a sensor and gives an example of a sensor configuration, that is, the complete phenotypic expression for the specified genotype.

3.1 Simulation

A car simulated in the Unity[3] game engine was used as the vehicle onto which the sensor configuration would be optimized. The car was modeled after a BMW M3 GTR and was scripted to have lifelike vehicle physics (see figure 3).

The track used was based on the parametric function (see figure 3):

$$f(u) = (x(u), y(u), z(u)) = (u, sin(u), cos(2u))$$

The track was built using Blender[4] with the Extra Objects add-on that comes pre-installed.

As the simulation began, obstacles were randomly placed on the track, each having a small random height, and the population was initialised, by generating random genotypes for each individual. The car was made to drive from a point A to a point B, using a heuristic driving AI [give credit]. While driving between the points, each sensor was checked to be detecting an obstacle, at a constant time step, and if so, the fitness of the individual whose phenotype held the sensor was increased. Importantly, if the sensor was still detecting the same obstacle the following time step, no extra fitness was allocated to it. Once the car had reached point B, a new generation was created using the parameters defined for the EA by the experiment and the process repeated until the maximum number of generations had passed.

[3]https://unity3d.com/
[4]https://www.blender.org/

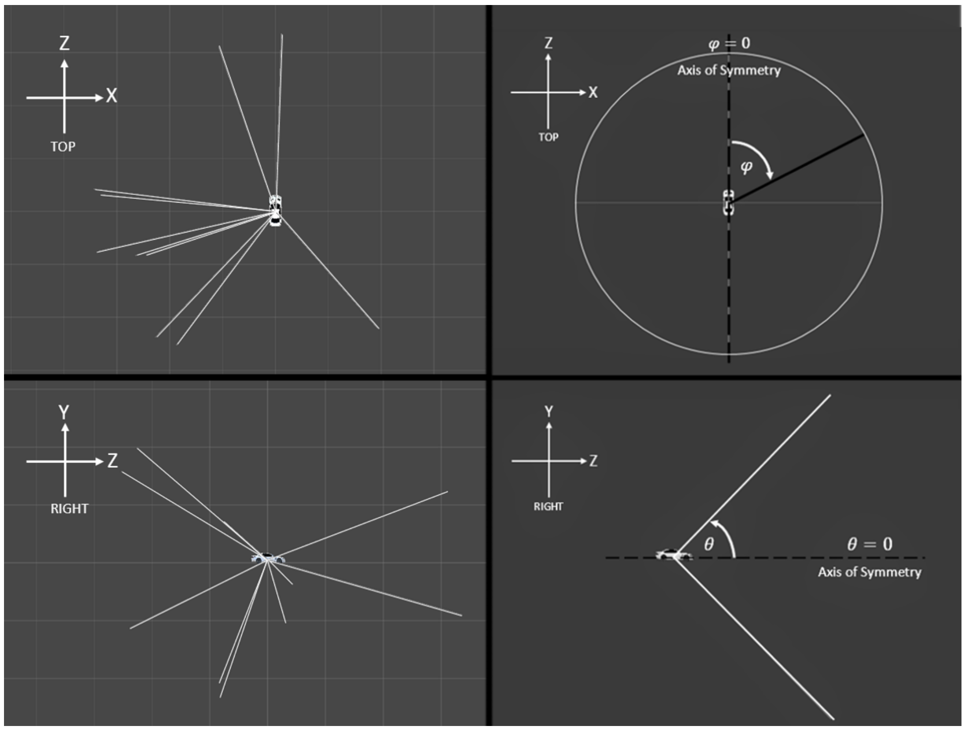

Figure 2: Left: *A Sensor Configuration*: showing a sensor configuration derived from the following genotype: $\{(6.9, 277)$, $(15.8, 1.9)$, $(334.1, 216.6)$, $(327.3, 138.7)$, $(42.4, 251.5)$, $(340.3, 340.9)$, $(19.8, 223.7)$, $(349, 257.3)$, $(37.3, 252.6)$, $(18.3, 275.4)\}$ — Right: *The Genotype To Phenotype Mapping*: showing how the values in a gene (θ, φ), where $\theta \in [0, 45] \cup [270, 360]$ was the angle of rotation about the X-axis (bottom) and $\varphi \in [0, 360]$ was the angle of rotation about the Y-axis (top), were used to place (map) a sensor onto the car

3.2 Operators Tested

The operators that were tested are detailed below, along with their sudo-code implementation. These operators were used as they are very commonly used throughout literature. Works such [26], [25], [13], [29], [30] show the use of these operators.

3.2.1 Selection Operators

Random Selection An individual is randomly selected from the population without taking its fitness into account. This is used as as control operator to show the effects of selection with and without a fitness bias.
See Algorithm 1

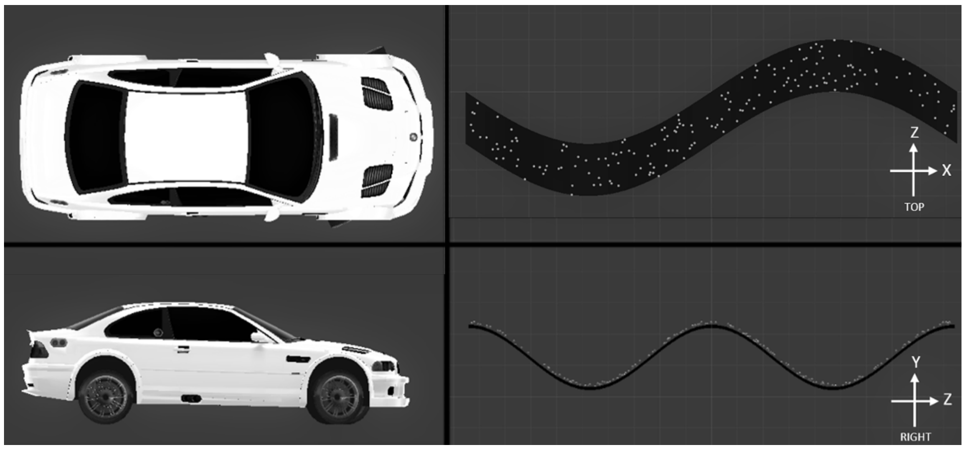

Figure 3: Left: *The Car* showing the car used in the simulations — Right: *The Track*: showing the track used with the obstacles randomly placed on it.

Fitness Proportionate Selection An individual is selected from the population with a probability of its fitness divided by the total fitness of the population, thus a higher fitness leads to a proportionately greater chance of being selected over others. See Algorithm 2

Tournament Selection An individual with the highest fittest is selected from a set (or tournament) of k individuals chosen randomly from the population. With this selection operator, the selection pressure (a measure of how biased the selection is to higher fitness individuals) can be manipulated directly by changing the tournament size, the larger the tournament size, the higher the selection pressure. Unlike with the Fitness Proportionate Selection operator, where the selection pressure is high and cannot be changed.
See Algorithm 3

3.2.2 Crossover Operators

One-point Crossover A descend is made by selecting a random point k in the genotype of the first parent ($P_a^{f_a}$) and copying the genes from the firs parent up to the point k, then the genes from the second parent ($P_b^{f_b}$) are copied from k until the end of the genotype.
See Algorithm 4

Uniform Crossover A descendant is made by randomly selecting (with a specified probability p) a gene from either of the two parents ($P_a^{f_a}$ and $P_b^{f_b}$). The gene selection probability p is the probability of selecting a gene from $P_a^{f_a}$, which is

defined to be the parent with higher fitness.
See Algorithm 5

Local Crossover Each gene of a descendant is determined by taking a weighted sum of the two corresponding genes of the parents $P_a^{f_a}$ and $P_b^{f_b}$. A randomly chosen value $\alpha \in [0, 1]$ is used as the weighting factor for each gene.
See Algorithm 6

3.2.3 Mutation Operators

Random Mutation An individual's gene is replaced by a new gene with a random value that is not depend on the value of the mutated gene.
See Algorithm 7

Gaussian Mutation An individual's gene is mutated by adding a random value chosen from Gaussian distribution to it. The Gaussian distribution has a mean of 0 (it is centered at 0) and a standard deviation of the range of allowed values divided by a precision number. The greater the precision, the smaller the standard deviation will be and thus the smaller, on average, the change will be.
See Algorithm 8

Breeder Mutation An individual's gene is mutated by adding or subtracting a random, but controlled amount (called the step size) from it. The step size is based on two precision values, the first defining the maximum step size and the second defining the minimum step size. The algorithm is based on the work by Mhlenbein [25].
See Algorithm 9

Algorithm 1: Random Selection

```
input  : The population $P = \{P_1^{f_1}, P_2^{f_2}, P_3^{f_3}, ...\}$
output : An individual from the population $P_s^{f_s}$

$N$: The total number of individuals in the population
$P_i^{f_i}$: An individual $i$ in the population with fitness $f_i$
rand_int($a, b$): Function that draws a random integer uniformly from the range $[a, b)$

begin
    $r \leftarrow$ rand_int($0, N$)
    $P_s^{f_s} \leftarrow P_r^{f_r}$
end
```

Algorithm 2: Fitness Proportionate Selection

```
input  : The population $P = \{P_1^{f_1}, P_2^{f_2}, P_3^{f_3}, ...\}$
output : An individual from the population $P_s^{f_s}$

$P_i^{f_i}$: An individual $i$ in the population with fitness $f_i$
$N$: The total number of individuals in the population
sum: The total fitness of the population
rand_float($a, b$): Function that draws a random number uniformly from the range $[a, b]$

begin
    $r \leftarrow$ rand_float($0$, sum)
    for $i \in [0, N]$ do
        $r \leftarrow r - f_i$
        if $r \leq 0$ then
            $P_s^{f_s} \leftarrow P_i^{f_i}$
            break out of the for-loop
        end
    end
end
```

Algorithm 3: Tournament Selection

```
input  : The population $P = \{P_1^{f_1}, P_2^{f_2}, P_3^{f_3}, ...\}$, an integer $k$ being the
         tournament size
output : An individual from the population $P_s^{f_s}$

$P_i^{f_i}$: An individual $i$ in the population with fitness $f_i$
$N$: The total number of individuals in the population

rand_int($a, b$): Function that draws a random integer uniformly from the
range $[a, b)$

begin
    $f_s = 0$
    for $k$ *number of times* do
        $r \leftarrow$ rand_int($0, N$)
        if $f_r > f_s$ then
            $P_s^{f_s} \leftarrow P_r^{f_r}$
        end
    end
end
```

Algorithm 4: One-point Crossover

```
input  : Two selected parents $P_a^{f_a}$ and $P_b^{f_b}$
output : A new individual $C_o^{f_o}$ not yet in the population

$P_i^{f_i}$: An individual $i$ in the population with fitness $f_i$
$P_i^{f_i}$ $[g]$: The $g$'th gene in the genotype of an individual $i$
$L$: The length of the genotype (i.e. the number of genes)
$k$: The pivot value

rand_float($a, b$): Function that draws a random number uniformly from
the range $[a, b]$

begin
    $k \leftarrow$ rand_float($0, 1$)
    for $g_a \in [0, k)$ do
        $C_o^{f_o}[g_a] \leftarrow P_a^{f_a}[g_a]$
    end
    for $g_b \in [k, L)$ do
        $C_o^{f_o}[g_b] \leftarrow P_b^{f_b}[g_b]$
    end
end
```

Algorithm 5: Uniform Crossover

```
input  : Two selected parents $P_a^{f_a}$ and $P_b^{f_b}$, the gene bias factor $b \in [0, 1]$
output : A new individual $C_o^{f_o}$ not yet in the population

$P_i^{f_i}$: An individual $i$ in the population with fitness $f_i$
$P_i^{f_i}[g]$: The $g$'th gene in the genotype of an individual $i$
$l$: The length of the genotype (i.e. the number of genes)

rand_float($a, b$): Function that draws a random number uniformly
from the range $[a, b]$

begin
    for $g \in [0, l)$ do
        $r \leftarrow$ rand_float($0, 1$)
        if $r \leq b$ then
            $C_o^{f_o}[g] \leftarrow P_a^{f_a}[g]$
        else
            $C_o^{f_o}[g] \leftarrow P_b^{f_b}[g]$
        end
    end
end
```

Algorithm 6: Local Crossover

```
input  : Two selected parents $P_a^{f_a}$ and $P_b^{f_b}$
output : A new individual $C_o^{f_o}$ not yet in the population

$P_i^{f_i}$: An individual $i$ in the population with fitness $f_i$
$P_i^{f_i}[g]$: The $g$'th gene in the genotype of an individual $i$
$l$: The length of the genotype (i.e. the number of genes)
$\alpha$: The random weighting factor

rand_float($a, b$): Function that draws a random number uniformly
from the range $[a, b]$

begin
    for $g \in [0, l)$ do
        $\alpha \leftarrow$ rand_float($0, 1$)
        $C_o^{f_o}[g] \leftarrow (\alpha * P_a^{f_a}[g] + (1 - \alpha) * P_b^{f_b}[g])$
        Clip the value if necessary
    end
end
```

Algorithm 7: Random Mutation

input : The individual to be mutated $P_c^{f_c}$, the rate of mutation $r \in [0, 1]$
output : The possibly mutated individual $P_m^{f_m}$

$P_i^{f_i}$: An individual i in the population with fitness f_i
$P_i^{f_i}$ $[g]$: The g'th gene in the genotype of an individual i
$P_c^{f_c}$: The individual to be mutated
l: The length of the genotype (i.e. the number of genes)

`rand_float`(a, b): Function that draws a random number uniformly from the range $[a, b]$

```
begin
    for $g \in [0, l)$ do
        $rnd \leftarrow$ rand_float$(0, 1)$
        if $rnd \leq r$ then
            $P_m^{f_m}[g] \leftarrow$ rand_float(min,max)
            Clip the value if necessary
        else
            $P_m^{f_m}[g] \leftarrow P_c^{f_c}[g]$
        end
    end
end
```

Algorithm 8: Gaussian Mutation

input : The individual to be mutated $P_c^{f_c}$, the rate of mutation $r \in [0, 1]$, the mutation precision p which defaults to 10

output : The possibly mutated individual $P_m^{f_m}$

$P_i^{f_i}$: An individual i in the population with fitness f_i
$P_i^{f_i}$ $[g]$: The g'th gene in the genotype of an individual i
l: The length of the genotype (i.e. the number of genes)

`rand_float(`a, b`)`: Function that draws a random number uniformly from the range $[a, b]$
`next_gaussian(`$mean, std\_dev$`)`: Function that returns a number drawn at random from an EAussian distribution

```
begin
    $sd \leftarrow (max - min)/p$
    for $g \in [0, l)$ do
        $rnd \leftarrow$ rand_float($0, 1$)
        if $rnd \leq r$ then
            $P_m^{f_m}[g] \leftarrow (P_c^{f_c}[g] +$ next_gaussian($0, sd$)$)$
            Clip the value if necessary
        else
            $P_m^{f_m}[g] \leftarrow P_c^{f_c}[g]$
        end
    end
end
```

Algorithm 9: Breeder Mutation

```
input  : The individual to be mutated $P_c^{f_c}$, the rate of mutation $r \in [0, 1]$,
         a percentage of the range of allowed values $p\_range \in [0, 1]$
         dictates the max step size of the mutation and defaults to 0.1 (i.e.
         10%), the mutation precision $p$ which defaults to 4
output : The possibly mutated individual $P_m^{f_m}$

$P_i^{f_i}$: An individual $i$ in the population with fitness $f_i$
$P_i^{f_i}[g]$: The $g$'th gene in the genotype of an individual $i$
$l$: The length of the genotype (i.e. the number of genes)

rand_float($a, b$): Function that draws a random number uniformly
from the range $[a, b]$
pow($a, b$): Function that raises $a$ to the power $b$ (i.e. $a^b$)

begin
    $step \leftarrow$ (rand_float($-1, 1$)$*p\_range * (max -$
    $min)*$pow($2, -1*$rand_float($0, 1$)$*p$))
    for $g \in [0, l)$ do
        $rnd \leftarrow$ rand_float($0, 1$)
        if $rnd \leq r$ then
            $P_m^{f_m}[g] \leftarrow (P_c^{f_c}[g] + step)$
            Clip the value if necessary
        else
            $P_m^{f_m}[g] \leftarrow P_c^{f_c}[g]$
        end
    end
end
```

Table 1: EA Settings

Parameter	Value
Encoding	real-valued
Population size	100
Max generations	40
Number of runs per operator	20
Number of fitness evaluations (ticks) per generation	900
Fitness function	Number of unique obstacles detected per tick (theoretical max of 10 per tick)

Table 2: Base EA Operator Instances And Their Parameter Values

Operator	Instance	Parameters	Values
Selection	Tournament	Tournament size	$k = 10$
Crossover	Uniform	Bias factor	$b = 0.5$
Mutation	Breeder	Rate	$r = 1/l$*
Replacement	Elitism	Percentage	$p = 1\%$

*l is the length of the genotype

3.3 Experiments

Table 1 and table 2 shows the operator instances and their parameter values used by the Base (or standard) EA, which was the control EA used to compare the other EAs defined by each experiment to. Table 1 shows the settings used by the EA during the simulations. Each new experiment used the same parameter values and operators as the base EA, but changed at least one of them, which was the parameter or operator being tested.

40 generations was defined to be the termination condition. Once that had occurred, the EA would be reinitialise with new parameter values from the next experiment, using the the same initial random population and obstacles layout as the other experiments, to unsure fairness. Due to the stochastic nature of EAs, multiple runs of the same experiment were necessary in order to get a fair account of the effect the parameters specified by the experiment had on the performance of the EA. There were a total of 20 runs for each experiment and every 5 runs, a new random

Table 3: Experiments for Round 1

Operator	Instance	Parameters	Values
Selection	Random	n/a	
Selection	Fitness Proportionate	n/a	
Crossover	One-Point	n/a	
Crossover	Local	n/a	
Mutation	Random	Rate	$r = 1/l$*
Mutation	Gaussian	Rate	$r = 1/l$*

*l is the length of the genotype

Table 4: Experiments for Round 2

Operator	Instance	Parameters	Values
Selection	Tournament	Tournament size	$k = 5$
Selection	Tournament	Tournament size	$k = 20$
Mutation	Breeder	Rate	$r = 3/l$*
Mutation	Breeder	Rate	$r = 0$

*l is the length of the genotype

set of obstacles were placed on the track and a new initial random population was generated. The purpose of these changes was to filter out the chance of a random population or obstacle layout benefiting one EA more than another.

Two rounds of experiments were performed. The first round (see Table 3) was focused on determining the effect different operators had on the search performance of the EA (the operator centric round) and the second round (see Table 4) was focused on altering the parameter values used by the operators to explore the effects different parameter values had (the parameter centric round).

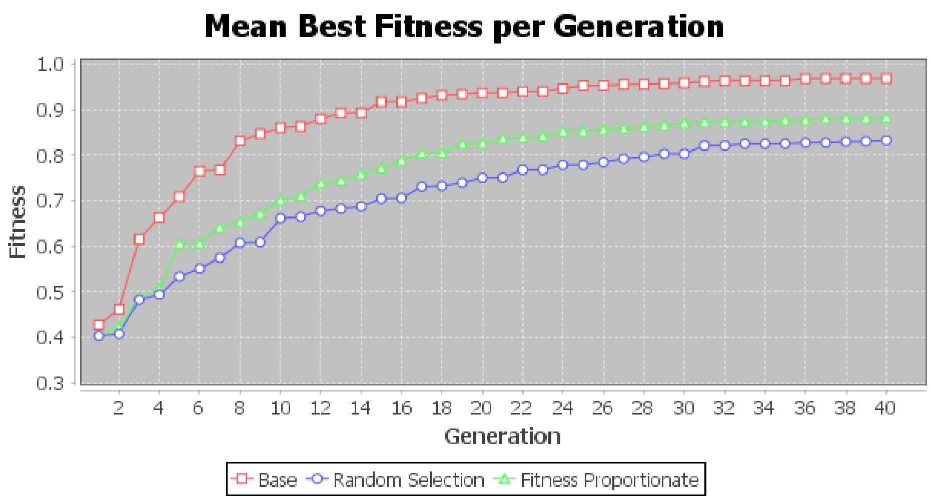

Figure 4: Showing the results of round 1 (see table 3), testing three different selection operators, namely Tournament (from the Base EA), Fitness Proportionate and Random

4 Results and Discussion

The results obtained from the two rounds of experiments conducted are present below along with a discussion on the trends observed that relate to an effect an operator had on the search performance of an EA. The EA defined by each experiment was compared to the Base EA (see Table 2 for details) in each case. The results show are an average of the best fitness achieved by each experiment over 20 runs (also called the Mean Best Fitness, MBF [12]) per generation. The *Mann-Whitney U* ($p \leq 0.05$) was used to test for statistical differences between the fitness values of the last generation (generation 40) as an indication of the relative performance an operator had compared to that of the Base EA (see table 5 for the p values of the results of the experiments compared to the results of the Base EA).

The trends in fitness observed between the three selection operators in round 1—Tournament, Random and Fitness Proportionate—(see figure 4) show that the selection operator has a large impact on the performance of the EA. This result concurs with the results from [26] and is in agreement with the hypothesis that selection will have the greatest impact on the performance of the EA (as stated in section 1). The *Mann-Whitney U* ($p \leq 0.05$) test indicates that the Tournament selection operator (from the Base EA) significantly out-performed both the Random and Fitness Proportionate operators. This may be due to the difference between the selection pressure of the three operators. The Fitness Proportionate selection operator has a high selection pressure, which results in sub optimal convergence and the Random selection operator does not bias fit individuals at all, thus grows the least. Figure 5 shows the results of the experiments in round 2, using the

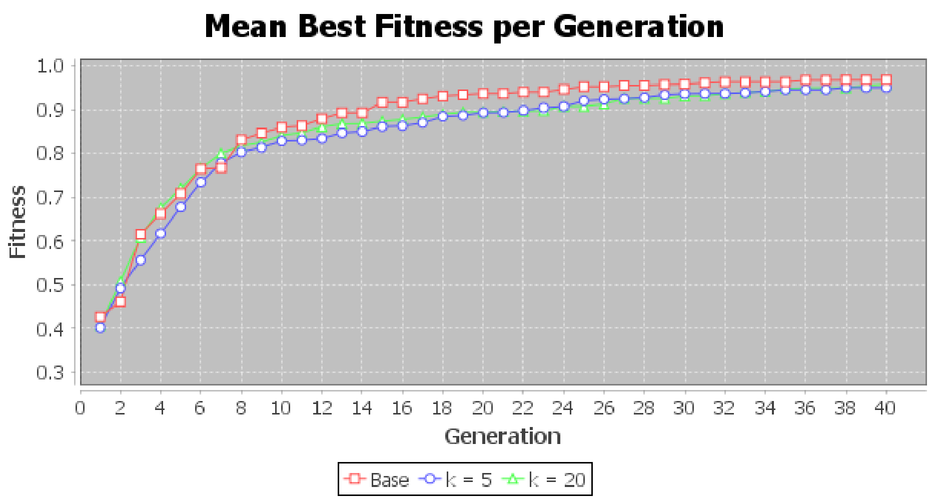

Figure 5: Showing the results of round 2 (see table 4), testing three different tournament sizes k of the Tournament Selection operator (Base had a tournament size of $k = 10$)

different tournament sizes for the Tournament selection operator. The *Mann-Whitney U* ($p \leq 0.05$) test indicates that there is no distinctive trend between the three tournament sizes of $k = 5$, $k = 20$ and $k = 10$ (the tournament size of the base EA). This was a surprising result as the selection pressure varies inversely to the tournament size, it would be expected to find a greater change in the trends of the three experiments, as seen in round 1. However, this may be attributed to not using more extreme tournament size values, as the difference between the effects of tournament sizes of 5,10 and 20 was, evidently, negligible. Thus, no concrete conclusion about the effect selection pressure has on the performance of the EA may be made, as both a very high selection pressure (Fitness Proportionate) and a very low, nonexistent, selection pressure (Random selection) reduced the performance the EA and it was hoped that the values of the tournament sizes would shed light on the matter, however, that was not the case. But, it may be said that the selection pressure does play have important impact on the performance of an EA and that a moderate selection pressure is best (as is the case with a tournament size of $k = 10$).

The trends in fitness observed between the three crossover operators in round 1—Uniform, One-point and Local—(see figure 6) were not were not statistically different (*Mann-Whitney U* ($p \leq 0.05$) test) when compared to the Uniform (from the Base EA) operator. This indicates that the relative effect (or benefit thereof) of choosing one crossover operator over another, may be little to none. This lies in contrary to what was found by Picek et. al. [30]. The reason for this may be due to the fact that only three different operators were test in this paper and fifteen were tested by Picek et. al. Additionally, the results from the tests performed by Picek

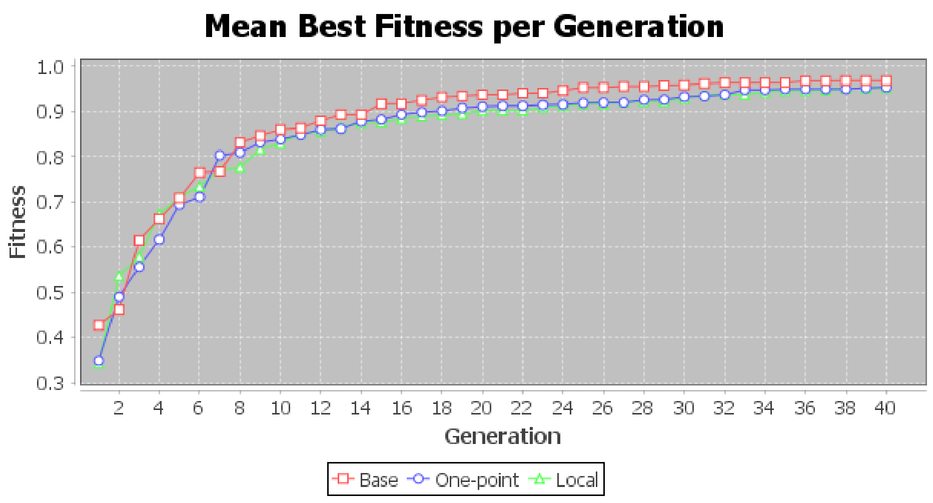

Figure 6: Showing the results of round 1 (see table 3), testing three different crossover operators, namely Uniform (from the Base EA), One-point and Local

et. al. show that the One-point and Uniform crossover operators for multi-modal problems (as is the one in this paper) perform very similarly (the Local crossover operator was not tested on the multi-modal problem by them).

The trends observed in fitness between the three mutation operators in round 1—Breeder, Random and Gaussian—(see figure 7) show that the mutation operator has a substantial impact on the performance of the the EA. The Breeder and Gaussian mutation operators performed very similarly and were not statistically different (*Mann-Whitney U* ($p \leq 0.05$) test). Both significantly outperformed the Random mutation operator (*Mann-Whitney U* ($p \leq 0.05$) test) indicating that a mutation operator that is based on the values of the genes themselves, can allow for ways of escaping local optima. The mutation operator is vitally important to the evolutionary process as genetic information is always lost with subsequent generations. Figure 8 shows the results of the experiments from round 2, using different rates of mutation for the Breeder mutation operator. With a mutation rate of 0 ($r = 0$), the EA is unable to break out the local optima and which results in significantly worse fitness at the end compared to the rate of $r = 1/l$ (where l is the length of the genotype, 10 in this case) from the Base EA (*Mann-Whitney U* ($p \leq 0.05$) test). The mutation rate of $r = 3/l$ and that of the Base EA ($r = 1/l$) showed statically different trends (*Mann-Whitney U* ($p \leq 0.05$) test) with the mutation of rate of $r = 3/l$ performing worse. The increase in the mutation rate led to an inability of the EA to climb to higher fitness values, as there are too many mutations, which resulted in too great a loss in the evolved gene values. Thus the optimum mutation rate was found to be $1/l$ which coincides with the studies by Hesser and Manner [20], Back et. al. [6] and is in agreement with the hypothesis that the optimal mutation rate is $1/l$.

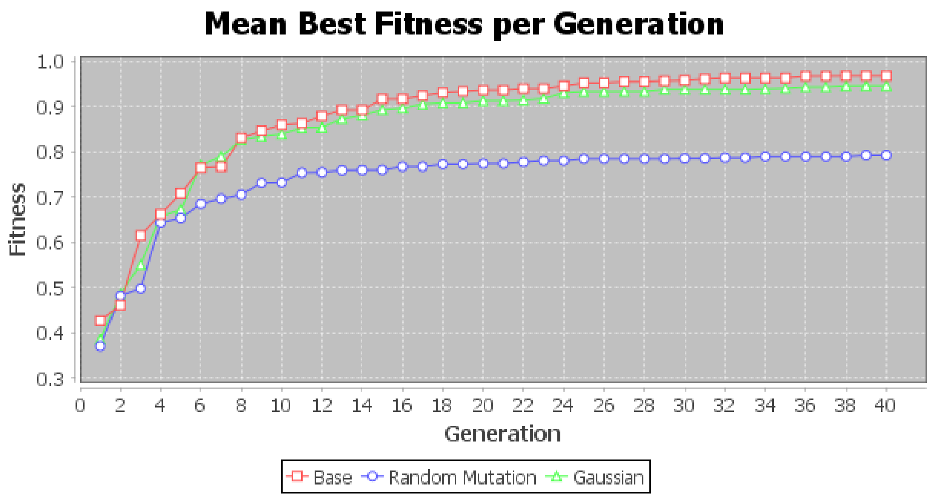

Figure 7: Showing the results of round 1 (see table 3), testing three different mutation operators, namely Breeder (from the Base EA), Random and Gaussian

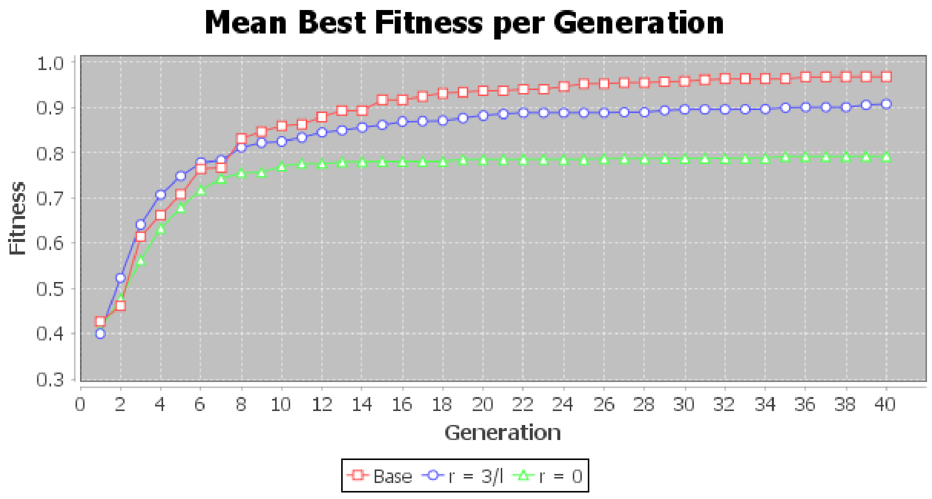

Figure 8: Showing the results of round 2 (see table 4), testing the three mutation rates r of the Breeder mutation operator (Base had a mutation rate of $r = 1/l$, where l was the length of the genotype, 10 in this case)

Table 5: Results of the *Mann-Whitney U* test showing the statistical difference between the fitness of the final generation (generation 40) for each of the conducted experiments and that of the Base EA. Experiments that had p values of 0.05 or less were considered to be statistically different

Experiment	p value	Statistically Different
Random selection	8.263e-8	Yes
Fitness Proportionate selection	2.258e-6	Yes
One-Point crossover	0.2942	No
Local crossover	0.1426	No
Random mutation	3.943e-8	Yes
Gaussian mutation	0.0994	No
Tournament size $k = 5$	0.2487	No
Tournament size $k = 5$	0.2524	No
Mutation rate $r = 3/l$	0.0027	Yes
Mutation rate $r = 0$	3.125e-7	Yes

5 Conclusion and Future Work

In this paper, two rounds of experiments were conducted in order to test three hypotheses which probed the effect different parameter values had on the search performance of an EA and in doing so, effectively parameter tuning the EA. Overall, the untuned 'Base EA' performed just as well or better than the others that were tested. The results of the experiments showed that different selection and mutation operators had the greatest impact on the search performance of the EA, which concurs with the work by Nannen, Smit and Eiben [26]. The Tournament selection operator with a tournament size of 10 preformed the best when compared to the Fitness Proportionate and Random selection operators. The effect of selection pressure was not clearly evident in the results obtained by the experiments, however, the effect of the mutation rate was. No mutation resulted in the sub-optimal convergence as did a mutation rate that was too high. It was found that $1/l$ (the reciprocal of the genotype length) was the most optimal mutation rate, which agrees with an with the works by Hesser and Manner [20], Back et. al. [1], [6]. There was little to no difference between different crossover operators tested in this paper, which contradicts what was found by Picek et. al. [30], however, the results they obtained for the crossover operators that were tested in this paper and that were also tested by them, did not differ significantly in performance either. In future work, a thorough testing of tournament sizes would show more clearly the effect selection pressure has and testing many more crossover operators would provide better detail on effects crossover operators have (if any at all).

Acknowledgements

Many thanks to Geoff Nitschke for the guidance given to me throughout this research project and Allen Huang for collaborating with me, helping me preform the runs of the experiments and analyse the results.

References

[1] BÄCK, T. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In *PPSN* (1992), pp. 87–96.

[2] BÄCK, T., FOGEL, D. B., AND MICHALEWICZ, Z. *Evolutionary computation 1: Basic algorithms and operators*, vol. 1. CRC Press, 2000.

[3] BÄCK, T., AND SCHWEFEL, H.-P. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation 1*, 1 (1993), 1–23.

[4] BOOKER, L. B., GOLDBERG, D. E., AND HOLLAND, J. H. Classifier systems and genetic algorithms. *Artificial intelligence 40*, 1 (1989), 235–282.

[5] BRIDGES, C. L., AND GOLDBERG, D. E. An analysis of reproduction and crossover in a binary-coded genetic algorithm. *Grefenstette 878* (1987), 9–13.

[6] BCK, T. Optimal mutation rates in genetic search. In *Proceedings of the fifth International Conference on Genetic Algorithms* (1993), Morgan Kaufmann, pp. 2–8.

[7] DARWIN, C. On the origin of species by means of natural selection, or. *The Preservation of Favoured Races in the Struggle for Life, London/Die Entstehung der Arten durch natürliche Zuchtwahl, Leipzig oJ* (1859).

[8] DE JONG, K. Genetic algorithms: a 30 year perspective. *Perspectives on Adaptation in Natural and Artificial Systems 11* (2005).

[9] DE JONG, K. A. *Evolutionary computation: a unified approach.* 2006.

[10] DEB, K., AND AGRAWAL, S. Understanding interactions among genetic algorithm parameters. *Foundations of Genetic Algorithms* (1999), 265–286.

[11] EIBEN, A. E., HINTERDING, R., AND MICHALEWICZ, Z. Parameter control in evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on 3*, 2 (1999), 124–141.

[12] EIBEN, A. E., AND SMIT, S. K. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation 1*, 1 (2011), 19–31.

[13] EIBEN, A. E., AND SMITH, J. E. *Introduction to evolutionary computing*. Springer Science & Business Media, 2003.

[14] GOLDBERG, D. E. Genetic algorithm in search, optimization and machine learning, addison. *Wesley Publishing Company, Reading, MA 1*, 98 (1989), 9.

[15] GOLDBERG, D. E., AND RICHARDSON, J. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms* (1987), Hillsdale, NJ: Lawrence Erlbaum, pp. 41–49.

[16] GREFENSTETTE, J. J. Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on 16*, 1 (1986), 122–128.

[17] HANCOCK, P. J. An empirical comparison of selection methods in evolutionary algorithms. In *Evolutionary Computing*. Springer, 1994, pp. 80–94.

[18] HARP, S. A., SAMAD, T., AND GUHA, A. Towards the genetic synthesis of neural network. In *Proceedings of the third international conference on Genetic algorithms* (1989), Morgan Kaufmann Publishers Inc., pp. 360–369.

[19] HERRERA, F., LOZANO, M., AND VERDEGAY, J. L. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial intelligence review 12*, 4 (1998), 265–319.

[20] HESSER, J., AND MÄNNER, R. Towards an optimal mutation probability for genetic algorithms. In *Parallel Problem Solving from Nature*. Springer, 1991, pp. 23–32.

[21] HOLLAND, J. H. Adaptation in natural and artificial system: an introduction with application to biology, control and artificial intelligence. *Ann Arbor, University of Michigan Press* (1975).

[22] JONG, K. D. Adaptive system design: a genetic approach. *Systems, Man and Cybernetics, IEEE Transactions on 10*, 9 (1980), 566–574.

[23] KRISHNAKUMAR, K., AND GOLDBERG, D. E. Control system optimization using genetic algorithms. *Journal of Guidance, Control, and Dynamics 15*, 3 (1992), 735–740.

[24] MITCHELL, M. *An introduction to genetic algorithms*. MIT press, 1998.

[25] MÜHLENBEIN, H. The breeder genetic algorithm-a provable optimal search algorithm and its application. In *Applications of Genetic Algorithms, IEE Colloquium on* (1994), IET, pp. 5–1.

[26] NANNEN, V., SMIT, S. K., AND EIBEN, A. E. Costs and benefits of tuning parameters of evolutionary algorithms. In *Parallel Problem Solving from Nature–PPSN X*. Springer, 2008, pp. 528–538.

[27] Ochoa, G., Harvey, I., and Buxton, H. Optimal mutation rates and selection pressure in genetic algorithms. In *GECCO* (2000), Citeseer, pp. 315–322.

[28] Oliver, I., Smith, D., and Holland, J. R. Study of permutation crossover operators on the traveling salesman problem. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA* (1987), Hillsdale, NJ: L. Erlhaum Associates, 1987.

[29] Ortiz-Boyer, D., Hervás-Martínez, C., and García-Pedrajas, N. Improving crossover operator for real-coded genetic algorithms using virtual parents. *Journal of Heuristics 13*, 3 (2007), 265–314.

[30] Picek, S., Jakobovic, D., and Golub, M. On the recombination operator in the real-coded genetic algorithms. In *Evolutionary Computation (CEC), 2013 IEEE Congress on* (2013), IEEE, pp. 3103–3110.

[31] Wright, A. H. Genetic algorithms for real parameter optimization. *Foundations of genetic algorithms 1* (1991), 205–218.