Optimization of tandem solar cells with genetic algorithms

Bachelor thesis report

Koen W. van Arem

Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

Optimaliseren van tandem zonnecellen aan de hand van genetische algoritmen

(Optimization of tandem solar cells with genetic algorithms)

Verslag ten behoeve van het
Delft Institute of Applied Mathematics
als onderdeel ter verkrijging

van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

Koen W. van Arem

Delft, Nederland
Juli 2021

BSc verslag TECHNISCHE WISKUNDE

"Optimization of tandem solar cells with genetic algorithms"

("Optimaliseren van tandem zonnecellen aan de hand van genetische algoritmen")

Koen W. van Arem

Technische Universiteit Delft

Begeleiders

Dr.ir. J.T. van Essen Dr.ir. R. Santbergen

Overige commissieleden

Dr. J.L.A. Dubbeldam

Juli, 2021 Delft

Contents

Page

Layman's summary vi

Abstract vii

Acknowledgements viii

1 Introduction 1
- 1.1 Tandem solar cell 1
- 1.2 Problem description 3
- 1.3 Optimization algorithms 4
- 1.4 Thesis outline 5

2 Literature 6
- 2.1 GenPro4 6
- 2.2 Optimization of solar cells 7
  - 2.2.1 Current-voltage characteristics 8
  - 2.2.2 Use of optimization algorithms 9
- 2.3 Genetic algorithms 12
  - 2.3.1 Representation of solutions 13
  - 2.3.2 Selection operators 14
  - 2.3.3 Variation operators 18
  - 2.3.4 Other parameters 19
  - 2.3.5 Matlab genetic algorithm function 20

3 Objective function 21
- 3.1 Distribution of the estimates 21
- 3.2 Improving accuracy 23
- 3.3 Conclusion 26

4 Genetic algorithms 27
- 4.1 Representation 27
- 4.2 Selection operators 28
- 4.3 Variation operators 29
- 4.4 Creation method 30
- 4.5 Population size 30
- 4.6 Stopping criterion 31
- 4.7 Parallel computing 34

5 Results 36
- 5.1 Results for different population sizes 36
  - 5.1.1 General results 36
  - 5.1.2 Population size 41
- 5.2 Stopping criterion 45
- 5.3 Parallel computing 52
- 5.4 Case study 54
  - 5.4.1 Focus on high quality of the solution 54
  - 5.4.2 Focus on a short computation time 55
- 5.5 Discussion of fixed settings 55
  - 5.5.1 Fixed settings of the objective function 55
  - 5.5.2 Fixed settings of the genetic algorithm 56

6 Conclusion 58

7 Recommendations 60

Bibliography 61

Appendices 63

A Genetic algorithm with inaccurate objective function 64

Layman's summary

Tandem solar cells are a new type of solar cells that are currently being developed. Soms proporties of these solar cells are variable. In this scription, it is studied how genetic algoritms can be applied to optimize these tandem solar cells. Genetic algoritms are algorithms that find a good solution using the principles of evolution theory. It is studied what the influences are of several choices. In this way, this scription gives which genetic algorithms should be applied in several situations. Hereby, it provides insight in how genetic algorithms can be applied to optimize genetic algorithms.

Abstract

The production of renewable energy is increasingly important. A source of renewable energy is solar energy, which can be produced using solar cells. Solar cells consist of multiple thin layers of specific materials. The energy of solar rays is converted into electrical energy by these layers. A type of solar cells that is currently being developed is a tandem solar cell, which consists of two solar cells on top of each other. This results in a complicated structure, which makes it hard to determine the influences of several parameters of the solar cell, such as used materials and thicknesses of the layers. The software of *GenPro4* provides the opportunity to estimate the current produced by a tandem solar cell. As a higher current gives a higher output of power, it is desirable to have a solar cell that produces the optimal current. This results in an optimization problem with discrete and continuous variables. In this thesis, it is studied how genetic algorithms can be applied to this optimization problem. Genetic algorithms are optimization methods, which means that they are designed to find a solution that is near or equal to the optimal solution. Genetic algorithms are based on the principles of evolution theory. The algorithms consider a population that develops over time. This is done by forming subsequent generations of this population that slowly improve to find a good value. In this way, a good solution is found for the optimization problem.

An important factor in the optimization problem is the implementation of the objective function, which is the current of the tandem solar cell. Because *GenPro4* estimates the current with a simulation, the objective function can be different for function calls with identical input. There exist several settings of *GenPro4* that provide the possibility to change the accuracy of the objective function and decrease the variation within identical function calls. The influence of the settings are studied and the best settings are found. The found settings are 30 angular intervals, 1000 simulation of rays and a step size of numerical integration of 0.005μm. In this way, *GenPro4* estimates the current with an accuracy of 0.01 mA/cm^2 in 21.8 seconds.

Other important settings that influence the performance of the genetic algorithm are settings of the genetic algorithm itself. Settings that increase the chance of finding a good solution often result in a long computation time. This creates a trade-off between a good final solution and a reasonable computation time. One of these settings is the population size, which is studied by running the genetic algorithm multiple times with different population sizes. In this way, it is found that a larger population size improves the chance of the genetic algorithm to find a good solution. On the other hand, a large population size also gives a long computation time. In a situation in which a solution is desired quickly, a population size between 50 and 100 is found to be a good choice. When the solution should be as good as possible and computation time does not matter that much, a population size between 100 and 150 is a better choice. In this way, insight is found in the trade-off between a short computation time and a good solution.

Another important setting is the stopping criterion. Due to the accuracy and the high computational costs of the objective function, the default stopping criterion of the Matlab function for a genetic algorithm does not function properly. Using the information about the genetic algorithms with different population sizes, two new possible stopping criteria are found. One can be used when the computation time of the genetic algorithm should be low at the cost of a possibly worse solution. The other stopping criterion can be used when the focus should be on finding a good solution.

The last setting that is studied is the use of parallel computing. With parallel computing, some functions are called at the same time independently on multiple processors to reduce the computation time. Parallel computing can either be applied within *GenPro4* or within the genetic algorithm itself. The influences of these two implementations are studied. They do not change the probability of a genetic algorithm to find a good solution. However, both implementations do decrease the computation time. The largest decrease is obtained by implementing parallel computing within the genetic algorithm itself. In this way, the computation time can be decreased up to 75%.

In this thesis, genetic algorithms are applied to the optimization problem to test their performance. The solutions that are found range from 20.79 mA/cm^2 to 21.10 mA/cm^2. These different studies resulted in insight in the trade-off of the genetic algorithm between finding a good solution and finding a solution within a short computation time. It is found that the objective function should have accurate settings and parallel computing should be used for all implementations of genetic algorithms for this problem. Two settings are provided: one for finding a high quality solution and one for finding a good solution within a short computation time.

Acknowledgements

Throughout working on this project, I received help and support from several people that I would like to thank.

Firstly, I would like to thank my supervisors Theresia van Essen and Rudi Santbergen for providing me with the possibility to partake in this project. Not only did I enjoy the weekly meetings, but I also learned a lot of academical skills during them. Because of the given advice and assistance, I enjoyed working on this thesis, even despite the limitations regarding COVID-19. More specifically, I would like to thank Theresia for providing a lot of helpful feedback. I would like to thank Rudi for the assistance in fine-tuning the settings of *GenPro4*.

Secondly, I would like to thank my parents and my girlfriend for the personal support they gave me. They helped me to stay motivated during the project and to put the situation into perspective from time to time.

1 Introduction

The production of renewable energy has recently become increasingly important. As the emissions due to fossil fuels are an important factor in global warming and can have a negative influence on the environment and human health, it is beneficial to reduce the use of it [2]. Since the usage of renewable energy sources provides the opportunity to use less fossil fuels, it is an important part of the solution to these issues.

The importance of this problem has been recognised by almost all governments. By signing the Paris Agreement of 2015, 197 countries agreed that it is important to make sure that the rise in the global temperature is kept well below 2 °C by 2050. This can be done by reducing the emission of greenhouse gasses, which are mostly caused by the usage of fossil fuels according to the International Renewable Energy Agency (IRENA) [1]. IRENA even states that “accelerated deployment of renewable energy and energy efficiency measures are the key elements of the energy transition". According to their research, it is possible to achieve 90% of the needed reduction of emission necessary to accomplish the goals set by the Paris Agreement by focusing on renewable energy sources and more efficient energy usage.

A well-known renewable energy source is solar energy. The usage of this renewable energy source has grown the last decades. This is illustrated by data about the Dutch use of energy sources, provided by the CBS [19]. In 2010, 56 million kWh of electricity was produced in the Netherlands by means of solar energy. Almost ten years later, in 2019, this increased to 5170 million kWh. Next to that, solar energy produced 0.046% of the Dutch electricity in 2010 compared to 4.23% in 2019. From this data, it can be concluded that solar energy has a growing importance in the production of electricity.

The solar energy that is used for electricity consumption is created using solar cells. The increasing importance of solar energy therefore gives an increasing importance to solar cells. Solar cells are found on solar panels and are created by placing different materials in layers on top of each other to catch the energy of photons and convert it into an electrical charge. In this way, the energy of sunlight can be converted into electrical energy. The amount of the current a solar cell produces depends on properties of the materials used and the way they are used, for instance the thickness of the layers [4]. Using physical models, it is possible to calculate the amount of the current produced by a certain solar cell.

Such a physical model was presented in [25] and is called *GenPro4*. It gives the opportunity to calculate the absorbance of the energy of photons of different wavelengths for the different layers of a solar cell. This model computes the result relatively fast, especially when only the size of layers are adjusted. With this, the possibility arises to study the optimal combination of layer sizes for specific types of solar cells.

1.1. Tandem solar cell

A type of solar cell which is currently being developed is a so-called tandem perovskite/silicon solar cell [4]. This type of solar cells consists of two parts on top of each other, called interfaces of materials which both absorb photons of different wavelength. In this way, the solar cell is able to convert the energy of more photons in electrical energy. One interface of the perovskite/scilicium solar cell has as main element perovskite and the other interface scilicium. Next to these materials, the interfaces also consist of different smaller layers. Both interfaces together consist of 10 layers in total.

In order to understand how the two different interfaces interact and create a current, it is necessary to understand how a traditional solar cell works. Therefore, this is discussed first. The general structure of a solar cell is depicted in Figure 1.1. The solar cell consists of an absorber with on one side a layer of electron transporting material (ETM) and on the other side a layer of hole transporting material (HTM) [4]. Both the ETM and HTM layers are connected with contacts, which can be connected to a electric circuit.

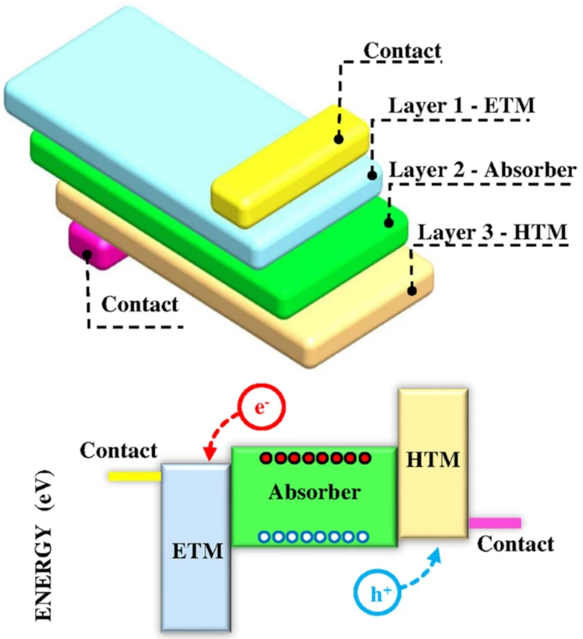


Figure 1.1: A visualisation of the general structure of a solar cell [4].

Sunlight consists of photons and when sunlight shines on a material, these photons will either reflect, go through it or be absorbed by the molecules of the material. The solar cell is made in such a way that some photons are absorbed by the absorber layer. This causes an electron in the absorber layer to go to a higher energy level and this makes it possible for the electron to move a little bit. The electron then moves to the electron transporting material. Since the electron has a negative charge, a positive hole will remain in the absorber. This hole moves to the hole transporting material. In this process, a negative charge is obtained by the ETM and a positive charge is obtained by the HTM. This difference in charge is potential electrical energy and the corresponding voltage is dependent on the materials used. The electrons in the absorber material need a photon to have a minimal value of energy in order to be able to bring the electron to a higher energy level. This minimal value determines the voltage. A higher voltage or a higher current results in a higher potential energy generated by a solar cell. In short, the energy of the photon is absorbed by this process and converted to electrical energy.

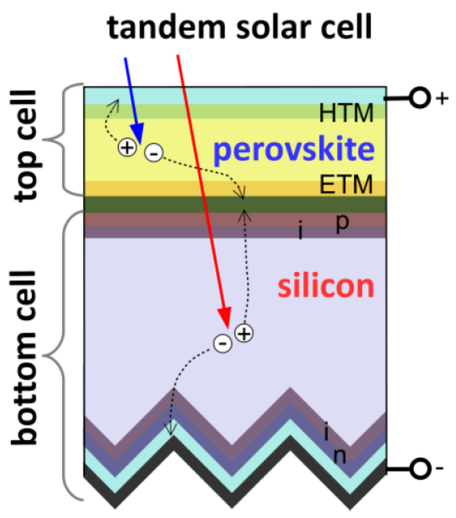


Figure 1.2: A visualisation of a perovskite/silicon tandem solar cell.

A tandem solar cell can be seen as two solar cells stacked on top of each other with only two contacts [8]. Because of this, the tandem solar cell consists of 10 different layers. The structure of a tandem solar cell is visualised in Figure 1.2. The tandem cell has two different absorbers, which are perovskite and silicon. Both materials absorb photons with different wavelengths. In this way, more photons can be converted to electrical energy.

On top of this, a part of the photons which are not absorbed by the upper interface can be reflected by the lower one. Therefore, the photon has an extra possibility to be absorbed by the upper layer and more energy can be converted by a tandem solar cell.

Since the contacts of the solar cell are only at the upper and lower layer, the two interfaces are connected in series. This means that the negative ETM of the perovskite interface and the positive HTM of the silicon interface are next to each other. Therefore, the electrons created by the perovskite part and the holes generated by the silicon interface meet and cancel each other out. The charge created by the total tandem cell is thus caused by the positive holes of the perovskite interface and the negative electrons of the silicon interface. The current of the cell is equal in both interfaces since they are connected in series and is therefore determined by the interface which generates the lowest current. This means that it is necessary for a tandem cell that both interfaces create a current as high as possible to have the highest value of the current.

In a tandem cell, one electron in the perovskite and one electron in thesilicon must both absorb photons to create a charge. This causes the voltage of the charge to be larger compared to a situation with only one cell. Therefore, the total electrical power created by the solar cell can be higher compared to a normal solar cell when the decrease of the highest current is as low as possible.

To summarize, a general solar cell consists of an ETM, absorber and an HTM. The perovskite/silicon tandem solar cell consists of two solar cells, one with absorber perovskite and the other withsilicon, stacked on top of each other. In this way, the solar cell can absorb photons with a broader spectrum of wavelengths.

1.2. Problem description

The *GenPro4* model presented in [25] can be used to simulate tandem solar cells. In order to do this, the model needs the properties of the used materials and the thickness of the layers. *GenPro4* then calculates the obtained current under the given conditions. It can therefore be seen as a function with the thickness of the layers and the materials used as input variables and the current as outcome. In order to store as much solar energy as possible, it is necessary to find a combination of input variables that outputs a current as high as possible. Because of the complexity of the model used by *GenPro4*, this is not a linear function. On top of this, it is not known whether the function is smooth or nicely behaving. Therefore, it is nontrivial which combination of variables results in the highest efficiency.

The problem that rises is a nonlinear optimization problem. There exist multiple methods for such problems, for example the use of genetic algorithms [4]. Genetic algorithms are optimization heuristics based on the principle of genetics and evolution [17]. They exist in many variations and are therefore applicable in a broad range of problems. They have been applied to the problem of solar cells in for example [4]. In these studies, they are compared to other methods and are found to be good in finding a global maximum although they might need a lot of function evaluations. Therefore, it is interesting to compare the performance of different genetic algorithms on this optimization problem.

In order to test the performance of the genetic algorithms, a real-live case of the optimization of tandem solar cells is considered. Here, the materials used for the ETM and HTM of the perovskite interface can be chosen out of three different materials for each of them. On top of that, there is the possibility to use an anti-reflexive layer and a pyramid texture, which both might lead to a higher current. These four possible choices can be viewed as four discrete variables and define the structure of the solar cell. Next to the variations in the structure, there are six of the ten layers with variable thickness. These layers are the first six layers seen from the top of the cell in Figure 1.2 except for the contact layer. The layers are all included in the first interface of the solar cell. The values of their thicknesses can vary between 0.040 μm and 0.500 μmand these variables can be seen as continuous variables. In total, there are ten variables with four discrete and six continuous variables.

The six continuous variables can be denoted by $x_1, x_2..., x_6$ and the four discrete variables as $x_7, ..., x_{10}$. Together they can be seen as one vector, $\mathbf{x} = (x_1, x_2, ..., x_{10})$. For each $\mathbf{x}$, *GenPro4* can calculate the corresponding current. This can be denoted as $f(\mathbf{x})$ or $f(x_1, ..., x_{10})$. Using this notation, it is possible to formulate the optimization problem in a compact way. The problem that will be considered in this thesis is shown in Problem (1.1).

$$
\begin{array}{llllr}
\max\limits_{x_1,\dots,x_{10}} & f(x_1,...,x_{10}) & & & (1.1a) \\
\text{subject to:} & x_i \in \mathbb{R} & \forall i = 1,2,3,4,5,6 & \text{(Thickness variables)} & (1.1b) \\
 & 0.040 \le x_i \le 0.500 & \forall i = 1,2,3,4,5,6 & \text{(Boundaries thickness)} & (1.1c) \\
 & x_7 \in \left\{\text{spiro-OMeTAD}, \text{FTO}, \text{TiO}_2\right\} & i = 7 & \text{(ETM)} & (1.1d) \\
 & x_8 \in \{\text{PTAA}, \text{NiO}\} & i = 8 & \text{(HTM)} & (1.1e) \\
 & x_9 \in \{0,1\} & i = 9 & \text{(With/without AR-coating)} & (1.1f) \\
 & x_{10} \in \{0,1\} & i = 10 & \text{(With/without pyramide structure)} & (1.1g)
\end{array}
$$

Using this case study, different genetic algorithms can be assessed by their performance. The performance can be compared using the found maximum for each method and the number of function evaluations. By means of said, it is possible to give an answer to the question: how can genetic algorithms best be used to optimize tandem solar cells? The answer to this research question makes it possible to make the search for the optimal properties of tandem solar cells straightforward. Finding the answer to this research question will therefore be the main goal of this bachelor thesis.

1.3. Optimization algorithms

The mathematical field of optimization studies how to find a good solution for a given problem. If the maximal value of a function should be found, this function is called the objective function and the problem is called a maximization problem. In this thesis, the objective function is the value of the current produced by a tandem solar cell. There exist several analytical techniques to find the maximum of a function. However, the function most behave nicely for these to work. More importantly, knowledge about the function is necessary, such as a formula. If this is not available, other methods have to be used to find the maximum. In these situations, optimization algorithms are often used.

Maximization algorithms are methods that describe how to find a maximum of a function. These methods do not necessarily find the highest function value that is possible, but often get stuck in so-called local maximums. A local maximum is a point on the function that is higher than the points in its neighborhood. The overall highest function value is called the global maximum. As the global maximum is higher than all points, this is also a local maximum. Therefore, the global maximum can be seen as the highest local maximum. To illustrate this, Figure 1.3 shows the graph of the function $f(x) = x \cdot \sin(x)$ on the interval $[0,10]$. The points on the graph with $x \approx 2$ and $x \approx 8$ have a higher function than the surrounding points. Therefore, these are local maximums. The local maximum at $x \approx 8$ is the highest value on the graph and the highest local maximum. Thus, this is the global maximum.

When a function should be maximized, the goal is to find the global maximum. Optimization methods however often converge to a local maximum instead of a global maximum. This means that the method does

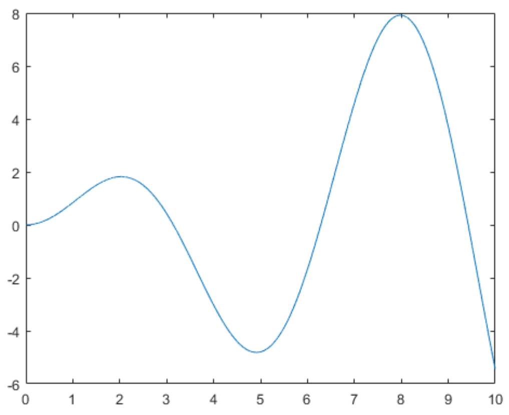


Figure 1.3: The graph of the function $f(x) = x \cdot \sin(x)$ on the interval $[0, 10]$.

not find the best solution. Therefore, optimization methods might sometimes not find the result that is desired. There exists different types of optimization methods, with different behavior for this problem [17]. Local optimization methods converge fast, but often find a local maximum. On the other hand, global optimization methods are more likely to find the global maximum, but they often take more computation time.

Problem (1.1) is a maximization problem with 10 input variables and this problem most likely has local optima in which local optimization methods are probably to get stuck. This results in a worse solution than could be achieved. A global computation method on the other hand is more likely to find the global optimum. This would therefore be better for this optimization problem. Genetic algorithms are global optimization methods. This means that they have the possibility to find the global optimum when they are implemented in the right way. As Problem (1.1) most likely contains local maximums, the genetic algorithms could be suitable optimization algorithm for this problem.

On top of that, *GenPro4* uses simulations to determine the current, which makes the objective function subject to small changes. A genetic algorithm is a robust method and does therefore perform well despite these changes in the objective function. This makes it likely that a genetic algorithm performs well for this type of problem. Therefore, the optimization method chosen for this optimization problem is a genetic algorithm.

1.4. Thesis outline

In order to study how genetic algorithms can best be used to optimize tandem solar cells, it is first important to investigate existing theory relevant for this thesis, which is be done in Chapter 2. After this orientation, it is discussed in Chapter 3 how the objective function can best be implemented using *GenPro4*. Subsequently, it is discussed how genetic algorithms can be implemented for this optimization problem and how the influence of certain settings can be studied. This is be done in Chapter 4. The results of these implementations are given and discussed in Chapter 5. Finally, an answer to the research question is be formulated in Chapter 6 using the found results.

2 Literature

The first step in finding an answer to the research question is a literature study. In this chapter, existing literature with importance to this thesis is discussed. First, the modeling software *GenPro4* is discussed. This model is important to this thesis because it provides a value of the current, which should be optimized by a genetic algorithm. Secondly, literature about optimization of solar cells is discussed in order to get a better knowledge of possible behavior of the problem. These properties provide the context of the problem and can help to understand how certain versions of genetic algorithms might work better. At last, the concepts of a genetic algorithm are discussed and different suitable versions of genetic algorithms are treated. This results in possible variations of genetic algorithms which can be used on the given optimization problem. Combined, these three parts provide a basis for the research that is carried out in later chapters.

2.1. GenPro4

As mentioned before, the objective function $f(\mathbf{x})$ in Problem (1.1) represents the value of the current obtained by a simulation of the solar cell using *GenPro4* which is presented in [25]. Therefore, the method of *GenPro4* and its properties are important for this optimization problem. The most important properties for implementation and the result are discussed in this section.

The process of capturing the light and converting it into electricity in a solar cell is an called optical process in physics. A physical model that is based on ray optics is called an optical model. *GenPro4* is such an optical model and it estimates the current produced by a solar cell. Another type of model which is often used is based on wave optics. These models simulate the behavior of light by solving the Maxwell equations, and therefore, they take into account the way light behaves as electromagnetic waves. However, solving these Maxwell equations is computationally a hard task. Because of this, these models must be restricted to a small problem or take a really long time to compute. *GenPro4* is an accurate optical model that has a considerably lower computation time. This is favorable since in an optimization context, the model needs to be used repeatedly. To conclude, *GenPro4* is a fast and accurate optical model and is, therefore, useful for optimization problems of solar cells.

The *GenPro4* model considers the cell as layers of different materials with interfaces in between. When a light ray reaches a layer, it can either be reflected at the interface, absorbed or transmitted by the layer. This happens at every layer and is shown in Figure 2.1. *GenPro4* computes the reflectance, transmittance and absorptance of each layer for light rays of different wavelengths. When light rays are absorbed by the absorber material, they create a current as explained in Section 1.1. The model uses the reflectance, transmittance and absorbtance of each layer in a system of linear equations to compute the current created by a solar cell.

This system of linear equations has certain coefficients, which together form the so-called scattering matrices. The majority of the computation time of *GenPro4* is used to calculate these scattering matrices. But, when only the layer thicknesses are changed, these scattering matrices stay the same. Thus, computation time can be reduced in optimization problems when only the layer thicknesses are changed. This can help to reduce computation time when *GenPro4* is used in a genetic algorithm.

The fact that *GenPro4* is a ray-based method, causes that some physical phenomena which are described by electromagnetic wave behavior of light are not taken into account. When the thickness of the layers is large compared to the coherence length of the light, these phenomena can be neglected. But when this is not the case, inference effects occur and alter the behavior of the light significantly. The coherence length of the in-

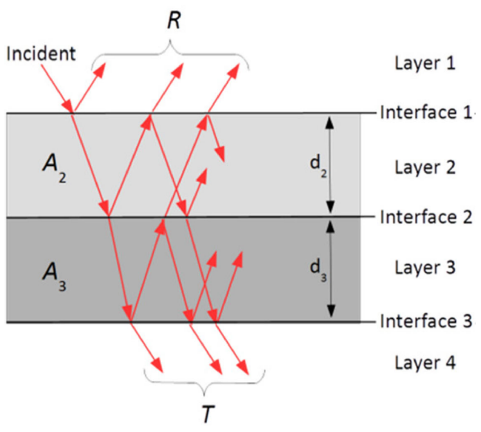


Figure 2.1: A visualisation of the reflectance (R), transmittance (T) and absorptance (A_2, A_3) in *GenPro4* simulating a solar cell [25].

cident sunlight is around 1 μm, which is larger than the possible thicknesses in the problem and, therefore, inference effects occur. To deal with such situations, *GenPro4* has the option to see a layer as a “coating". When a layer is considered as a coating, a different calculation method is used which takes inference into account. The layers in the problem can therefore still be used in *GenPro4* when they are seen as coatings.

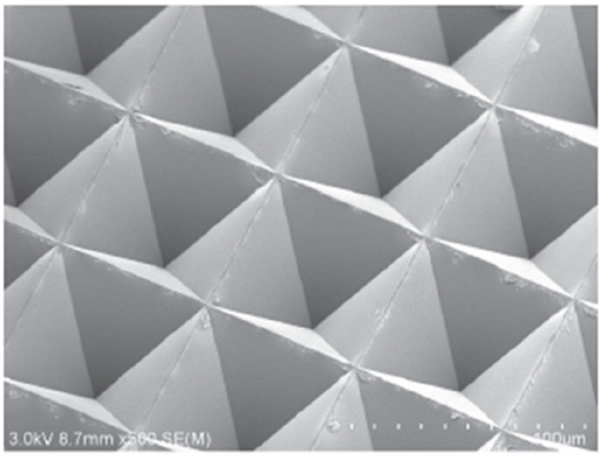


Figure 2.2: A SEM image of anti-reflective texture foil with an inverted pyramid texture[25].

GenPro4 also has other options to deal with more complex solar cells. As can be seen in Figure 2.1, the interfaces are considered to be flat. However, *GenPro4* offers the possibility to add textures to the interfaces. These textures can be of different shapes, for instance pyramid-shaped as can be seen in Figure 2.2. The variable x_{10} represents whether there is a pyramid texture or not. Therefore, this option in *GenPro4* is essential for this optimization problem.

To summarize, *GenPro4* provides an optical model to simulate different types of solar cells including the tandem solar cell. Due to the fact that it is an optical model and other properties, it is computationally fast, and thus useful in the optimization of tandem solar cells when algorithms are used. On top of that, it still has a great deal of flexibility and has different features that make it possible to extend the model in such a way that other effects can be taken into account, for instance thin layers and interfaces with textures.

2.2. Optimization of solar cells

In order to better understand the problem, relevant literature is analyzed. A better understanding of the problem can help explaining the behavior of genetic algorithms and could help in fine-tuning the process of genetic algorithms. On top of that, existing studies can provide context of the problem, which might be helpful when interpreting the results of the genetic algorithms.

If a solar cell is made, the fabricator wants the solar cell to be as good as possible. Therefore, researchers have tried to find the properties of solar cells which result in the highest efficiency. In this section, different methods used to improve the efficiency of solar sells are discussed.

2.2.1. Current-voltage characteristics

The main goal of the optimization of solar cells is to find parameters which result in the highest possible efficiency. The efficiency is the proportion of solar energy that is converted to electrical energy. This efficiency can be expressed in formulas involving physical quantities. In most cases, the relationships between the quantities can be formulated or approximated. A method which is most frequently used to optimize the solar cells is to investigate the direct influence of parameters on certain physical quantities. Using these relations, it can be roughly found what the best set of parameters is.

There exist several methods to investigate the relation between the parameters of a solar cell and the efficiency. One method is based on the relation between the power (P), voltage (V) and current (I), which is described by $P = VI$ [5]. Optimizing the efficiency of a solar cell is the same as optimizing the power produced by the solar cell when the circumstances are fixed.

Every value of the variables gives rise to a so-called current-voltage characteristic, which can be visualised in an IV curve as is explained in [5]. An example of an IV curve is shown in Figure 2.3. Every point on the IV curve corresponds to a value of the power P. This can be calculated by multiplying the I-coordinate and the V-coordinate. This is also shown in Figure 2.3.

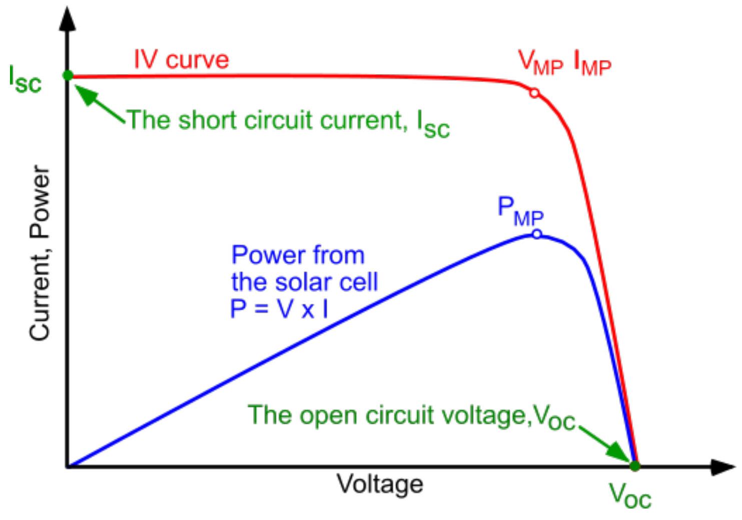


Figure 2.3: Current voltage (IV) curve of a solar cell (red) with for each point the the corresponding power (blue) [5].

The current of the circuit when the voltage over the solar cell is zero is called the short circuit current and is denoted by I_{sc}. This is the highest possible current. The voltage of the solar cell when there is no current over the circuit is called the open circuit voltage, V_{oc}. This is the highest voltage that can be achieved.

Let (v, i) be a point on the VI curve. Then $(v, 0)$ and $(0, i)$ are the orthogonal projections on respectively the V-axis and I-axis. These points form a rectangle A together with the origin, which is shown in Figure 2.4. The value corresponding to (v, i) is equal to $v \cdot i$, which is equal to the area of rectangle A. Then a point (V_{MP}, I_{MP}) can be defined as the point corresponding to the square with the largest area. On top of that, a similar rectangle B can be constructed with the point (V_{oc}, I_{sc}). This gives that rectangle A is contained in B. The area of A divided by the area of B is called the fill factor (FF) and it could be seen as how much the VI curve looks like a rectangle.

Using the defined quantities, the value of the efficiency of a solar cell, denoted by η, can be determined. This can be done using Equation (2.1), where P_{in} gives the value of the total power of the sunlight. From Equation (2.1) can be concluded that a higher efficiency can be obtained by a higher V_{oc}, I_{sc} and FF. Since the IV curves contain the information of all these quantities, the curves are a visual tool to asses the performance of a solar cell.

$$\eta = \frac{V_{oc} I_{sc} FF}{P_{in}} \tag{2.1}$$

In most studies, a good combination of parameters for solar cells is obtained by creating the IV curves for

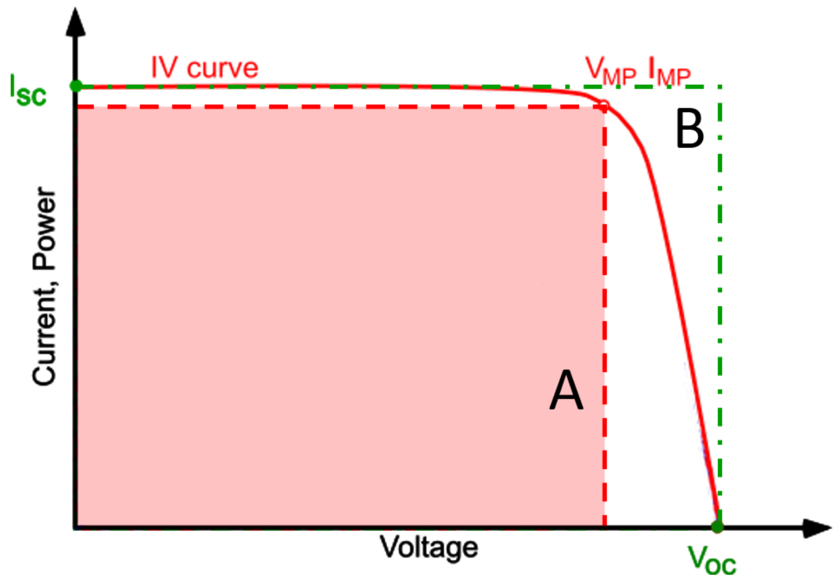


Figure 2.4: The current voltage (IV) curve of a solar cell with the rectangles *A* and *B* [5].

different combination of parameters. The curves are often gained using a model or experimental data. Then using the curves, the quality of each combination can be assessed and the best one can be found.

Finding an IV curve is a hard problem for some type of solar cells. In these cases, a fitting method can be used. In [6] and [26], device simulators are used in order to find IV curves and other graphs for respectively InGaN tandem solar cells and perovskite solar cells. Different values were tried and plotted. The best solar cells were obtained by analysing the influence of the variables using these plots. The curves can also be obtained by describing these curves using parameters. These parameters are then extracted using various methods. In [18], the parameters are extracted using a hybrid genetic algorithm. This means that there exist methods to obtain those IV curves, but this might be a time consuming and difficult process.

Optimizing a solar cell by studying the influences of parameters on the IV curve is a method which provides a lot of information about the behavior of the problem. For instance, it clearly shows the influence of the parameters. Therefore, investigating IV curves might lead to interesting insights about good solutions. A downside is that the method is not suitable for optimization with multiple variables. In order to apply this method, the influences of the variables are considered and optimized one by one. This is a quite greedy way and is likely to result in solutions which are not global optima.

However, IV curves do provide insight in how solar cells are studied. Two important factors for a high efficiency are the voltage and the current. It is hard to determine the efficiency of a solar cell with calculations. However, *GenPro4* gives the possibility to estimate the current produced by a solar cell. As a high current results in a high efficiency, the current should be optimized. This is the motivation to optimize the current of a tandem solar cell.

2.2.2. Use of optimization algorithms

Analysing the problem using graphs such as the IV curve often results in sub-optimal solutions. The use of optimization algorithms can provide a way to find better solutions and thus give a higher efficiency. Several ways in which algorithms are used to optimize the efficiency of a solar cell are discussed now.

Although the optimization problem of solar cells is a multidimensional problem, algorithms are not often used in order to perform full parameter space optimization. For instance in [7], the Newton Method is applied to optimize only the thickness of the perovskite layer of a tandem solar cell. Other parameters such as thicknesses of the silicon layer are not taken into account.

Genetic algorithms are rarely applied to optimize solar cells, but other optimization methods are slightly more frequently used. In [11], local optimization methods are used in order to find the best thicknesses of a tandem solar cell. The methods are executed from various starting points to enlarge the chance of finding global optima. One of the conclusions of this research is that the application of a textured surface reduces reflection losses. The found solution produces a current of 18.0 mA/cm^2.

Another optimization method is used in [16], where simulated annealing is implemented to optimize perovskite/silicon tandem solar cells using *GenPro4*. In the paper, the effect of inverting the order of the layers in both cells is investigated. The structure of these solar cells is visualised in Figure 2.5.

After repeated application of simulated annealing, all found solutions were nearly identical and resulted in a current of 17.6 mA/cm^2. The absorption of the materials was calculated and is shown in Figure 2.6.

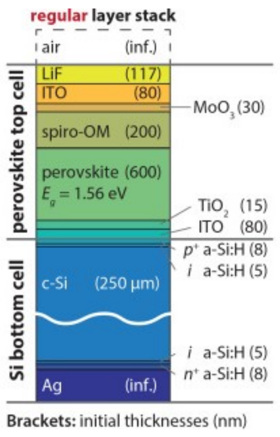


Figure 2.5: The structure of the solar cell [16]

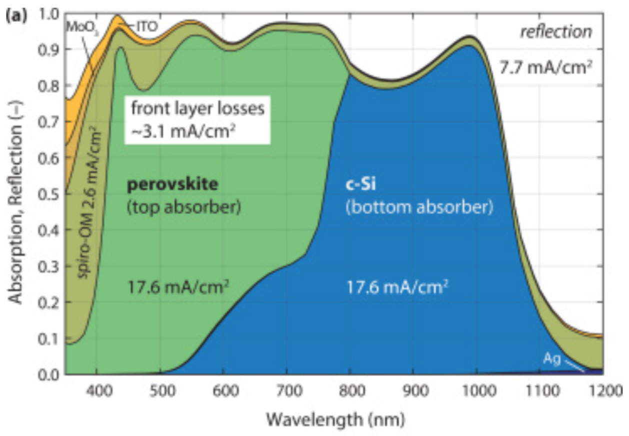


Figure 2.6: The absorption plotted against the wavelength of an optimized perovskite/silicon tandem solar cell [16].

In Figure 2.6, the absorption of a tandem solar cell is shown for each wavelength of the incoming light. It can be seen that the perovskite absorbs the most photons with smaller wavelengths and the silicon mainly absorbs the photons with larger wavelengths. However, when the wavelength is around 600 and 700 nm, the light is absorbed by both materials. This makes it possible to increase the light absorbed by one material at a cost of a decreased absorption of the other material, when the parameters are tuned. For wavelengths of around 400 nm and lower, it can be seen that other materials absorb the light, which is called parasitic absorption. For light with wavelengths of 1100 nm and more, the light is mainly reflected.

If there would be less parasitic losses for light with wavelengths that can be absorbed by the perovskite or silicon, more light could be absorbed. This can be controlled by the materials used and the thickness of the layers to optimize the efficiency. The found optimal values of the top contact layers were very close to the given lower boundaries. Therefore, it is likely that these layers should be as thin as possible to reduce these parasitic losses. This is something that might also occur when genetic algorithms are applied to Problem (1.1).

In [3], simulated annealing is also implemented to optimize different types of perovskite/silicon tandem solar cells in combination with *GenPro4*. The thicknesses of the layers of the perovskite cell are the variables. This is similar to Problem (1.1) when the discrete variables $x_7, ..., x_{10}$ are ignored. Only some of the used materials are different, which might cause the results to be slightly different. The found solar cell gives a current of 17.5 mA/cm^{-2}. Since the paper does not focus on the how this optimization method behaves for this type of problem, no further conclusions about the performance of algorithms can be drawn from this research.

Another relevant study for this thesis is [4], in which several different optimization methods are applied to the problem of optimizing solar cells. It discusses two different cases in which two types of solar cells should be optimized. Different types of optimization algorithms are applied to both problems.

The quantity that is optimized is the efficiency which is expressed in Equation (2.1). The general procedure used to optimize the efficiency is visualised in Figure 2.7. For the solar cells, different properties of the solar cells are variable or fixed. The fixed properties are denoted by a vector **a** and the variable properties are denoted by the vector **v**. For each entry of **v**, a lower bound and upper bound are given in respectively $\mathbf{v_L}$ and $\mathbf{v}_U$.

Each cycle of the process starts with a set of possible choices for **v** which adhere to the lower and upper bounds. It also has the parameters **a** which stay the same. Using both values of **v** and **a**, a physical model, SCAPS, is used to approximate the efficiency. In order to do this, it solves different equations such as a one dimensional Poisson equation and transport equations. This provides a method to find the efficiency η corresponding to all possible values of **v**.

Using this simulation method, different optimization methods can be used in order to optimize the problem. One of the implemented methods is a local gradient based method, which is a local method. Three global methods are used which are genetic algorithms, pattern search and particle swarm optimization. On top of this,

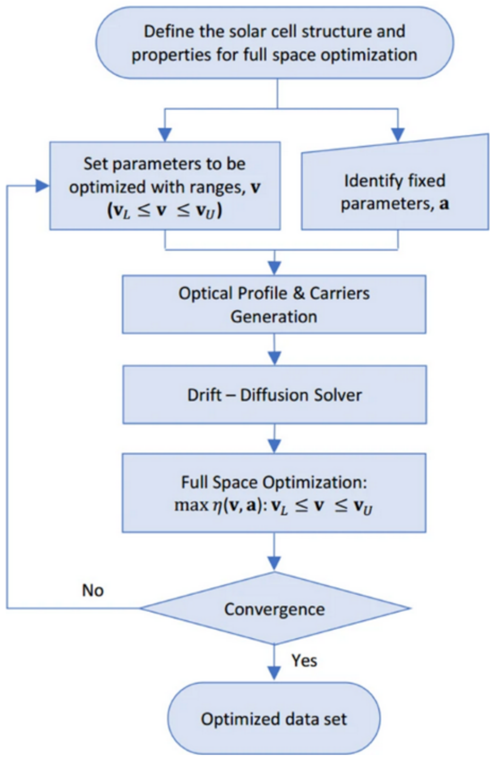

Define the solar cell structure and properties for full space optimization
Set parameters to be optimized with ranges, $\mathbf{v}$ ($\mathbf{v}_L \leq \mathbf{v} \leq \mathbf{v}_U$)
Identify fixed parameters, $\mathbf{a}$
Optical Profile & Carriers Generation
Drift – Diffusion Solver
Full Space Optimization: $\max \eta(\mathbf{v},\mathbf{a}) : \mathbf{v}_L \leq \mathbf{v} \leq \mathbf{v}_U$
No
Convergence
Yes
Optimized data set

Figure 2.7: A visualisation of the optimization procedure of solar cells used in [4].

the three global methods were combined with the local method to form three hybrid methods. The methods are implemented using the toolboxes in MATLAB.

One of the solar cells that is optimized using these methods is a perovskite solar cell. The variables are the thicknesses of different layers and properties of the HTM and ETM layers, such as the band gap, electron mobility and hole mobility. There are in total 23 parameters which are expressed in **v**. This solar cell is similar to the bottom cell of the tandem solar cell and the results of this study are therefore relevant for this thesis.

The gradient based method is found to be the fastest method. According to the article, this is caused by the smoothness and continuity of the physical models and the fact that the objective function is convex. Since the problem in this article and the problem considered in Problem (1.1) are quite similar, it might be the case that the problem of this thesis is well-behaving as well. A note should be made that the problem of this thesis includes discrete variables and not only continuous variables as in [4]. On top of that, the structure of the solar cell is more complicated and the objective function is equal to the minimal current generated by both absorbers. Thus, it is likely that the problem in (1.1) is less well-behaving than the problem in [4]. Therefore, different optimization methods might result in different values of the objective function.

Compared to most other methods, the genetic algorithm appears to need more function calls. It only needs less than the two hybrid algorithms of the gradient based method with the particle swarm method and the genetic algorithm. This gives that the genetic algorithm is not favorable for the optimization problem of this paper. However, the article does not state which genetic algorithm is used. Thus, trying other genetic algorithms could be worth the try. On top of that, the problem of this thesis is possibly not that well-behaved as in the article. Because genetic algorithms are global methods, they are likely to perform better in the problem of this thesis.

This research also shows is that the appearance of defects does not seem to change the number of function counts of the genetic algorithm, whereas it does influence the number of function counts for most other methods. Therefore, the genetic algorithm is likely to be robust for this problem. Since *GenPro4* uses Monte Carlo simulations to calculate the absorbtance [25], the resulting current might be subject to some small changes. Therefore, the robustness of the method is likely to have a positive effect on the result.

From [4], it can be concluded that the genetic algorithm has similar results to other optimization methods but at higher computational costs when optimizing a perovskite solar cell. However, the genetic algorithm is likely to perform relatively better in the problem of this thesis due to certain properties as the robustness and the fact that it is a global method.

This is also shown in [13], where several optimization algorithms are implemented for the optimization of a perovskite/silicon tandem solar cell. In this study, the exact same optimization problem is studied as Problem (1.1). This means the objective function is the current produced by the solar cell and is estimated using *GenPro4*. It is found that genetic algorithms find better solutions than the interior-reflexive Newton method and the simplex search methods. The highest found current is 20.94 mA/cm^2. However, the genetic algorithms also needed many function evaluations more than the other methods.

Although the genetic algorithm found the best solutions, only the influence of the parameter population size is studied. The population size describes size the group of solutions is considered by a genetic algorithm at the same time. It is found that genetic algorithms with larger population sizes find better solutions at the cost of a higher computation time. However, other settings of a genetic algorithm are not considered in this study. It might be the case that they improve the performance of the genetic algorithm.

From this study, it can be concluded that the genetic algorithms are likely to perform well for Problem (1.1). However, the long computation time could be a problem and should therefore be investigated too. A parameter that is important for both the quality of the solution and the computation time is the population size. But, it might also be beneficial to study the influence of other settings of the genetic algorithm.

To summarize, the optimization of solar cells is performed using different methods. The methods try to optimize the efficiency of the solar cells. As a higher current results in a higher efficiency, the current is sometimes optimized as well. Often, experimental data are used to find good properties of solar cells. Experimental data can also be used to build a model which describes the influence of parameters on the performance of a solar cell like could be done using IV curves. In some cases, optimization algorithms are used to optimize the performance of solar cells. Although several possible optimization methods could be applied to optimize solar cells, the optimization method chosen to use in this thesis is a genetic algorithm. Since genetic algorithms are rarely applied to optimize solar cells, this is a topic with knowledge gaps that could be filled by this thesis.

2.3. Genetic algorithms

In order to optimize the tandem solar cell, genetic algorithms are implemented in this thesis. There exist various versions of genetic algorithms which might perform differently for a specific problem. The optimization problem of this thesis, as described in Problem (1.1), has different types of variables. It includes both continuous and discrete variables. In combination with the complex objective function, which is the current calculated by *GenPro4*, this creates problem with a 10-dimensional parameter space and with, most likely, local optima. Different genetic algorithms are discussed in this section as an orientation for the actual implementation in later chapters. The main source of information is [17]. If different literature used, this is explicitly mentioned.

Genetic algorithms are a special type of evolutionary algorithms. Evolutionary algorithms are methods based on the mechanisms of evolution. This means that they are based on the application of survival of the fittest. Thus, there exists some type of competition in which the best performing individuals have descendants. In this way, the characteristics for the good performance are inherited to the next generation. Genetic algorithms are evolutionary algorithms with mechanisms based on genetics introduced in 1975 by J.H. Holland.

As mentioned before, different genetic algorithms exist. These methods all have the same general structure. The methods work similar to a species in nature, which evolves to reach a best fitness. In the context of a genetic algorithm, solutions are often called individuals and the objective function is called the fitness value. For this thesis, it means that an individual or solution represents the actual properties of a tandem solar cell. The objective function or fitness value in this thesis is the current produced by this solar cell. At each moment, the algorithm considers a set of individuals which is called a population. This population changes per iteration and those different populations are called generations. The process of a genetic algorithm is visualised in Figure 2.8.

The genetic algorithm starts with an initial population of μ individuals of which the fitness values are calculated. In each generation, the individuals generate λ individuals as offspring using certain operators and the corresponding settings. The individuals that generate offspring are called parents. Then, the best solutions in the parents and the offspring are selected to form the new generation. This is done in such a way that the individuals with better performance are more likely to be selected. Most of the times, the generation does not change drastically since parents are often kept. In this way, the population consists of better performing individuals on average for each new iteration. This continues until some stopping criterion is met.

While most genetic algorithms have the same general structure, they differ in the details. Genetic algorithms are determined by their operators. The two most important characteristics of genetic algorithms are the variation and the selection operators. A variation operator determines how an individual generates offspring to

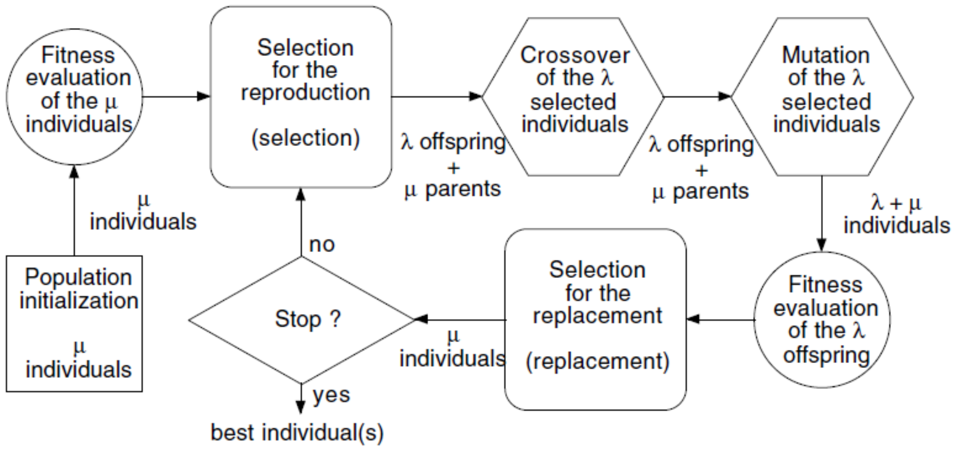

Fitness evaluation of the μ individuals
Selection for the reproduction (selection)
Crossover of the λ selected individuals
Mutation of the λ selected individuals
λ offspring + μ parents
λ offspring + μ parents
μ individuals
λ + μ individuals
no
Population initialization μ individuals
Stop ?
μ individuals
Selection for the replacement (replacement)
Fitness evaluation of the λ offspring
yes
best individual(s)

Figure 2.8: A visualisation of the process of a genetic algorithm [17].

increase the size of the population. A selection operator on the other hand selects the best performing individuals to reduce the size of the population or to determine which individuals will be used to produce offspring. Next to these operators, genetic algorithms also differ in how they represent solutions as genes and the size of the population.

2.3.1. Representation of solutions

The solutions in a genetic algorithm are seen as individuals. The way that the individuals are stored and used by the genetic algorithm differs and is called the representation. Different representations exist and are applicable to different problems.

In the first genetic algorithms, introduced by J.H. Holland in 1975, solutions were represented by binary words. That means that every solution corresponds to a vector with entries that are either 1 or 0. This way of representation is well-known and has been often used for years, for instance with the binary alphabet. There even exist theorems about the convergence of this type of algorithms, although they are limited to very specific cases.

For variation operators, it is important to have some sense of distance between different solutions. That means that if two solutions are similar, their representations must be similar too. In this way, a small change in the representation corresponds to a small change in the solution. This property is necessary to make sure that children are similar to the parents. In order to do this, several coding schemes exist, such as Gray coding [12]. Gray coding represents integer solutions in such a way that the representation of two consecutive numbers only differs in one bit in their binary word. In this way, similar solutions have representations that are similar too.

Although there exist ways to implement continuous variables as binary vectors, this representation method is not preferable. One reason for this is that the representations would be too long. This is illustrated by the following example. If there would be 100 variables with values in the interval $[-500, 500]$ with six decimals precision, the length of the representation would be 3000 [12]. This is clearly really long and is therefore not favorable since it would slow the other operators down too much. While the problem in this thesis does not have 100 variables, the length of the binary vectors would too large.

A more suitable representation for Problem (1.1) would be a real-coded representation [12]. In this representation method, the continuous variables $x_1, ..., x_6$ are simply stored in a vector $\mathbf{x}'$. Although this method is more distinct to the actual mechanisms in genetics, it directly provides a sense of distance between solutions. Two representations are close if their entries have values which are close in the sense of the variables. On top of this, the possibility arises to use other types of variation and selection operators. Another advantage is the fact that the representation is intuitive to interpret as the representations do not have to be decoded to be used for determining the fitness. Therefore, a real-coded representation is used in this thesis.

2.3.2. Selection operators

The selection operators are important operators of a genetic algorithm that select individuals from a population. There exist two types of selection operators of which the first determines which individuals are selected to apply the variation parameters on in order to create offspring. This process is often just called selection. The second selection parameter determines which individuals of the parents and children will become the new generation. Since this is the same as determining which individuals are left out, this process is often called replacement. This selection parameter makes sure that the population size remains constant.

Selection pressure

An important concept for selection operators is selection pressure. Selection operators are designed to favor solutions with better values of the objective functions over individuals with lower values. The selection pressure gives the extend to which the fitter individuals are favored by the selection operators. This concept can be described with several quantities.

When the variation operators only make copies of individuals, they are as simple as possible. If one individual is better than the others, this individual and its offspring are favored by the selection operator and the number of copies of this better individual in such population will grow. The takeover time is defined as the number of generations that it takes to fill the whole population with copies of this better individual. When the selection pressure is high, this will correspond to a lower takeover time.

The selection intensity is another quantity which can represent the selection pressure. To find this, a generation and the created offspring are necessary. Let $\bar{f}$ denote the average fitness of the the generation and its offspring before selection and let σ_f denote the corresponding standard deviation. $\bar{g}$ is defined as the average of the individuals in the part of the offspring that is selected by the selection parameters to be in the new generation. Then the selection intensity can be defined as in follows.

$$S = \frac{\bar{g} - \bar{f}}{\sigma_f} \tag{2.2}$$

This definition can also be used to define selection pressure for selection operators that determine the individuals to create offspring. In general, a higher selection intensity occurs when there is a higher selection pressure.

When good individuals are favored much more than others, the genetic algorithm will focus on the best values and select only those to create offspring. In this way, the genetic algorithm might not create offspring which are close to the global optima, since this might not be close to the best values in the current population. This means that a high selection pressure might cause the genetic algorithm to focus on local optima. When this happens, it is called premature convergence and it results in a lower found value of the objective function. This is not favorable.

On the other hand, a low selection pressure also has downsides. When the selection pressure is too low, the offspring will create more diversity in the new population than in the original population. If this happens too often, the genetic algorithm will not converge and find any local or global optimum. In general, a low selection pressure results in a low speed of convergence. Therefore, a low selection pressure causes the computation time of the genetic algorithm to be too long, which might also not be favorable.

A high selection pressure thus results in worse solutions and a low selection pressure in high computation time. This means that it is important to find a selection operator that results in a selection pressure which finds good solutions in reasonable time.

Proportional selection

A possible type of a selection operator is a proportional selection operator. This type of operator is only used to determine which individuals will become parents. These operators have the property that the expected number of offspring of an individual is proportional to the fitness value. Therefore, an individual with a high fitness value will produce more offspring. There exist adjusted selection operators which use proportional selection to select a new generation from a population and its offspring. These adjustments are mostly done by scaling and are discussed later in this section.

Let f_j be the fitness value of an individual j, let μ be the population size and let λ the number of individuals that should be created using the operator. Then, the expected number of selections of an individual i to reproduce itself is λ_i and can be defined as in the following expression.

$$\lambda_i = \frac{\lambda}{\sum_{j=1}^{\mu} f_j} f_i \tag{2.3}$$

A selection method that has this property, is called a proportional selection method. There exist different selection methods that have this property. The nowadays most common used method is stochastic universal sampling (SUS). With this method, the selection is still stochastic but the variance of the different possibilities is not extremely high.

In the SUS-method, a straight line segment is considered. The line is partitioned in as many smaller segments as there are individuals. The size of the segment of an individual i is proportional to λ_i. That means that individuals with a higher fitness have a larger segment. Then, equidistant points are drawn on the line. In order to find the place of the first point, a random drawing is performed using a uniform distribution on a specified small interval near the end of the line. The other points are drawn with equal distances between each other on the line and this is visualised in Figure 2.9 where these points are the pointers. The number of these points is the number of individuals to be selected to create offspring.

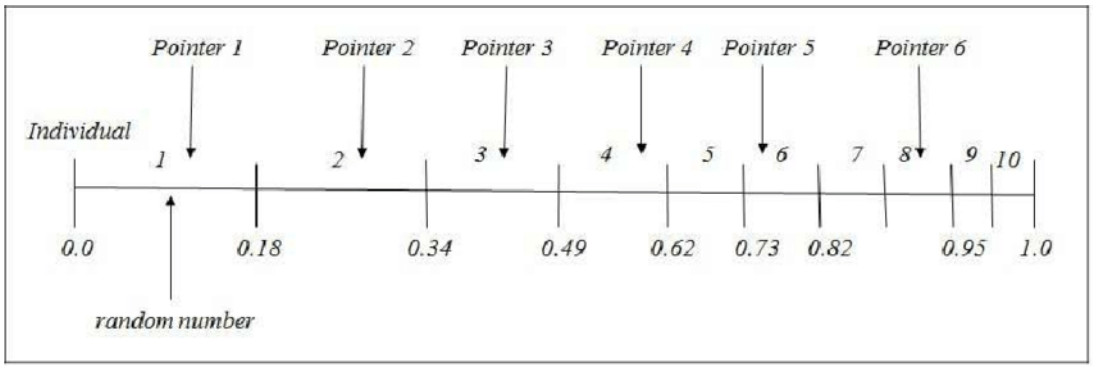


Figure 2.9: A visualisation of the SUS-method [14].

The number of desired offspring should be specified beforehand for this method. If the total number of offspring is larger than the size of the parent population, the SUS-method guarantees that individuals with fitness values above the average are selected. This is caused by the fact that in such situation the distance between the points is smaller than the size of the line segment corresponding to individuals with an above average fitness.

Proportional selection has a downside which can be shown using an example. For proportional selection, the selection pressure can be expressed in the following way. Consider a situation in which the μ individuals are selected as parents from a population with the same size μ. The fitness value of the best individual in a population is denoted by F and the average fitness value of the population is $\bar{f}$. Then, the selection pressure can be defined as in Equation (2.4). This quantity expresses the expected number of selections of the best performing individual.

$$p_s = \frac{F}{\bar{f}} \tag{2.4}$$

As μ parents are selected from μ individuals, an individual is selected once on average. In such situation, a value of $p_s \approx 1$ will indicate that each individual is selected nearly once. Therefore each individual is selected the the same number of times. This means that there is almost no selection pressure as the worst individuals have nearly the same probability of being selected as the best. If $p_s > 1$ on the other hand, the best individual is expected to be selected more than once. Since all individuals are selected once on average, this means that a worse solution has an expected number of selections that is lower than one. Therefore, the best individual is selected more times than the worse individuals and the selection pressure is higher.

This can be illustrated by an example. For this example, the objective function is $f(x) = e^{x^2}$, which should be maximized. The maximum occurs at $x = 0$ with a function value of $f(0) = 1$. The initial population is uniformly distributed on the interval $[-2,2]$ with the individual $x = 0$ included. This gives an average fitness of $\bar{f} \approx 0.441$, which results in a selection pressure of $p_s \approx 2.27$. That means that the best value is expected to be selected twice by the proportional selection method. This is reflected in Figure 2.10 by the fact that there exists a clear difference in the fitness value for different values of x.

When the genetic algorithm continues for some time, the individuals will be more concentrated around the optimum. To simulate this, the values are now taken in the interval $[-0.2,0.2]$. This results in a value of $\bar{f} \approx 0.986$ and a selection pressure of $p_s \approx 1.01$. In Figure 2.11, the fitness values are visualised. The fact that

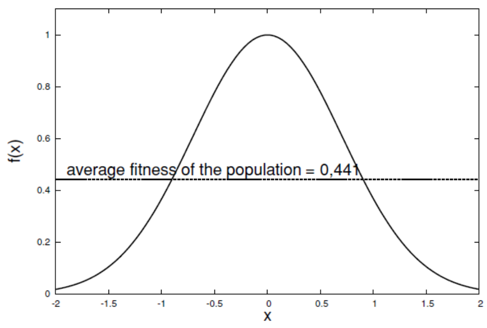


Figure 2.10: The fitness function of $f(x)$ on the interval $[-2,2]$ [17].

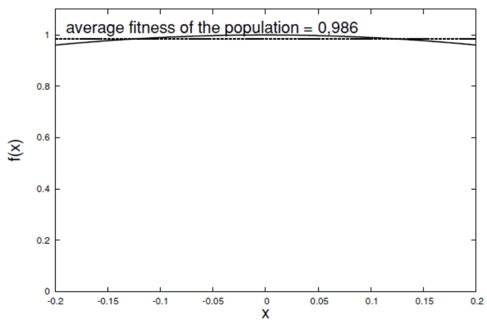


Figure 2.11: The fitness function of $f(x)$ on the interval $[-0.2,0.2]$ [17].

there is a small distinction in the fitness values gives that there is a small selection pressure. In this situation, because of the fact that the selection pressure is close to 1 and the fitness function is almost linear, there exists almost no selection pressure.

This example shows that the selection pressure of the proportional selection can decrease drastically over different generations of a genetic algorithm. This behavior can keep the algorithm from converging which is not desirable.

Scaled fitness functions

There exist methods to overcome this drawback, such as scaling the fitness function in order to regulate the selection pressure. This can be done in several ways, but all methods are based on changing the fitness function.

One method which can be used is linear scaling. This method calculates the fitness values f_i' by subtracting a constant a, which gives $f_i' = f_i - a$. This constant a should be chosen in such a way that the selection pressure does not become either too large or too small. It is also necessary to make sure that the function values do not flip sign since the fitness values are proportional to the size of the line segments and this size cannot be negative. In such cases, a negative value can be set to equal 0.

In a maximization problem, the value of a can be determined using the desired selection pressure using Equation (2.5) with a value $p_s > 1$.

$$a = \frac{p_s \bar{f} - \hat{f}}{p_s - 1} \tag{2.5}$$

This can also be applied to the example of optimizing $f(x) = e^{x^2}$. If a selection pressure equal to 2 is wanted, this gives a value of $a = 0.972$. Rescaling using the linear transformation gives the situation visualised in Figure 2.12, in which the differences in fitness value are more clear than in Figure 2.11.

Another way to redefine the fitness values is ranked scaling [24]. This means that the values of the individuals are ranked from the individual with the highest value to the individuals with the lowest value. The fittest individual gets rank 1 assigned. Then, the fitness value is scaled by applying $f_j' = \frac{1}{\sqrt{r}} f_j$ where r is the rank of individual j. In this way, individuals which are good are favored and if a lot of individuals have similar fitness, only one of them will be favored by the method.

Using ranked scaling changes the selection pressure. If all individuals have similar objective function values as is the case in Figure 2.11, the scaling gives the best solution a higher fitness value. Therefore, the selection pressure is changed to be higher. On the other hand, if there already is one individual that is significantly better, the fitness value of this individual even slightly increases compared to the other individuals. It can be said that the selection pressure is increased in general by this scaling method and it makes sure that the selection pressure does not vanish.

Ranked selection

The proportional selection operators with or without scaling are mostly designed to determine which individuals are selected to apply the variation parameters on. After the offspring is created using the variation parameters, the population size has grown too much and should be decreased. The covered operators based on proportional selection are not often used for this type of selection. However, there exists an operator which is often used for this action. This type of selection is ranked selection.

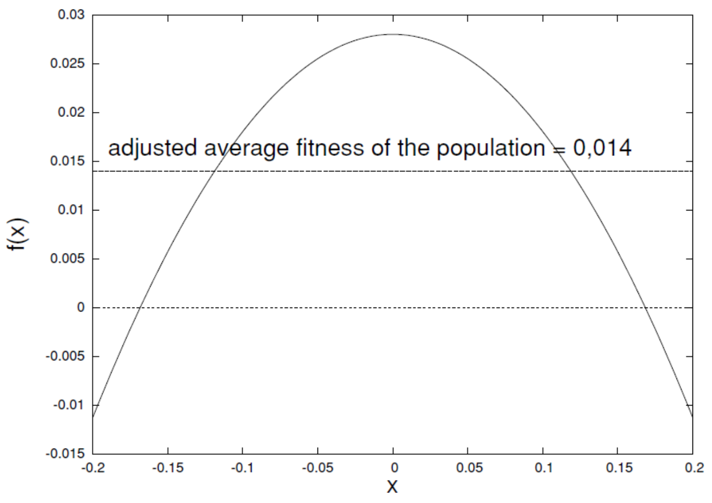


Figure 2.12: The scaled fitness function of $f(x)$ on the interval $[-0.2, 0.2]$ [17].

In ranked selection, the raw fitness values of all the individuals in the original population and the offspring are determined using the objective function. Subsequently, the individuals are ranked from the fittest individuals to the least fit individuals. After this is done, a new fitness value is assigned to each individual of rank r using Equation (2.6), where μ is the population size.

$$f'_r = \left(1 - \frac{r}{\mu}\right)^p \tag{2.6}$$

The value of p is dependent on the desired selection pressure. When the definition of selection pressure in Equation (2.4) is used, the relationship is $p_s = 1 + p$. On these new fitness values, proportional selection is then applied to choose a number of individuals.

An advantage of the ranked selection is that it does not need any understanding of the objective function. On top of that, it is really flexible. It works on both maximization and minimization problems and it does not need all values to be positive or negative. A downside is that small differences in the objective function are not taken into account by ranked selection. However, this stimulates the diversity in the population, which is not necessarily a bad thing. Therefore, ranked selection is a widely used selection method.

Elitism

The discussed selection operators which select a new generation out of an old generation and offspring, are all based on a stochastic process. This is due to the use of proportional selection. Despite the fact that most good solutions from the old generation will be selected, this makes it possible that the best individuals are not selected to be in the next generation. As the algorithm tries to find the best solutions, this is contradictory.

Elitism occurs when the best solutions of a generations are carried down to the new generation. This can happen automatically depending on the selection operators. However, it can also be forced to occur by placing the best solutions of the old generation automatically in the new generation. The selection operators can then be applied to the other individuals and the remaining spots in the new generation. In this way, the best solution in a genetic algorithm is guaranteed to be non-decreasing, which is often preferable.

Even if elitism is forced, most of the offspring is worse than the original population. Therefore, the generations often do not change that much. In general, elitism is desirable because of the fact that the best value is non-decreasing. But it decreases the variation in the population over time and should thus not be too strongly apparent.

In short, there exist several selection operators. When a selection operator is applied, it is important that the selection pressure is not too low or too high. This might cause the genetic algorithm to converge prematurely or converge too slowly, respectively. Next to these operators, elitism can also occur which guarantees that the best solutions are carried down to the next generation.

2.3.3. Variation operators

In the genetic algorithms, individuals should be able to create offspring. This is done using variation operators. Variation operators are based on how the genes of a parent are used to create offspring in nature. This happens in two ways which both induce a category of variation operators. These are called mutations and crossovers.

The first variation operator that is discussed is the mutation operator. When the genes of a parent are copied to create the genes of offspring, mutations occur, which are small changes in the DNA. Because of this, the new individual will have some different characteristics than its parents. Variation operators based on this are called mutation operators and they increase the diversity in the population.

Mutation operators often have the same structure. It namely has as input a parent which is a real valued vector. It then iterates over the vector entries. For each entry, a draw from a Bernoulli distribution is performed to determine whether or not a mutation is applied to this entry. The chance that a single entry is selected to be mutated is called the mutation rate and is in general around the values 0.01 and 0.1. If a mutation is applied often a stochastic method is used to change the existing entry a little bit. In principle, this stochastic method is based on adding a drawn value from a distribution with mean zero. In this way, mutations do not influence the average values and do not drive the solutions into one direction. The new vector gained by such a process is then a mutated vector of the parent which was given as input. Thus in short, a mutation in a genetic algorithm means that some values of an individual are slightly changed to create a new individual.

Crossover operators are the second type of variation operators. For almost every specie, the DNA of two parents is used to create the DNA of their offspring. This is done in such a way that the new individual is similar to both parents by taking part of the chromosomes of one parent and the other part of the other parent. During this process, so-called crossover occurs. This is shown in Figure 2.13. In short, a new chromosome is made by taking a part of the paternal DNA for the new chromosome and for the complementary part of the chromosome, the maternal DNA is used. Variation operators that create a new individual by combining two parents are called crossover operators.

Now, several possible mutation and crossover operators are discussed. It is important that those operators are applicable to the real valued representation and can be used for bounded variables.

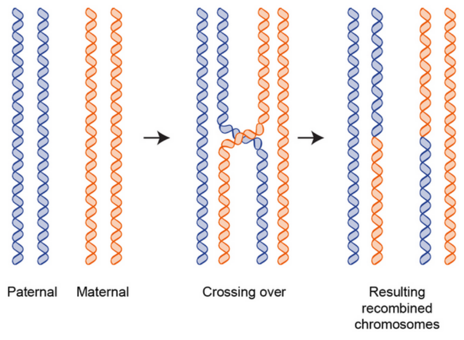


Figure 2.13: A visualisation of the crossover process of DNA [15].

Uniform mutations

One of the most accessible type of mutations are uniform mutations. For this, the entry that is chosen to be mutated must have the domain $\mathbb{R}$. The method then draws from a $[-a, a]$ uniformly distribution for some fixed a. The found value is added to the entry that must be mutated.

This can also be generalized to higher dimensions. But with this method in any dimension, it is often the case that the genetic algorithm is in a local optimum and a jump of more than a is needed. Then, the genetic algorithm will not converge to a global optimum. Therefore, it is in most cases better to use a distribution which has an unlimited support. In this case, there always is a chance that the method jumps out of the local maximum.

Gaussian mutations

The Gaussian distribution is such a distribution with unlimited support. It is also a widely used type of mutations. But it can only be applied to real valued and unbounded variables. When a Gaussian mutation is applied, a value drawn from a $\mathcal{N}(0,\sigma^2)$ distribution is added to the entry. For a Gaussian mutation, the most important thing is to find the right value of the σ^2 as it determines whether the mutation is likely to be large or not. Large mutations might be necessary to get out of local optima and the best value σ^2 is often not a fixed value and is hard to find.

In most cases, the best value of σ^2 even changes during the iterations of the genetic algorithm. When the algorithm is stuck in a local optimum, the variance within the population is often small. In this case, quite a large step might be necessary to get out of the local optimum. Therefore, a higher value of σ^2 is more likely to give a better result.

There is no guideline which is universally the best to adjust the variance of the Gaussian distribution. However, some guidelines do exist. For instance, the $\frac{1}{5}$-rule prescribes that σ^2 should be increased when less than $\frac{1}{5}$ of the mutations result in a positive change of the individual. In this case, the mutations are not large enough to jump out of the local optimum. On the other hand, when more than $\frac{1}{5}$ of the mutations result in a positive change, the value of σ^2 should be decreased. This rule of thumb can be implemented in different way. In 1981, Schwefel for instance formulated his version of this method.

Such adaptive Gaussian mutations are most successful and therefore most often used. A downside for the problem of this thesis is the fact that the real valued $x_1, ..., x_6$ variables are bounded and this type of mutation has an unlimited support. This means that the mutations could cause the individuals to cross the bounds which is a problem. However, this can be compensated by adjusting the method. If a value would jump out of the bounds, it can be set to a value which does not conflict with the boundary constraints. This can for instance be the boundary or a new draw from the Gaussian distribution. In the last case, it might be again needed to have a new draw until the value is within the bounds.

An adjusted Gaussian mutation might be a good type of mutation to apply to the real valued variables of the problem. For the discrete variables, something more simplistic can be applied because of the small number of possible values. For instance, a mutation in the variables x_8, x_9, x_{10} can only be done by switching to the other possible value, since these variables only have two possible values. If a mutation in variable x_7 would be needed, one of the options would be in the existing vector. A drawing form a Bernoulli distribution with chance 0.5 could be used to determine to which of the other two options should be changed.

Crossovers
Just as with the mutations, different types of crossovers exist, single point and two point crossovers for instance.

Let $\mathbf{v} = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$ and $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \dots & w_n \end{bmatrix}$ be vectors in $\mathbb{R}^n$ which represent two individuals of a population which are selected for the crossover operator. Then a random integer k is drawn from the set $\{1, 2, ..., n\}$. This integer determines where the crossover takes place. The first k entries of $\mathbf{v}$ and the last $n - k$ entries of $\mathbf{w}$ are taken as the entries of a vector which represents their child. This vector will thus be $\begin{bmatrix} v_1 & v_2 & \dots & v_k & w_{k+1} & \dots & w_n \end{bmatrix}$. This new vector is an individual which is similar to both $\mathbf{v}$ and $\mathbf{w}$.

For two point crossover, two integers k and m are drawn randomly from the set $\{1, 2, ..., n\}$. Now the crossover is applied twice instead of one time. This means that the first k entries of $\mathbf{v}$ will be used. Then entries $k + 1$ up to m are used of $\mathbf{w}$ and entries $m + 1$ up to n from $\mathbf{v}$ are used. This results in the new individual
$\begin{bmatrix} v_1 & v_2 & \dots & v_k & w_{k+1} & \dots & w_m & v_{m+1} & \dots & v_n \end{bmatrix}$ which is also similar to both $\mathbf{v}$ and $\mathbf{w}$.

Another commonly used, but more complex type of crossovers exists and is called uniform or scattered crossover. For this type of crossover, a random binary vector is used. This binary vector is obtained by drawing n values from a Bernoulli distribution with parameter $p = 0.5$, which are then used as the entries of the vector. This gives a vector $\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}$ in $\{0, 1\}^n$. This vector determines which entry is taken from $\mathbf{v}$ and which entry is taken from $\mathbf{w}$. This means that for each i'th entry of the child, the entry v_i is taken if $b_i = 0$ and the entry w_i is taken if $b_i = 1$. In this way, the entries from $\mathbf{v}$ and $\mathbf{w}$ are randomly mixed to form a new individual which is similar to both the parents $\mathbf{v}$ and $\mathbf{w}$.

It can be concluded that there exist several mutation and crossover operators which can be applied to the optimization problem of this thesis. Some methods are even known to perform better such as the adapting Gaussian mutations and would be preferable to use in a genetic algorithm.

2.3.4. Other parameters
Next to the already discussed possible choices, a genetic algorithm also has other options that should be specified. The most important parameters are the population size and the stopping criterion.

During the iterations of a genetic algorithm, the variation operators create new individuals and the selection operators reduce the number of individuals in order to make the new generation. In general, this is done in such a way that the size of the population is constant for all iterations. The size of the population is an important parameter of the genetic algorithm.

A larger population size means that there are more individuals in a population and in general also more offspring for each generation. This means that more function evaluations must be done. Since a function evaluation takes around several seconds, a large population size could easily mean that the computation time of the genetic algorithm is too long. If a genetic algorithm would be able to find the best value in less time, this would obviously be favorable. Therefore, a smaller population size could be preferable.

However, decreasing the population size also means that less different points in the solution space can be visited. Consequently, a smaller population size means that a smaller part of the solution space is investigated by the genetic algorithm. This means that it is more likely that the genetic algorithm does not investigate the space round the global optimum when the population size would be smaller. Therefore, the approximation of the optimal value found by the genetic algorithm can be worse when the population size is decreased. This makes it necessary to investigate the influence of the population size on the performance of the genetic algorithm in order to find the right trade off between good results and a reasonable computation time.

The genetic algorithm iterates over different generations. However, the genetic algorithm needs an initial population to start the process. This should be done in such a way that the individuals cover a part of the solution space that is as large as possible. In that way, the genetic algorithm considers a larger part of the domain.

After the genetic algorithm is started, the genetic algorithm iterates over the different generations. This iterative process should stop when the values have converged enough and therefore a stopping criterion is needed. A possible stopping criterion is to implement a maximum number of generations or a maximum number of function evaluations [22]. These numbers can be chosen when there is knowledge about the problem. Due to lack of such detailed knowledge about the problem, this number cannot be determined yet. Therefore, other stopping criterion are needed.

Another stopping criterion is to stop the algorithm if no better new value has been found for a number of generations. If no better value is found, no improvement can apparently be made by the genetic algorithm and the algorithm has converged enough to the optimal value. It is good to note that the values of the objective function simulated by *GenPro4* is subject to changes. Therefore, it should be predefined what differences in function value are seen as significant.

To summarize, genetic algorithms have other variable options which can be tuned next to the different choices of the variation and selection operators. These other parameters should be set to good values in order to find the best performing genetic algorithms.

2.3.5. Matlab genetic algorithm function

The software of *GenPro4* is made in the computer language Matlab. Therefore, it is a natural choice to use Matlab to implement the genetic algorithms for this thesis. The Global Optimization Toolbox in Matlab provides a function 'ga' which applies a genetic algorithm to an optimization problem [22]. The use of a standard Matlab function for this optimization problem is preferable since it lowers the threshold to use the outcomes of this thesis for future research.

This function has predefined default settings, but it also offers many possibilities to change the algorithm [20]. This can be done by for instance changing a default setting to an existing non-default option. On top of this, the 'ga' function also offers the possibility to implement custom made functions for the operators. In this way, the genetic algorithm can almost be changed entirely but it also provides structure.

The 'ga' function offers the possibility to use both continuous and discrete variables in the genetic algorithm. It also gives the possibility to integrate the bounds on the variables in the problem. On top of that, a function can be written to specify for each generation what should be done with the data of that generation. It also gives the possibility to specify stopping criteria and the population size as well as the possibility to plot the data during the process.

However, the function 'ga' has a limited number of options for several settings for a bounded mixed-integer optimization problem. It is for instance not possible to change the selection or variation operators for such optimization problems, not even a custom written operator. As the problem of this thesis is a bounded mixed-integer problem, the options for this thesis are also limited. But for many settings, it is still possible to change their values. Even for the variation operators, there are parameters that can change the behavior of the operators. Although the options to change the genetic algorithm of 'ga' are somewhat limited for this optimization problem, there still exist enough settings that can be changed. Therefore, this function is still used.

In short, Matlab provides a predefined genetic algorithm function which has a lot of flexibility but also provides the necessary structure. Since an implementation of a genetic algorithm using a standard function is also more accessible for later use, it is desirable to use the predefined Matlab function 'ga'.

3 Objective function

The objective function is the function for which an optimal value should be found. In Problem (1.1), this is the current generated by the solar cell corresponding to certain values of the variables. This current should be maximized and is estimated by *GenPro4* using a simulation. The estimation is likely to make an unknown error. The simulation performed by *GenPro4* uses ray-tracing based on the Monte-Carlo method. Due to the estimation error of the simulation, repeating the simulation with identical input does not give identical output. This introduces random errors into the objective function. The current produced by *GenPro4* can therefore be seen an estimation of the real current produced by the tandem solar cell.

The estimation is better when the errors are small. In this situation, the results of two nearly identical solutions is also likely to be identical, since the actual current is relatively smooth and the errors are small. This gives a smooth objective function, which is favorable as genetic algorithms work better when applied to smooth function. As the estimations are variable, they have a distribution. If there is less spread within the estimates, the objective function is smoother. Therefore, it is important to make sure the standard deviation of the distribution is small. In this chapter, the distribution of the estimates are studied. This is then used to determine what settings of *GenPro4* could best be used to increase the accuracy of the estimates in order to find the best objective function for the genetic algorithms that are implemented and discussed in the next chapters.

The *GenPro4* software is implemented to simulate the solar cell corresponding to this optimization problem. To implement the simulation, the input variables should be chosen in the right way. In this problem, all layers, even those with variable thicknesses, are thinner than the coherence length of the incident light, which is 1 μm, and are therefore implemented as coatings. The continuous variables, which correspond to the thicknesses of the layers of the first interface, are implemented as continuous variables with values between 0.040 nm and 0.500 nm. For the variables x_7 and x_8, each material is matched to an integer number, which represents the material. The variable x_9 determines the presence of an anti-reflexive coating of MgF_2. For this variable, a layer of coating is added to the model. When a solar cell should have this coating, the layer has the material MgF_2. Otherwise, this layer is made out of air. Since the coating is at the top of the solar cell, this gives the right result. If variable x_{10} implies that the solar cell has a pyramid structure, the texture ‘pyramids_20um’ is included in the structure of the solar cell on top of the second interface.

3.1. Distribution of the estimates

Using this representation, the software of *GenPro4* can be used to determine the current of a tandem solar cell. This can be seen as a function where the properties $x_1, ..., x_{10}$ are the input variables and the estimated current is the output variable. As discussed, the output is obtained by a simulation and is therefore an estimation. *GenPro4* has default options and to investigate the behavior of the objective function estimated by *GenPro4* using the default options, the distribution and accuracy are analyzed. This is done by repeating the simulation for the same input variables numerous times. To make sure this result is applicable to the optimization problem, a realistic combination of input variables is used, also referred to as an individual in the context of genetic algorithms, which is displayed in Table 3.1. The simulation with this individual is ran 2000 times to study the spread of the estimations and its distribution.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
0.1065	0.0573	0.2015	0.3933	0.3476	0.0693	2	1	1	1

Table 3.1: The input variables used for the simulation to study the accuracy and variation of the estimations.

The results are shown in a histogram in Figure 3.1 and a QQ-plot in Figure 3.2. The corresponding mean is 20.9404 mA/cm^2 and the standard deviation is estimated to be 0.0295 mA/cm^2. The histogram shows that resulting values are similar to the curve of the normal distribution that is shown in red. The resulting values however differs from the normal distribution for some values. For instance, the peaks around 20.925 mA/cm^2 and 20.94 mA/cm^2 are too large and the frequencies between these values and between 20.925 mA/cm^2 and 20.94 mA/cm^2 and the pair 20.95 mA/cm^2 and 20.965 mA/cm^2 are lower. However, these differences are not that large and are not necessarily significant.

Figure 3.2 supports that these differences might not be significant. In this plot, the data is normally distributed if the data is similar to the red dashed line. The data shows a relation which seems to be similar to the red line. Only at the sides, it differs slightly from the line. The way they differ from the line, means that the data might have light tails. To determine whether the described differences from the normal distribution are significant, a Kolmogorov-Smirnov test is applied to the data. This tests the null hypothesis that the data is normally distributed against the alternative hypothesis that the data is not normally distributed. The test resulted in a p-value of 0.7193 and thus, the null hypothesis cannot be rejected. This means that the normal distribution with a mean of 20.9404 mA/cm^2 and standard deviation of 0.0295 mA/cm^2 is a good fit for the data.

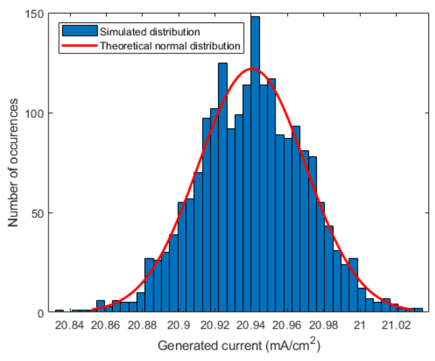


Figure 3.1: A histogram of the simulated data with the probability density function of the normal distribution.

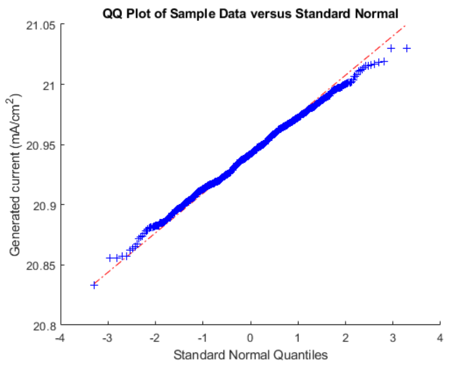


Figure 3.2: A quantile-quantile plot of the simulated data versus the normal distribution.

The distribution of the estimates gives insight in the accuracy of the estimates. A normal distribution has the property that 32% of the data deviates more than one standard deviation from the mean and 5% of the data more than two standard deviations. The mean of the distribution can be seen as the real value of the current as it is likely to be a good estimate for it due to the large sample size. This would mean that 32% of the estimates differs more than 0.0295 mA/cm^2 from the real value. This implies that the estimates are not accurate for the second decimal. On top of that, 5% of the estimates has an error larger than 0.059 mA/cm^2. For the data to be accurate in the first decimal, the errors should be smaller than 0.05, which is not the case. As this size of error occurs less frequently, most estimates are accurate in the first decimal. However, with a probability of 0.05 the data is not accurate up to one decimal. Since thousands of estimates will be performed, such error is likely to occur. In short, the estimates are not accurate up to two decimals and with a smaller possibility not accurate in the first decimal. The errors in the estimates are large and this has several downsides. Therefore, a higher accuracy, and thus a lower standard deviation, might be needed.

If the estimates of the objective function are not accurate, it has several downsides. The first downside is that a found optimum is most likely not significantly different than other values which are clearly lower. This also gives that a found large value can be found because the estimate of the value is larger than the other values only due to the randomness rather than it being an optimum. This might cause the genetic algorithm to focus on the wrong local optima or solutions that are not even optima. On top of that, when the estimates are not accurate, the errors create a lot of noise in the implemented objective function. Noise makes the objective function less smooth and genetic algorithms work better on smooth functions. Therefore, the genetic algorithm is likely to perform worse when the estimates are not accurate. For this optimization problem, it means that the accuracy of the objective function should be improved.

3.2. Improving accuracy

GenPro4 provides the opportunity to change settings in order to make the simulation more accurate. However, with more accurate settings, the simulation takes more time. To determine whether the variation can be decreased in reasonable computing time, several options to increase the accuracy of *GenPro4* are used to simulate this same value 100 times. These can be used to investigate the relationship between the accuracy and the settings.

GenPro4 uses a number of angular intervals, which has a default value of 18. To study its influence, the values 18, 24 and 30 are taken into account. A higher number of angular intervals should mean a higher accuracy. On the other hand, it also means that the scatter matrices which *GenPro4* uses to calculate the current become larger and with it the computing time increases.

GenPro4 recreates the situation in the solar cell by simulating the behavior of light rays. The higher the number of rays that are simulated, the higher the accuracy. On the other hand, the simulation will take a longer computing time since the trajectories of more rays must be simulated. The standard number of rays is 100 and the values of 50, 100, 500 and 1000 are used to study. All possible combinations of the number of rays and the number of angular intervals are used one hundred times to simulate the same solar cell. This means that for all 12 combinations of options, the simulation is repeated 100 times and the results and computing time are determined.

GenPro4 uses numerical approximations of an integral in order to estimate the current. A smaller step size of this numerical approximation should give a better result. However, a smaller step size means that for more different wavelengths, the corresponding data must be simulated. A smaller step size thus results in higher computational costs. Therefore, the accuracy of the simulations is investigated for different step sizes. The default step size is 0.020 μm and the step sizes which are considered are 0.020 μm, 0.010 μm, 0.005 μm and 0.0025 μm. These are all simulated 100 times to study the accuracy. This is done after the best combinations of the number of solar rays simulated and the number of angular intervals are determined. It is not possible to take all possible combinations into account for all number of rays, number of angular intervals and step sizes, because otherwise 48 different combinations should be considered and this would result in a too long computation time for this thesis.

The accuracy of the estimated current by *GenPro4* is first studied for different numbers of angular intervals and different numbers of solar rays used in the simulation. This is done by studying the relation between the settings and the estimate of the standard deviation. Since the estimates have a normal distribution, the standard deviation provides information on the accuracy of the method. The results are shown in Figure 3.3. It can be seen that there is a strong correlation between the number of rays and the standard deviation. A higher number of rays results in a strong reduction of the variance, and therewith, the noise in the objective function. Therefore, using a higher number of rays would be better.

On the other hand, the number of angular intervals does not seem to have a strong relationship with the standard deviation. The relationship seems to be dependent on the number of rays. For the situation with 50 solar rays, the angular interval does influence the standard deviation significantly as the standard deviation is clearly higher when there are 18 angular intervals. In a situation with 1000 solar rays, the number of angular intervals does also seem to influence the standard deviation. A higher number of angular intervals results in a lower standard deviation in this case. The standard deviation of this combination is 0.0081 mA/cm^2. The best combination seems to be the one with 1000 solar rays and 30 angular intervals. These numbers are the maximal settings of *GenPro4* for these options. Therefore, these would be favorable.

However, these settings increase the computation time. The duration of the computations for each combination is shown in Figure 3.4. The computation times are all found using the simulation of the same device. This device runs on windows 8.1 and has an Intel i-4460 CPU with a clock speed of 3.20 GHz and 4 cores. The RAM-memory on this device is 8,00 GB. The relation of the number of rays and the computation time which is shown in Figure 3.4 is clearly linear. From the graph, it can also be concluded that a higher number of rays or a higher number of angular intervals results in a higher computation time. The combination with 1000 solar rays and 30 angular intervals even results in a computation time of 252 seconds. This would mean that it would take 70 hours for 1000 function evaluations. As the genetic algorithms need several thousands of function evaluations, it could take more than a week for the genetic algorithms to run. This computing time is too long.

To reduce the computing time, *GenPro4* has an option to save the scatter matrices and reuse them. In this case, the scatter matrices do not have to be estimated every function call by simulating the solar rays, which makes the function call faster. The tandem solar cell in this optimization problem consists of two interfaces. All variables, except x_{10}, concern properties of the first interface. The second interface of this solar cell, which

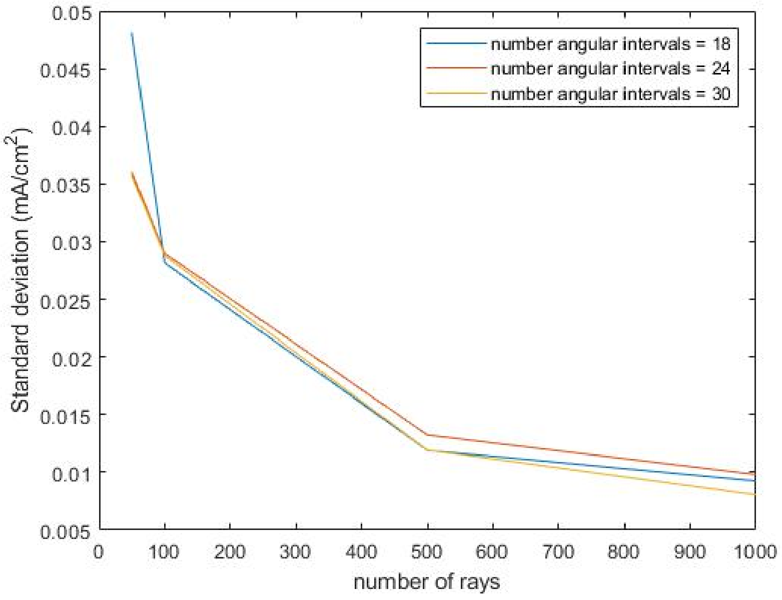


Figure 3.3: The standard deviation of the estimates for different numbers of rays and angular intervals.

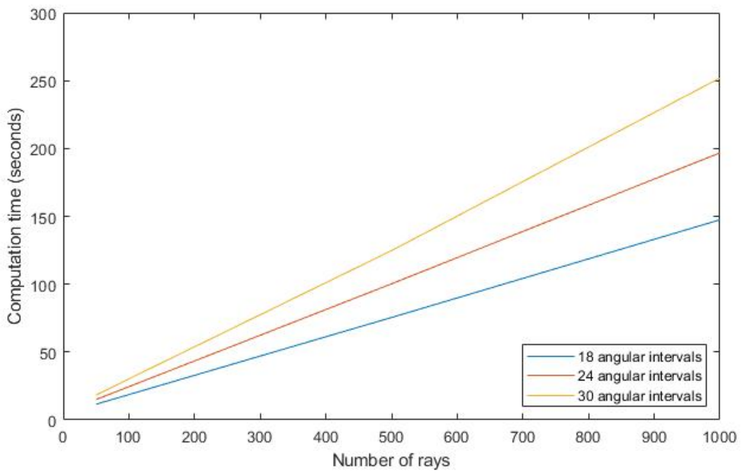


Figure 3.4: The computation time of the objective function for different numbers of rays and angular intervals.

is made of silicon, is only affected by the variable x_{10}, which determines the usage of a pyramid structure. For the cases with and without pyramids, the corresponding scatter matrices can thus be stored and reused to skip a part of the simulations. This reduces the computation time of the objective function from 252 seconds to 11.5 seconds, which is a better computation time for the use of a genetic algorithm. A downside of using this to implement a genetic algorithm is the fact that an error in determining the scatter matrices the first time contributes to an error in the estimations for all other individuals in the genetic algorithm. Therefore, it is necessary to make sure the accuracy of the first simulation is as high as possible.

On the other hand, reusing the scatter matrices also has advantages. For instance, the same simulation is used at every function call and therefore, there is no spread anymore in the estimates. This means that the standard deviation is zero. As a result, the objective function is more smooth and the genetic algorithm is likely to perform better. On top of that, the simulation only has to be performed once. Therefore, it is beneficial to use very accurate settings for this simulation. This results in a long computation time of mainly the first function call. At this small cost, it gives a more accurate objective function.

Another way to increase the accuracy of the objective function is to decrease the step size used for numerical integration. The default step size is 0.020 nm and other step sizes that were analysed are 0.010 μm and 0.005

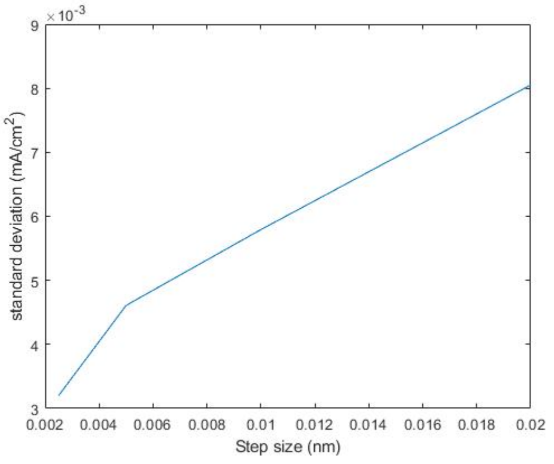


Figure 3.5: The standard deviation of the objective function for different step sizes of the numerical integration.

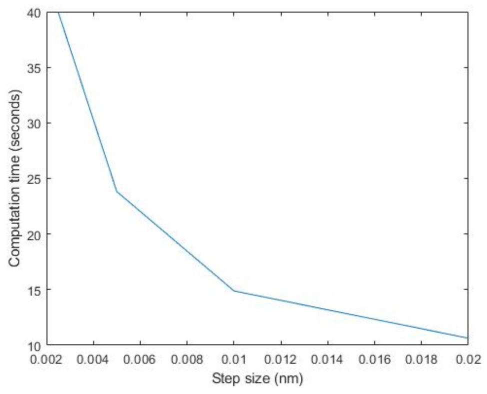


Figure 3.6: The computation time of the objective function which reuses scatter matrices for different step sizes of the numerical integration.

μm. The results are shown in Figure 3.5 and show a linear correlation. The smallest standard deviation is 0.0046 mA/cm^2 which is significantly smaller than the standard deviation of the default settings which was 0.0481 mA/cm^2. This smallest standard deviation is achieved with a step size of 0.005 μm. This correlation shows that a smaller step size results in a smaller standard deviation, and thus, a smaller estimation error. Therefore, it would be preferable to use a step size as small as possible, possibly even smaller than 0.005 μm.

The computation times for the different step sizes are shown in Figure 3.6, where the step size of 0.0025 is also included. The figure shows that a smaller step size results in a higher computation time in a non-linear way. The function evaluations of the implementation with a step size of 0.020 μm took 20 seconds to evaluate. Therefore, the 24 seconds corresponding to a step size of 0.005 μm is likely to work too, since the genetic algorithm could converge faster because the objective function is more smooth. A step size of 0.0025 μm results in a computation time of 40 seconds. This is roughly twice the computation time and is likely to cause the genetic algorithm to take too long. Therefore, a step size of 0.005 μm is used.

At this point, the settings of *GenPro4* are found that give an accuracy of 0.01 mA/cm^2 within a reasonable time. However, the choice of these settings are based on the accuracy of the simulation of one individual solution. This is the solution shown in Table 3.1. The error could be different for other solutions and in this case, the found settings of the objective function might not be good settings. In order to investigate this, other simulations are performed for three different solutions. Each solution is simulated 100 times with the default settings of *GenPro4* and with 30 angular intervals, 1000 rays and a step size of 0.005 μm.

Name	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
y_1	0.1065	0.0573	0.2015	0.3933	0.3476	0.0693	2	1	1	1
y_2	0.2592	0.3951	0.1578	0.2598	0.4825	0.0925	1	1	0	0
y_3	0.1036	0.0536	0.4041	0.3811	0.3235	0.0740	2	1	1	1
y_4	0.1011	0.0552	0.0404	0.3816	0.2742	0.0737	2	1	1	1

Table 3.2: The solutions used for the simulation to study the accuracy and variation of the estimations.

The three solutions that are chosen are solutions that could be chosen by the genetic algorithm and are chosen to be diverse. The solution in Table 3.1 is labeled as y_1 and the other solutions are shown in Table 3.2. After the objective function is determined 100 times for each solution, the standard deviation of the values are calculated. These resulting values are shown in Table 3.3 for both the default settings as the settings with a high accuracy.

Table 3.3 shows that the standard deviation of the objective function of y_2 is equal to 0. This means that there is no variation within the objective function. This can be explained by the fact that y_2 does not include a pyramid structure in the tandem solar cell. *GenPro4* only has to simulate the incoming rays if a pyramid structure is used. This means that for y_2, no simulations are executed in order to estimate the current. The current is simply determined by calculations and does therefore not vary.

Solution	Standard deviation	
	Default settings	High accuracy settings
y_1	0.0303	0.0044
y_2	0	0
y_3	0.0021	0.0024
y_4	0.0040	0.0002

Table 3.3: The standard deviation in mA/cm^2 of the objective function for different solutions and different accuracy settings.

In Table 3.3, it can also be seen that the standard deviation of the more accurate settings is clearly lower than the standard deviation of the default settings of *GenPro4* for solution y_4. The standard deviation of solution y_3 is in both cases really small. This means that the more accuracy settings give that the highest standard deviation is lower. Since the accuracy is not constant for all solutions, it might be possible that there exist solutions for which the accuracy is less than for y_1. This would make the objective function less accurate. However, it is impossible to study the variation of the objective function on the whole domain as the variable space has 10 dimensions. On top of that, if the whole domain would be studied using the objective function, it would no longer be necessary to apply a genetic algorithm since the objective function for the whole domain would be known. Although the accuracy can vary slightly over the domain, the accuracy seems to be more than 0.01 mA/cm^2 for most solutions. Therefore, it is concluded that the objective function with 30 angular intervals, 1000 rays and a step size of 0.005 μm is more accurate than the default settings and it most likely provides an accuracy of 0.01 mA/cm^2.

3.3. Conclusion

Several options of *GenPro4* were considered in this section, such as the number of angular intervals, the number of rays and the step size of the numerical integration. These settings provide the possibility to increase the accuracy of the estimates produced by *GenPro4*, which are used as the objective function for the genetic algorithms. The best settings within reasonable computation time are with 30 angular intervals, 1000 rays and a step size of 0.005 μm. These result in errors with a standard deviation of 0.0046 mA/cm^2. Since the errors are normally distributed, this means that in 68% of the estimates, this is smaller than 0.005 mA/cm^2. Therefore, the estimates are for 68% of the simulations accurate up to two decimals. For 5% of the estimates, the error is larger than 0.0092 mA/cm^2. This means that, in 5% of the cases, the estimate has an error of roughly 0.01 mA/cm^2. The function values also seem to have this accuracy for other points in the domain.

The settings with higher accuracy can be realized within reasonable time by reusing the scatter matrices. These scatter matrices are estimated using a simulation that takes a lot of time, which is saved in this way. As the simulation is not performed anymore, the standard deviation in the objective function has completely vanished. This results in a more smooth objective function. As genetic algorithm is more likely to perform better on a smooth objective function, this probably results in better solutions of the genetic algorithm.

If a low computation time is really important, it is possible that different settings are more beneficial. In such case, it is still best to reuse the scatter matrices as they give a shorter computation time and a smooth objective function. If the scatter matrices are reused, the simulation is only performed once. As the number of rays specifies how many rays are used in this simulation, it does not influence the computation time of a function call that reuses the scatter matrices. This means that this setting could best be set to the most accurate setting. But, all other accuracy settings do influence the computation time. Therefore, it might be useful to use less accurate settings as this decreases the computation time.

As reusing the scatter matrices gives a smooth objective function and a shorter computation time, this is used in this thesis in the implementation of the objective function that is used for the genetic algorithms. For this thesis, the more accurate settings are chosen. This is done because is desirable that the results of different genetic algorithms should be compared. If the accuracy of the objective function is lower, it is more likely that obtained differences in the results are insignificant. Therefore, the objective function is implemented with 30 angular intervals, 1000 rays and a step size of 0.005 μmin such a way that it reuses scatter matrices.

4 Genetic algorithms

With the knowledge obtained in Chapter 2 and the fine-tuned objective function from Chapter 3, a genetic algorithm can be implemented. As discussed, the 'ga' function provided by the Global Optimization Toolbox in Matlab [22] is a genetic algorithm which can be applied to Problem (1.1). In this thesis, genetic algorithms are implemented using this function 'ga'. In line with the conclusions from Chapter 3, the objective function is implemented using *GenPro4*, with accuracy settings of 30 angular intervals, 1000 rays and a step size for numerical integration of 0.005 μm. In this chapter, it is explained how the best implementation of the genetic algorithm for this optimization problem is determined in terms of performance. The performance is measured in both the quality of the solutions and the computation time of the genetic algorithms. The function 'ga' provides several default settings, which give the algorithm certain properties. The source [20] is used in this section for information on the settings and their properties. For each setting, these default options are discussed and it is predicted whether they are suitable for this optimization problem. Some of the default settings of the function 'ga' might not be well-suited for Problem (1.1). These settings are the population size, the stopping criterion and the use of parallel computing in the genetic algorithm. For these settings, it is discussed how they influence the performance of the genetic algorithms in order to find the best settings for the genetic algorithm. The code used for the implementations can be found at `https://github.com/KoenvanArem/BEP-Genetic-Algorithms-on-Solar-Cells`.

It is important to note that the function 'ga' only works for minimization problems. The maximization problem in Problem (1.1) can be changed to an equivalent minimization problem by multiplying the objective function with -1. The results of the genetic algorithm are again multiplied with -1 to find solutions for the maximization problem. All results in Chapter 5 are converted back to the original maximization problem.

4.1. Representation

An important characteristic of the genetic algorithm in 'ga' is the representation that it uses. The optimization problem in Problem (1.1) has ten variables, of which six are continuous and four are discrete. The 'ga' function considers a variable to be continuous by default. The variables first six variables, $x_1, ..., x_6$ are continuous and have values in the interval $[0.040, 0.500]$ when measured in μm. Therefore, these continuous variables are given as input for the function with the corresponding bounds. The representation used for these variables is a real valued representation.

Next to continuous variables, the 'ga' function can handle variables with integer values. The variable x_7 has the possible values spiro-OMeTAD, FTO and TiO_2, which are not integers but quantitative values. Therefore, each of these values is matched to exactly one integer. In this case, it is done by matching spiro-OMeTAD, FTO and TiO_2 to the numbers 0, 1 and 2, respectively. In this way, the discrete variables with quantitative values can be written as variables with integer values. The variable x_8 also has quantitative values. The possible values PTAA and NiO are matched to 0 and 1, respectively. In this way, the variable x_8 can also be written as a variable with integer values. The variables x_9 and x_{10} already had the possible values 0 and 1, and are therefore already integer valued. The variables x_7, x_8, x_9 and x_{10} are implemented by assigning them as integer variables in the function 'ga'. In order to do this, the possible values of the integer variables are also specified as function input for 'ga' in order to make sure that the variables only attain the possible values. In this way, a integer valued representation for the discrete variables x_7 up to x_{10} is used.

$$
\begin{array}{llllr}
\max\limits_{x_1,\ldots,x_{10}} & f(x_1,\ldots,x_{10}) & & & (4.1a)\\
\text{subject to:} & x_i \in \mathbb{R} & \forall i = 1,2,3,4,5,6 & \text{(Thickness variables)} & (4.1b)\\
& 0.040 \leq x_i \leq 0.500 & \forall i = 1,2,3,4,5,6 & \text{(Boundaries thickness)} & (4.1c)\\
& x_7 \in \{0,1,2\} & i = 7 & \text{(ETM)} & (4.1d)\\
& x_8 \in \{0,1\} & i = 8 & \text{(HTM)} & (4.1e)\\
& x_9 \in \{0,1\} & i = 9 & \text{(With/without AR-coating)} & (4.1f)\\
& x_{10} \in \{0,1\} & i = 10 & \text{(With/without pyramide structure)} & (4.1g)
\end{array}
$$

This representation is an intuitive choice for this optimization problem. The new formulation of the optimization problem is given in Problem (4.1). Since both integer valued and real valued variables exist, the problem now clearly is a mixed-integer optimization problem. In this specific case, the optimization problem also has lower and upper . Specifically for mixed-integer optimization problems with bounds, the Matlab function ‘ga’ is limited in possible settings [21]. It can still be applied, but some operators cannot be changed to non-default or custom made operators. This is the case for the crossover and mutation operators. This does not mean that other mutation and crossover operators cannot be used for this type of optimization problem, but the function ‘ga’ does not provide the possibility to do so. If another operator would be desired, a genetic algorithm should be build from scratch. Due to the limited time for this thesis, this is not preferable. On top of that, the function ‘ga’ still provides numerous settings to adapt the genetic algorithm for this specific optimization problem. Therefore, this function is still used to solve this optimization problem.

An advantage of this representation is that the solutions do not have to be decoded to determine the fitness value. This is the case when, for instance, a binary representation is used. Due to the relatively high computation time for the objective function, the time lost for coding and decoding would be relatively small. Therefore, this loss of time would not be a problem. However, as pointed out in Subsection 2.3.1, a binary representation would need large binary vectors, which is not favorable due to high storage costs. Therefore, a binary representation is probably a worse representation than the mixed real and integer valued representation as described above. In general, it can thus be expected that this representation with both real and integer valued variables is most suited for this particular optimization problem.

4.2. Selection operators

Another main element of the genetic algorithm is the selection operator. As discussed in Section 2.3, there exist several selection operators. The choice of selection operators is limited because of the restrictions for mixed-integer problems with bounds implemented in the Matlab function ‘ga’. Therefore, it is only possible to use the default selection operator. However, it is possible to change the scaling of the fitness function.

The selection is, by default, performed using proportional selection with scaling of the fitness function. This is implemented using the stochastic uniform sampling method combined with ranked scaling. Both the SUS-method and the ranked scaling are discussed in Subsection 2.3.2. As explained in this subsection, the fact that the operator uses a scaling of the fitness function, makes sure the selection pressure does not vanish.

In Subsection 2.3.2, it is also explained that there exist two types of selection operators. The first type selects from each generation with size μ the individuals that become parents. With the variation operators, these parents produce λ children. The population then consists of μ individuals from the previous generation and λ children. The second type of selection operators are now used to select μ individuals from these $\mu + \lambda$ individuals. These selected individuals make up the new generation.

The genetic algorithm that is implemented in the function ‘ga’ only uses one selection operator. Before the selection operator is applied, the best performing 5% of the old generation is carried down to the new generation. This means that some elitism occurs. The population size is kept constant, and therefore, the other 95% is determined by the selection operators. The operator then selects which individuals in become parents. The number of parents is approximately 95% of the population size. The variation operators create children from the parents and these children are placed in the new generation. In this way, the generation consists for 5% of elite children and for 95% of children from parents selected by the selection operator. As all parents are changed by the variation operators, this means that no parents are carried down to the next generation, except for the elite individuals. With this structure, the function ‘ga’ does not use the second type of selection operators, but only the first type which selects the individuals to become parents.

This use of the selection operator is different than the selection operators that were discussed in Subsection 2.3.2. However, the structure cannot be changed. Moreover, due to the limitations of the function 'ga' for this type of problems, it is not possible to use anything else than stochastic uniform sampling. Therefore, this SUS-method is used in the implementations. On the other hand, the method for the scaling of the fitness function can be changed to other scaling methods. For the implementation, the ranked scaling method is used. In Chapter 5, it will be evaluated whether this scaling method seems to work.

4.3. Variation operators

In Section 2.3, it was discussed that variation operators are one of the main elements of a genetic algorithm. Due to the limitations for mixed-integer problems, the type of variation operators cannot be changed to a non-default operators. For this implementation of the genetic algorithm, the default operators for both crossover and mutation operators are used.

As discussed in Section 4.2, the selection operators determine which individuals are parents of the individuals in the new generation. These parents are given as input to the variation operators. These parents are either given to the crossover operator or the mutation operator. The crossover rate determines the number of individuals that are given as input to the crossover operator. This is by default 0.8, which means that crossover is applied to 80% of the parents. The other 20% of the parents are given as input for the mutation operator.

The default crossover operator is a scattered crossover operator, which uses a random binary vector to create the children and was treated in Section 2.3. This operator has two parents as input and creates two children as output. The binary random vector determines which entry of the parents goes to which child. This is discussed in more detail in Subsection 2.3.3. This operator often performs well and is therefore an often applied crossover operator. Most probably, this operator will perform well on this particular problem and does not need to be changed. On top of that, the function cannot be changed due to the limitations of the 'ga' functions for bounded mixed-integer problems. Therefore, the scattered crossover operator is used in the implementation.

The default mutation operator of 'ga' uses adaptive Gaussian mutations, which are discussed in Subsection 2.3.3. However, the adaptation rules for this operator are slightly different and it is based on two parameters, the shrink and scale parameters. The scale parameter determines the standard deviation of the Gaussian mutations for the first generation. A higher scale parameter results in larger mutations. The shrink parameter is used to control the rate with which the standard deviation shrinks during the run of the genetic algorithm. This is done using the following expression in which $\sigma_{i,k}$ expresses the standard deviation of the mutations in x_i of the k'th generation and 'generations' is the maximum number of generations.

$$\sigma_{i,k} = \sigma_{i,k-1}\left(1 - \text{shrink}\frac{k}{\text{generations}}\right)$$

This adaptive type of Gaussian mutations shrinks the size of the mutations over time to make sure that the algorithm converges, independent of the selection pressure caused by the selection operator. The default values of the scale and shrink parameters are both 1. This would mean that a standard deviation of 1 μm would be used for the mutations. This gives problems with the lower and upper bounds as they are 0.040 μm and 0.500 μm, respectively. In order to deal with this, the function 'ga' scales it down at the beginning of the algorithm to the shrink parameter multiplied with the difference of the upper and lower bound. Both scale and shrink parameters can be changed to non-default values, despite the limitations of the function 'ga' for bounded mixed-integer problems. For the implementations, the default functions are used. Using the results, it is evaluated using the other results whether the default values are right for this particular optimization problem.

This type of mutation also gives rise to other problems. The mutation does not work for integer variables and even for the continuous variables, the Gaussian mutations can result in individuals which do not adhere to the bounds. To handle the integer variables, a modified method is used which is introduced in [10]. It uses a specific truncation method in order to make sure the variables have integer values. In order to make the values of the continuous variables adhere to the lower and upper bounds, a method presented in [9] is used. This method introduces a penalty in the fitness function when the values are on the wrong side of the boundaries. In this way, the individuals will obey to the given boundaries. However, the method used to deal with the given bounds might cause the solutions to stay away from the boundaries. This might result in worse solutions if the global maximum is near the boundaries.

This type of adaptive Gaussian mutations is specifically designed for problems with the structure of Problem (4.1). Therefore, it is likely to perform well. The scale and shrink parameters provide the opportunity to influence the standard deviation of the mutations. If the size of the mutations does not change in the right way over time, these values could be used to improve the performance. As there are more influential parameters in the genetic algorithm for this specific problem, these are investigated. The relation between the mutation parameters and the performance is not studied due to the time available for this thesis.

4.4. Creation method

The genetic algorithm is able to make a new generation out of the previous generation using the selection and variation operators. However, the first generation should be created in another way. This must be done in such a way that the individuals are spread out as much as possible over domain. In this way, the genetic algorithm can take a larger part of the domain into account.

The default creation method for the function ‘ga’ is uniform creation. This method creates every individual by drawing from distributions for each variable of the individual. This means in this optimization problem that for the first individual, a drawing is performed formed for all variables $x_1, ..., x_{10}$ independently. The drawing that is performed is from a uniform distribution. The variables $x_1, ..., x_6$ are continuous variables with values in $[0.040, 0.500]$. To find the value for each one of the variables, a draw from the uniform distribution with the interval $[0.040, 0.500]$ is performed. This means that a random value is drawn from the interval $[0.040, 0.500]$ where each value has the same chance of being drawn. As the variables $x_7, ..., x_{10}$ are implemented as integer valued variables, the uniform draw works differently. In this case, a draw is performed where all integer values have the same chance of being picked.

Using these drawings from the uniform distributions, the starting generation will have variety within the population. However, it is possible that parts of the domain contain more individuals than others. This might cause the genetic algorithm to explore certain parts of the domain better than other in the search for optima. This problem is less likely to occur for larger population sizes and might therefore be not such a large problem. On top of that, the function ‘ga’ can only be implemented using the default creation method because the optimization problem of this thesis contains integer valued variables. However, it is possible to specify the starting generation beforehand, in which case the function ‘ga’ does not use a creation method. To specify a starting population in a good way, insight of the optimization problem would be helpful. However, a main reason to apply genetic algorithms to this optimization problem is that no detailed knowledge about the solar cell is needed. This makes the optimization process of tandem solar cells more accessible. Therefore, the genetic algorithm is implemented using the uniform creation method.

4.5. Population size

In this optimization problem, the most limiting factor is the computation time of the objective function as it takes 21.8 seconds per function evaluation. As discussed in Subsection 2.3.4, the choice for the population size greatly influences the number of function evaluations.

For instance, a genetic algorithm with a large population size is able to search for optimal values for a larger part of the domain than one with a small population size. This is due to the fact that within a larger population, there is more variety within the population. Therefore, each generation will cover a larger part of the domain when the population size is large. Because of this, a larger population size makes it more likely that the genetic algorithm finds a global maximum. However, a smaller population size increases the probability that the genetic algorithm converges to a local optimum.

On the other hand, a genetic algorithm with a large population size needs to evaluate the objective function of more individuals for each generation. Therefore, the computation time will be higher for a large population size, which is an important downside.

The default setting for this optimization problem is a population size of 100. This would mean that it takes around 35 minutes to compute the objective function values for all individuals in one generation. As the genetic algorithm at least has dozens of generations, the total computing time is long. On the other hand, the problem might contain many local optima. As genetic algorithms with smaller population sizes are more likely to converge to local optima, larger population sizes might also be beneficial. Therefore, the default value of the population size is not considered as fixed and the influence of the population size on the results should be studied.

To study the influence of the population size, the genetic algorithm is implemented for different population size. For each population size, the generations, the corresponding fitness values and the number of function evaluations are stored. The generations and the corresponding fitness values can be used to asses the quality of the results of the genetic algorithm. Storing the individuals of the generations also provides the possibility to see whether the genetic algorithm has converged already. On top of that, the individuals can also be used to study what variables are most important to create a high current as these variables are more likely to converge the fastest. To investigate the dependency of the computation time on the population size, the number of function evaluations performed by the genetic algorithm are kept track of. The computing time is not measured in seconds, minutes or hours, as this differs on the different computing devices which were used to run the genetic algorithms on. Instead, the number of function evaluations are used to measure the computation time.

The genetic algorithm is implemented for the population sizes 25, 50, 75, 100, 125 and 150. The largest population size is 150 as it takes too much computing time compared to the time available for this thesis to run the genetic algorithm for larger population sizes. 25 is the smallest implemented population size as smaller population sizes are likely converge to a local optimum instead of a global optimum.

In short, the genetic algorithm is implemented for population sizes ranging from 25 to 150 in order to study the influence of the population size on the quality of the results and the computing time. The individuals and fitness scores of each generation are stored to quantify the quality of the results and the number of function evaluations are stored in order to quantify the computing time.

4.6. Stopping criterion

Another characteristic of a genetic algorithms that should be studied is the stopping criterion. For each generation, the stopping criterion is tested and if the criterion holds true, the algorithm is terminated. The goal of the stopping criterion is to stop the genetic algorithm when it has converged in order to prevent it from iterating over unnecessary generations. On the other hand, the stopping criterion should not terminate the algorithm before a good solution is found. When no progress is made for a long time, the genetic algorithm is not likely to find any better solutions. In this situation, it is said that the algorithm has converged. A genetic algorithm that already has converged, should terminate because the new generations are not likely to find better solutions. On the other hand, the stopping criterion should not terminate the genetic algorithm before this situation occurs. This is due to the fact that if the algorithm stops before it has converged, the results are probably less good than when it would have continued. If the computational costs of the objective function would be low, it would be better to take a stopping criterion in such a way that the algorithm stops when it has converged for sure. This is not possible in this optimization problem due to the high computational costs. In short, it can be said that the choice of the stopping criterion is a choice in the trade-off between a reasonable computing time and good results, which is an important issue for this optimization problem due to the high computational costs of the objective function.

The ‘ga’ function has a default stopping criterion that keeps track of the best solutions of all generations and has two parameters, the max stall generations, M, and the function tolerance, ϵ. The algorithm terminates when the average increase of the best solutions over the last M generations is less than ϵ. This stopping criterion is based on the fact that the genetic algorithm has likely converged when there is no significant increase for a high number of generations. In Matlab, M is by default equal to 50 and ϵ is by default 10^{-6}. In Chapter 3, it was found that the objective function in most cases is accurate up to 2 decimals. Therefore, the value of 10^{-6} for ϵ is probably too small and the new value of ϵ is set to be 0.01. These values are used in the implementations to study the influence of the population size on the results and computing time of the genetic algorithm.

In Section 5.2, it is described that the stopping criterion with $\epsilon = 0.01$ and $M = 50$ does not perform well for most population sizes. In most cases, it stops the algorithm too late. Due to the importance of the stopping criterion for the quality of the results and the computing time, the relation between performance of the stopping criterion and the parameters are studied.

To study this, the information about the generations of the different runs is used to asses the quality of the stopping criteria. The results of the genetic algorithm for the different population sizes are used for this purpose. It is first determined for each population size at what point the algorithm should have been terminated. These points are called stopping points. In order to provide a choice in the trade-off between computing time and quality, three types of stopping points are determined for each run of the implementation. The first stopping point is chosen to be implemented for a genetic algorithm that has a low computation time and a lower quality of the results. The second stopping point is meant to have a good solution within reasonable stopping time and the third stopping point should have the best solution as possible at the cost of a higher computation time. The three types of stopping points are defined using a preferred accuracy, δ. This accuracy means that the solution of the stopping points differ less than δ from the highest found value. In practice, a stopping point of a higher accuracy terminates the genetic algorithm later. The first stopping point has an accuracy of 0.01 mA/cm^2, the second has an accuracy of 0.005 mA/cm^2 and the third has an accuracy of 0.001 mA/cm^2.

Using these accuracies, the three types of stopping points can be determined for each run of the genetic algorithm. In order to do this, the highest function value for every generation is found for each run. This information is used to find when the highest values of the whole run occurred for the first time. Denote this highest function value of the run as f_i and the highest function value of generation s by f_s. Denote the number of generations considered by the genetic algorithm with population size i as N_i. The three different stopping points are then defined as $S_{i,j} = \min\{s \in \{1,...,N_i\} : f_h - f_s < \delta_j\}$, where i denotes the population size of the genetic algorithm of which the generations are used and j means the jth type of stopping point. The point $S_{i,j}$ can be seen as the first generation that has the same highest function value as the best function value of all generations with respect to the given accuracy. If such a stopping point is within the last 4 generations, the genetic algorithm has most probably not converged yet with respect to the desired accuracy. In such a case, the point $S_{i,j}$ is set to ∞. The stopping point $S_{i,j}$ can be seen as the desired stopping point for a stopping criterion given a certain accuracy δ_j.

After the stopping points are determined, it is checked for different combinations of M and ϵ when the algorithm would have terminated if they would have been used as stopping criterion. The number of this generation is denoted by $T_i(M,\epsilon)$, where i denotes the population size. If there is no generation that satisfies the stopping criterion with M and ϵ, the genetic algorithm is not stopped by this stopping criterion. In such situations, $T_i(M,\epsilon)$ is given the value of ∞. In this way, $T_i(M,\epsilon)$ can be seen as the point at which the stopping criterion with M and ϵ stops.

A hypothetical situation of the points N_i, $S_{i,j}$ and $T_i(M,\epsilon)$ is shown in Figure 4.1. In this situation, the genetic algorithm has run for $N_i = 51$ generations. The desired stopping point for the given accuracy is at $S_{i,j} = 42$ and the stopping criterion with M and ϵ stops the algorithm at generation $T_i(M,\epsilon) = 30$. Since $T_i(M,\epsilon) < S_{i,j}$, the genetic algorithm stops before it should, which is not a favorable situation.

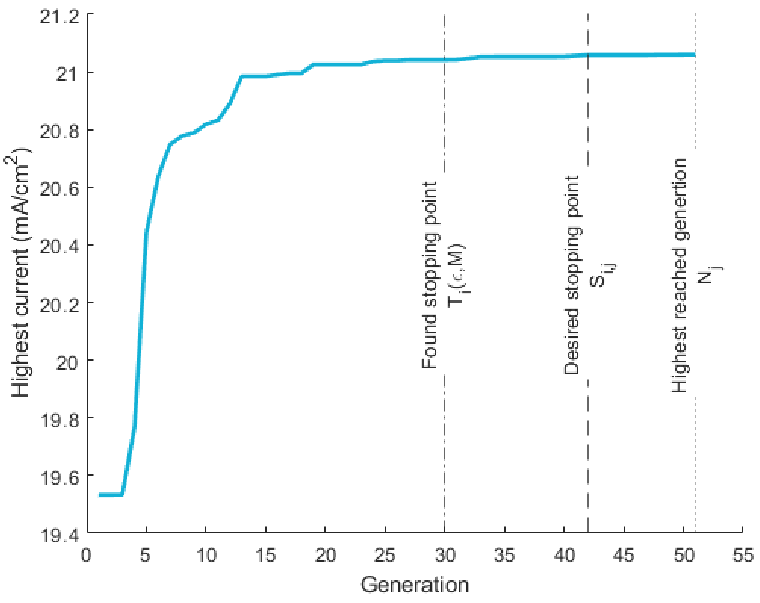


Figure 4.1: The highest found current for a hypothetical run of the genetic algorithm. The values of the highest reached generation N_i, the desired stopping point $S_{i,j}$ and the found stopping point $T_i(M,\epsilon)$ for a stopping criterion with M and ϵ are also visualized.

Using the values of N_i, $S_{i,j}$ and $T_i(M,\epsilon)$, the quality of each stopping criterion can be assessed. In order to do this, a penalty value $P_{i,j}(M,\epsilon)$ is created for the stopping criterion with M and ϵ, where $j \in \{1,2,3\}$ denotes the type of stopping point and i denotes the population size. This value quantifies the performance of the stopping criterion for each combination of M and ϵ. The penalty value is build up using different cases that can occur.

In the first possible case, the stopping criterion stops at the desired stopping point. This happens if $T_i(M,\epsilon) = S_{i,j}$ and is the ideal situation. Therefore, no penalty is assigned to this situation. If the stopping point is ∞ and the stopping criterion does not stop the genetic algorithm, the stopping criterion also works as it should. As both $T_i(M,\epsilon)$ and $S_{i,j}$ are ∞ in this case, this situation actually is also covered by the statement $T_i(M,\epsilon) = S_{i,j}$. Therefore, no penalty is assigned to such a situation.

In all other combinations of $T_i(M,\epsilon)$ and $S_{i,j}$, the stopping criterion does not terminate at the desired stopping point. This means that the stopping criterion makes an error, and therefore, a penalty is assigned. There exist two main possibilities if $T_i(M,\epsilon) \neq S_{i,j}$. The first stopping criterion is $T_i(M,\epsilon) > S_{i,j}$, in which the stopping criterion stops too late. In this case, the genetic algorithms iterates over too many generations. This increases the computation time. As every generation gives a longer computation time, a penalty of 1 is assigned for every generation that the genetic algorithm considers more than it should. If $T_i(M,\epsilon) < \infty$, this means that $T_i(M,\epsilon) - S_{i,j}$ more generations are considered than desired. This results in a penalty of $T_i(M,\epsilon) - S_{i,j}$.

If $T_i(M,\epsilon) > S_{i,j}$ and $T_i(M,\epsilon) = \infty$, the stopping criterion did not terminate the genetic algorithm after N_i generations, whereas it has already passed the desired stopping point. In this situation, it is unknown at what point the stopping criterion would have stopped the genetic algorithm as there is no information about the generations higher than N_i. The stopping criterion takes M generations into account to determine whether the increase in the highest function value is significant. This means that for a higher value of M, the stopping criterion most likely stops M generations after the last significant increase. The real stopping point is therefore estimated as $N_i + M$, which means that the genetic algorithm is estimated to consider $N_i + M - S_{i,j}$ generations too much. Therefore, a penalty of $N_i + M - S_{i,j}$ is given in this situation.

Another possibility is that $T_i(M,\epsilon) < S_{i,j}$. This means that the stopping criterion stops the genetic algorithm too early. This results in a highest found function value that is significantly worse than the best found result. This means that the stopping criterion does not result in the desired accuracy. As this is worse than computing a generation too much, this type of error has a higher penalty. This is done by giving a penalty of size 2 for each generation that the stopping criterion stops too early. The situation in which $T_i(M,\epsilon) < S_{i,j} < \infty$ holds is visualized in Figure 4.1. In this case, the algorithm runs $S_{i,j} - T_i(M,\epsilon)$ generations too few. For each generation that the stopping criterion stops too early, a penalty of 2 is given, this means that a penalty of $2(S_{i,j} - T_i(M,\epsilon))$ is given.

If $T_i(M,\epsilon) < S_{i,j} = \infty$, the genetic algorithm should not have stopped as it did not converge yet. However, the stopping criterion did stop the genetic algorithm. Similarly to the case $S_{i,j} < T_i(M,\epsilon) = \infty$, the number of generations after N_i is estimated by M. This means that it is estimated that the genetic algorithm should have run $N_i + M$ generations, whereas the stopping criterion stopped the genetic algorithm after $T_i(M,\epsilon)$ generations. As stopping too early results in a higher penalty, this situation results in a penalty of $2(N_i + M - T_i(M,\epsilon))$. This was the last possible case, and therefore, the penalty is defined for every possible case. The following expression summarizes the given penalty for all different cases.

$$P_{i,j}(M,\epsilon) = \begin{cases} 0, & \text{if } T_i(M,\epsilon) = S_{i,j} \\ T_i(M,\epsilon) - S_{i,j}, & \text{if } T_i(M,\epsilon) > S_{i,j} \text{ and } T_i(M,\epsilon) < \infty \\ N_i + M - S_{i,j}, & \text{if } T_i(M,\epsilon) > S_{i,j} \text{ and } T_i(M,\epsilon) = \infty \\ 2(S_{i,j} - T_i(M,\epsilon)), & \text{if } T_i(M,\epsilon) < S_{i,j} \text{ and } S_{i,j} < \infty \\ 2(N_i + M - T_i(M,\epsilon)), & \text{if } T_i(M,\epsilon) < S_{i,j} \text{ and } S_{i,j} = \infty \end{cases} \tag{4.2}$$

This penalty value $P_{i,j}(M,\epsilon)$ is determined for each possible combination of M and ϵ, each population size j and for each type of stopping point corresponding to the accuracy δ_j. Then, the final penalty of a combination M and ϵ for each type of stopping point is determined by $\sum_i P_{i,j}(M,\epsilon)$, which sums the penalty for all population sizes. The stopping criterion with the combination of M and ϵ that performs the best for a certain type of desired accuracy is the stopping criterion that has the lowest final penalty.

The values of M that are considered the values ranging from 1 to 20 as the value of 50 seemed too large. The considered values of ϵ are $10^{-1}, 10^{-2}, ..., 10^{-8}$ mA/cm^2, since these are values in magnitudes of the same order as the accuracy of the objective function. The final penalty is determined for all combinations of these values for M and ϵ for each desired type of stopping point.

In the case that a run of a genetic algorithm clearly terminated before it converged, it can be performed again with another stopping criterion that terminates it at a later generation in order to obtain more information about the convergence. By comparing the desired stopping points to the hypothetical stopping points of the stopping criteria using the defined penalty, it is determined what values of M and ϵ result in the best stopping criterion for the desired choice in the trade-off of the genetic algorithm between a short computing time and a good solution.

The default stopping criterion is based on the differences in the objective function as they will be small if the differences within the population are small. However, there exist other stopping criteria which are based on the differences within the individuals. When the individuals are insignificantly different, the genetic algorithm has converged for sure and should be terminated. The layers in a solar cell can be made with a precision of around

0.001 μm, and thus, smaller differences are deemed insignificant. With the results from the implementations for the different population sizes, it can also be assessed whether or not such stopping criterion would be a significant improvement.

To summarize, the stopping criterion is studied using the results of the implementations for the different population sizes. For each run, it is established when the algorithm should have terminated. Then it is determined when the genetic algorithm would have terminated for all combinations of M, with the values 5, 10, 25 and 50, and ϵ, with the values 0.1, 0.01, 0.001 and 0.0001. By comparing the desired point of termination with the point of termination for each boundary condition, the best values of M and ϵ can be selected. On top of that, it is studied whether a stopping criterion based on differences between the individuals would be preferable.

4.7. Parallel computing

As the computation time of the objective function is 24 seconds, the genetic algorithm needs the most time for computing the values of the objective function for all individuals in each generation. As discussed in Chapter 3, the computation time of the objective function can be reduced at cost of the accuracy of the objective function. However, an accurate objective function has several benefits as it creates as the results of the genetic algorithm are more accurate and the objective function is smoother, which improves the performance of a genetic algorithm. Therefore, it is not desirable to reduce the computation time of the objective function at cost of the accuracy. This gives that the most limiting factor in the application of genetic algorithms to Problem (4.1) is the computation time, especially the computing time of the objective function.

A great advantage of genetic algorithms as optimization method is the fact that they are suited for parallel computing. The genetic algorithms namely compute the fitness values of all individuals in each generation at the same time. This makes it possible to apply parallel computing to this process. When a program uses parallel computing, it distributes the calculations over different devices or processor cores instead of performing all calculations on one of them. As calculating the fitness function of two individuals are independent processes, they can be computed at the same time by different processors. In this way, multiple fitness values can be calculated at the same time, which decreases the average computing time for a function evaluation, and therefore, parallel computing could provide a possibility to decrease the computing time.

The function 'ga' provides an option for parallel computing when combined with the Parallel Computing Toolbox of Matlab. The default settings of the function 'ga' do not use parallel computing. But in combination with the Parallel Computing Toolbox, it provides to possibility to do so.

However, the software of *GenPro4* also provides the possibility to use parallel computing in order to calculate the objective function. This reduces the computing time and is not compatible with the parallel computing options for the function 'ga'. This means that parallel computing could be performed by either the function 'ga' or by *GenPro4*. If the function 'ga' uses parallel computing instead of *GenPro4*, it takes more computation time for one single function evaluation, but multiple function evaluations can be performed at once. This makes it hard to predict which use of parallel computing results in the fastest genetic algorithm.

To study the effect of using parallel computing within the function 'ga' and *GenPro4*, three different types of runs of a genetic algorithm are performed. In the first type of run, both *GenPro4* and the function 'ga' do not use parallel computing. This run is to determine how much faster the other methods are compared to the default implementation. In the second type of run, *GenPro4* uses parallel computing while 'ga' does not. In the third type of run, *GenPro4* does not use parallel computing while 'ga' does. For each run, the generations, the scores and the computing times in seconds are stored. With the generations and the scores, it can be studied whether the two ways to use parallel computing change the performance of the genetic algorithm. With the computing times, it can be investigated which implementation works faster. All runs of the genetic algorithms are performed with a population size of 100 and the algorithm is terminated after 10 generations. This device runs on windows 8.1 and has an Intel i-4460 CPU with a clock speed of 3.20 GHz and 4 cores. The RAM-memory on this device is 8,00 GB.

Number of cores	Software	Processor	CPU clock speed speed (GHz)	RAM (GB)
2	Windows 8.1	Intel(R) Core(TM) i3-4030U	1.90	6
4	Windows 8.1	Intel(R) Core(TM) i5-4460	3.2	8.00
6	Windows 10	Intel(R) Core(TM) i7-8750H	2.20GHz	16

Table 4.1: The specifications of the used devices to asses the speed of the different options for parallel computing.

All three types of runs are performed on three different devices, which have 2, 4 or 6 cores. More specifications can be found in Table 4.1. Both methods of parallel computing use the maximum number of cores available. However, it is unknown how large the part of the computations in *GenPro4* is that uses parallel computing compared to the part of the function 'ga' that uses parallel computing. These factors make it necessary to experimentally determine the performance of both ways to apply parallel computing. Both ways of parallel computing are at least as fast as not using parallel computing at all. By comparing the computation times for the two different runs on each device, it can be determined which implementation decreases the computing time the most.

To summarize, both *GenPro4* and the function 'ga' provide options to use parallel computing in order to decrease the computation time. The default function of 'ga' does not use parallel computing and the default settings of *GenPro4* do use parallel computing. However, it is not possible to use the parallel computing options of *GenPro4* and 'ga' at the same time, and therefore, it is investigated which implementation computes the objective function faster. Since this might depend on the device used, the genetic algorithm is run on three different devices with each 2, 4 or 6 cores.

5 Results

Chapter 4 describes how the Matlab function 'ga' is used in order to implement a genetic algorithm for Problem (1.1). Some settings of the function are fixed, such as the selection and variation operators. For other settings, Chapter 4 describes what methods are used in order to determine the best parameter values for these settings for this optimization problem. The settings that are be studied in this way are the population size, the stopping criterion and the use of parallel computing. These settings all influence the computation time and possibly the quality of the results. The goal is to find which implementation gives good results in the fastest time. In this chapter, it is discussed what the influences of the different settings are on the quality of the result and the computation time in order to find the best performing implementation.

In this chapter, the results of the different implementations are discussed. This is done by first discussing the implementation of the genetic algorithm using different population sizes in order to find the best population size. The results of these implementations are also studied to find properties of this particular optimization problem and to evaluate the choice of several settings of the algorithm, such as the variation operators. Subsequently, different settings of the stopping criteria are discussed using the information resulting from the implementations for different population sizes. Then, the influences of different ways of implementing parallel computing are studied in order to speed up the computational process.

5.1. Results for different population sizes

The genetic algorithm is run for different population sizes ranging from 25 to 150. First, using the results in general, the choice for several settings that are fixed in Chapter 4 are evaluated and general conclusions about the optimization problem are drawn. Then, the influence of the population size on the computation time and quality of the results is discussed.

5.1.1. General results

The results of the genetic algorithm for different population sizes provide insight in the relationships between the population size, the highest found values of the genetic algorithm and its computation time. However, the obtained results can also offer insight in the behavior of the genetic algorithm. This can be studied using the individuals of all generations and their fitness values that are saved. With this information, insight in how the genetic algorithm behaves for this optimization problem can be obtained. On top of that, it provides the possibility to evaluate the performance of the settings of the function 'ga' that are fixed as described in Chapter 4. These two topics are discussed in this subsection. The analysis of the influences of the population size on the performance is separately treated in Subsection 5.1.2.

Behavior of the genetic algorithms

The genetic algorithms are implemented and executed for the population sizes 25, 50, 75, 100, 125 and 150. Every implementation results in an individual **x**, which is the solution with the best value of the objective function found by the algorithm. The results are shown for every population size in Table 5.1 and Figure 5.1.

In Figure 5.2 and Figure 5.1, it can be seen that the highest current that is found differs for the different population sizes. The highest current in total is a current of 21.0759 mA/cm^2, which is found by the genetic algorithm with population size 125. On the other hand, the lowest result of the genetic algorithm is obtained

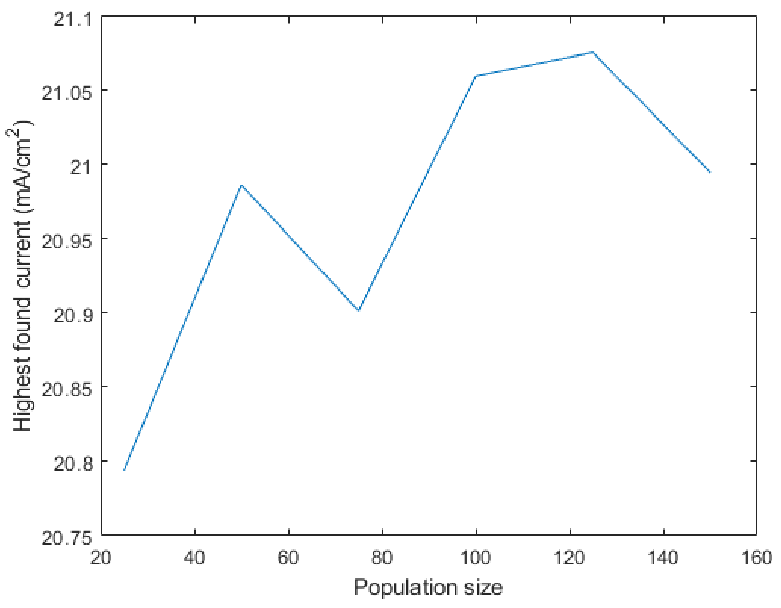


Figure 5.1: The optimal value of the genetic algorithm plotted against the population size.

with a population size of 25 and resulted in a current of 20.7940 mA/cm^2. The corresponding individuals are shown in Table 5.1.

Population Size	Highest found value	Number of function evaluations	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
25	20.7940	1300	0.1031	0.0524	0.2124	0.3654	0.1525	0.3852	2	1	1	1
50	20.9867	2600	0.1009	0.0666	0.1435	0.3824	0.1780	0.0699	2	1	1	1
75	20.9016	3900	0.1012	0.0539	0.2155	0.3691	0.3693	0.2265	2	1	1	1
100	21.0599	5200	0.1004	0.0534	0.0402	0.3813	0.2848	0.0732	2	1	1	1
125	21.0759	6500	0.1051	0.0554	0.0424	0.3847	0.0416	0.0752	2	1	1	1
150	20.9948	7800	0.1011	0.0557	0.3253	0.3785	0.2171	0.0762	2	1	1	1

Table 5.1: The results of the genetic algorithm for each population size.

Figure 5.1 shows how the best individuals of the genetic algorithms for the different population sizes improve over time. It can be seen that all genetic algorithms undergo a similar process. In the beginning, they show large growth, mostly with large jumps. These are most likely due to good results of mutations and crossovers. When the algorithm advances, the improvement in the best values becomes less. The improvements that do occur are small and occur less frequently. In some cases, the best value does not even change. For instance, the algorithm with a population size of 25 does not change after the 20th generation. In such cases, it can be said that the algorithm has converged. It can thus be concluded that in general, the algorithms show large and many improvements in the beginning of the algorithm and small and less frequent improvements at the end.

The individuals in Table 5.1 are local optima or values close to local optima of the objective function. The local optima with the highest function value corresponds to the genetic algorithm with population size 125. All other local optima differ more than 0.01, which is in most cases the accuracy of the objective function as discussed in Chapter 3. Therefore, they are almost certainly significantly lower than the one corresponding to a population size of 125. This means that all other found local optima are just local optima and not global optima. The local optimum corresponding to a population size of 125 could be the global optimum, but it cannot be said for sure due to the fact that a genetic algorithm is not guaranteed to find a global optimum.

When the local optima in Table 5.1 are studied, it can be seen that the local optima are in a similar part of the domain. This can be seen by the fact that most variables have similar values. The values of the variables x_1, x_2, x_4, x_7, x_8, x_9 and x_{10} barely differ for the different optima. This means that the influence of these variables on the performance is most likely strong. This is caused by the fact that the genetic algorithm favors the individuals with the best fitness values, which is the value of the objective function. If a variable has a strong influence on the objective function, this is reflected in the fact that the genetic algorithm first selects the individuals based on this variable. This means that the differences in important variables are decreased by the

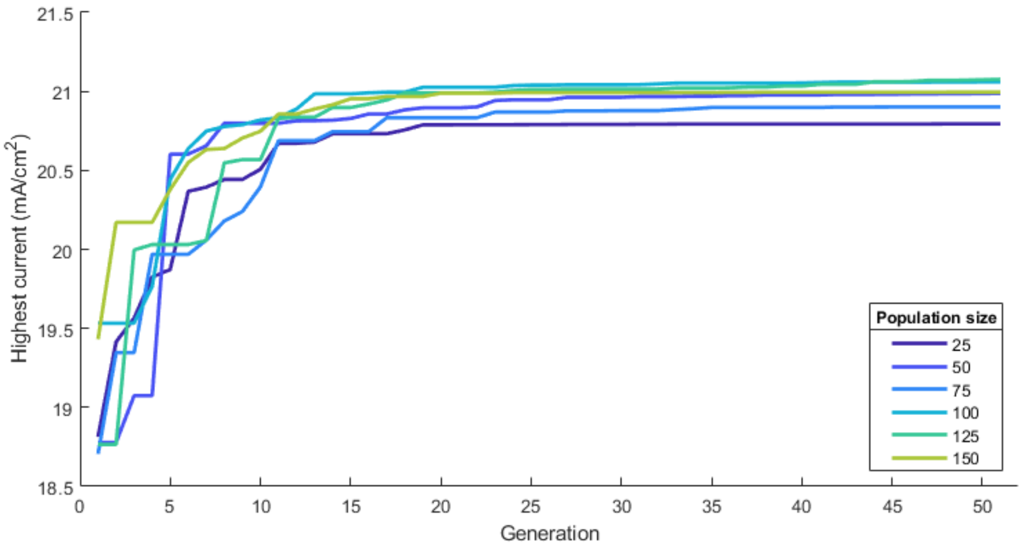


Figure 5.2: The highest found current plotted versus the generation of the genetic algorithms with different population sizes.

genetic algorithm in the beginning of the process. As a result, the optima found by the genetic algorithms for all different population sizes have similar values for these variables.

From this, it can for instance be concluded that the tandem solar cells perform significantly better when it has a pyramid structure between the two interfaces. This can be seen by the fact that the variable x_{10} has the value 1 for all optima. It can also be concluded from the values of x_9 that the anti-reflexive coating has a strong positive influence on the current produced by the tandem solar cell. These two conclusions could have been expected since, both the pyramid structure and the anti-reflexive coating are designed to make the solar cell perform better. Similarly, it can be seen that ETM and HTM are TiO_2 and NiO respectively, which can be concluded from the variables x_7 and x_8. If provided with enough insight of the materials and structure of the solar cell, the values of the variables x_7 up to x_{10} could have been predicted. Therefore, it could be considered to remove these from the optimization problem. However, as this knowledge of the solar cells was not available, this conclusion could not have be drawn beforehand. On top of that, it might be the case that the global optimum of the domain has for instance another ETM, although this is highly unlikely because of the fact that all of these variables seem to be important variables is supported by the results. Therefore, removing these variables would most likely only result in shorter computation time. Because of limited time available for this thesis, this is not studied. However, it could be investigated in further research in order to decrease the computation time.

The importance of the variables can also be investigated in other way. Figure 5.3, Figure 5.4, Figure 5.5 and Figure 5.6 show the scaled standard deviation of all variables for each generation of the genetic algorithms with population sizes 25 and 125. These population sizes are used as they found the worst and best result. In the graphs, the standard deviation is scaled by dividing them by the standard deviation of the variable for the first population. This means that all standard deviations start with a value of 1. This is done because the standard deviation for the different variables was different, especially when the continuous and discrete variables were considered. In this way, the increase and decrease of the variance within the population can be compared for all variables.

Figure 5.4 and Figure 5.6 both show that the standard deviation in the population of the discrete variables decreases fast. This gives that these variables are likely to have a large influence on the objective function as the genetic algorithm selects the individuals with the right values for these variables. Although all standard deviations of the variables decrease quite fast, it decreases less fast for the variable x_7. This means that the ETM is the least important discrete variable.

In Figure 5.4 and Figure 5.6, it can also be seen that the standard deviation does not increase again after it has became equal to zero. This means that the mutation apparently does not mutate the discrete variables after a standard deviation of zero is achieved. It might be possible that the mutations are not applied to the discrete variables at all. This cannot be determined because of the fact that the functions that should apply these mutations are not accessible.

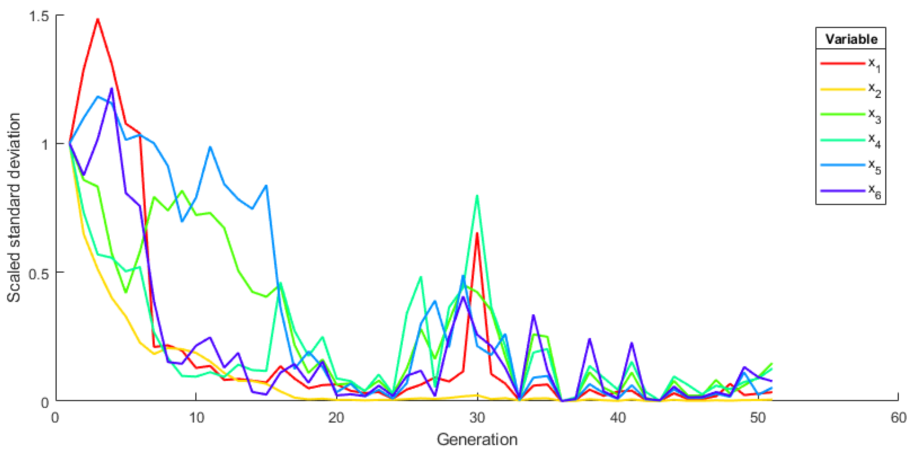


Figure 5.3: The scaled standard deviation plotted for each generation for the continuous variables of the genetic algorithm with population size 25.

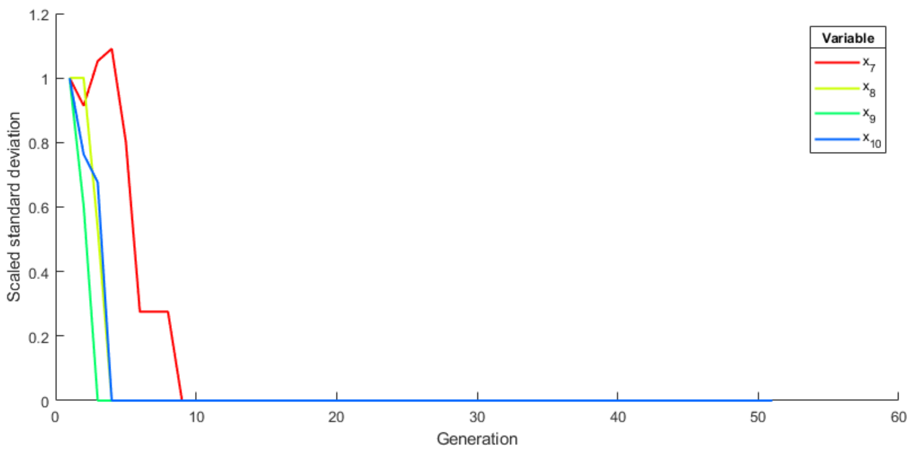


Figure 5.4: The scaled standard deviation plotted for each generation for the discrete variables of the genetic algorithm with population size 25.

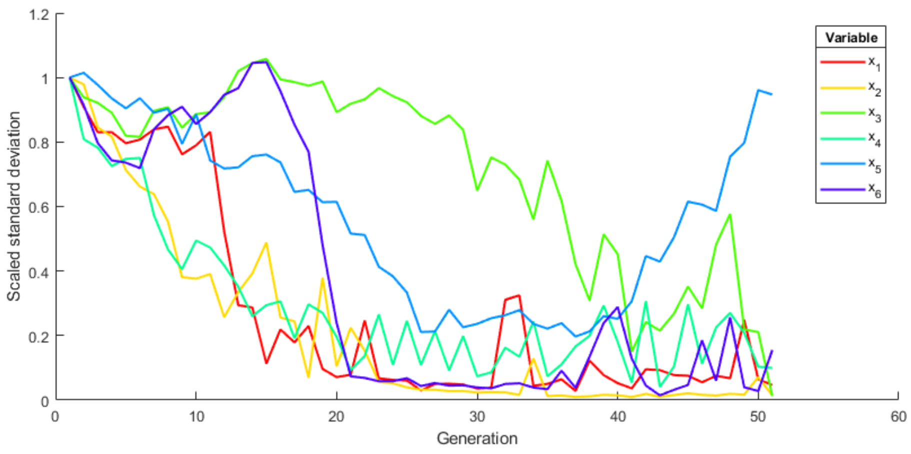


Figure 5.5: The scaled standard deviation plotted for each generation for the continuous variables of the genetic algorithm with population size 125.

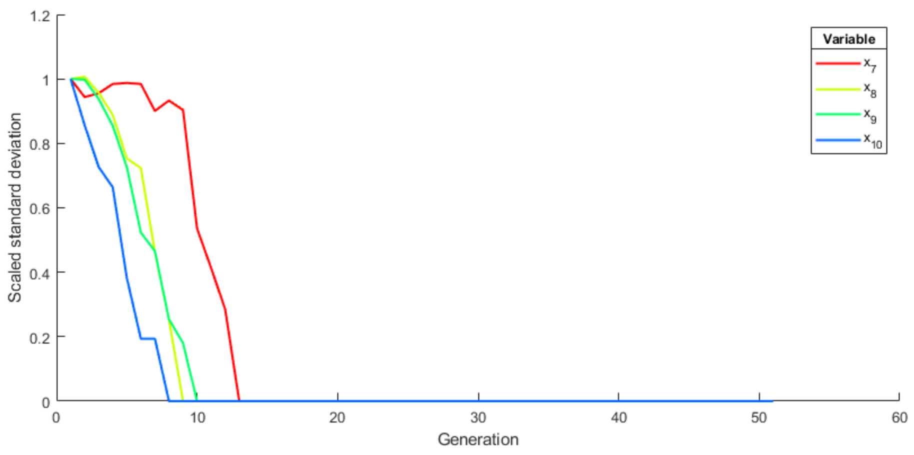


Figure 5.6: The scaled standard deviation plotted for each generation for the discrete variables of the genetic algorithm with population size 125.

Where the discrete variables behaved similarly for the population sizes of 25 and 125, Figure 5.3 and Figure 5.5 show differences in the behavior. The genetic algorithm with a population size of 25 has a larger decrease in the standard deviation in general. This means that it converges faster as the selection pressure is probably larger. This can be studied using for instance the takeover time. However, this could not be implemented within the time available for this thesis.

As can be seen, only in generation 30, the genetic algorithm shows an increase in the standard deviation of multiple variables. In Figure 5.7, it can also be seen that the standard deviation of the current increases at this point. These observations can be explained by one or two large mutations. The influence of these mutations on the standard deviation is larger because of the smaller population size. The genetic algorithm with a population size of 125 on the other hand shows that the variety in the population for some variables are kept longer, such as variable x_3 and x_5. This gives that the variables x_3 and x_5 probably have the least influence on the objective function. This is also supported by the fact that most differences can be found in the variables x_3 and x_5 in Table 5.1.

Figure 5.5 also shows that the variation within variable x_5 increases at the end. This could be caused by a mutation that resulted in a better solution. Subsequently, this solution is more likely to have offspring. As the value of x_5 of the population slowly makes a transition from the old value towards the new value, the variation within the population rises. This means that the genetic algorithm has not yet moved to the area around this better solution. Therefore, the genetic algorithm might not have converged and it might be beneficial to run this algorithm for a larger number of generations.

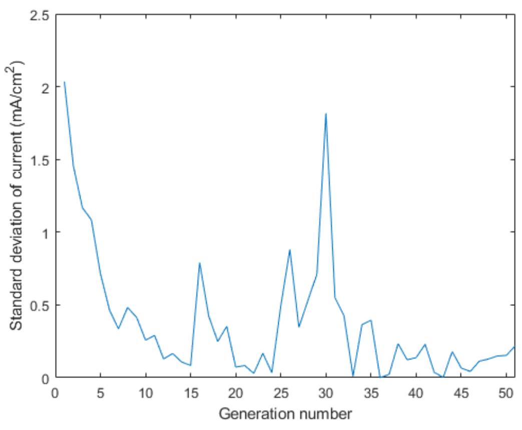


Figure 5.7: The standard deviation of the objective function for each generation of the genetic algorithm with population size 25.

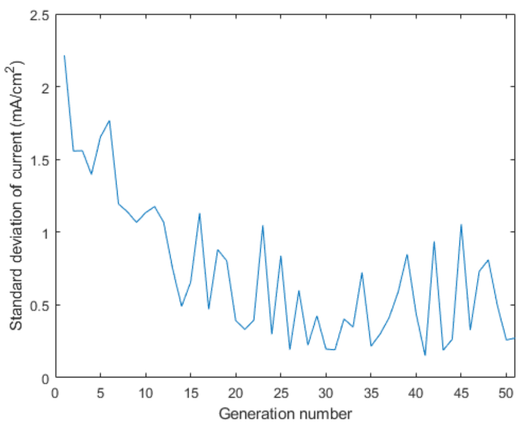
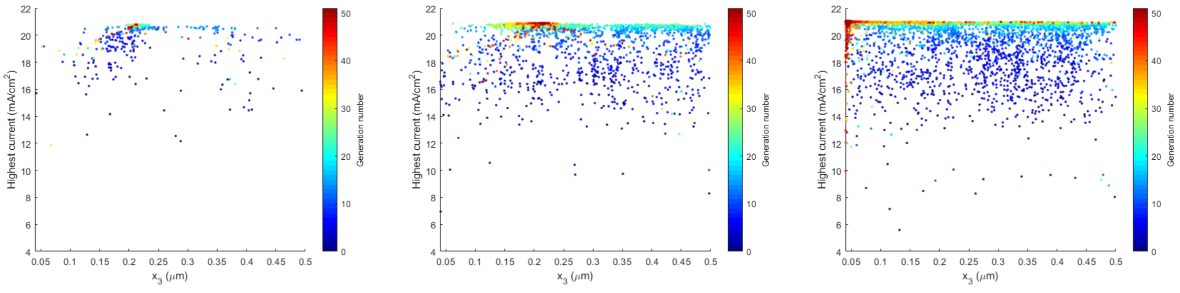


Figure 5.8: The standard deviation of the objective function for each generation of the genetic algorithm with population size 125.



(a) Population size 25 (b) Population size 75 (c) Population size 125

Figure 5.9: The value of the objective function plotted against the x_3-value for all individuals in each generation of the genetic algorithms with different population sizes.

It can also be studied how the different variables influence the objective function. To do this, the current can be plotted against the variable value for all individuals in the genetic algorithms. This is done for the variable x_3 in Figure 5.9 and for x_6 in Figure 5.10. The individual is a 10-dimensional vector and the function value has one dimension. Therefore, the individual with the function value together are a point in a 11-dimensional space. As only one variable and the function values are plotted in Figure 5.9 and Figure 5.10, these form a projection from the 11-dimensional space to a 2-dimensional space. Therefore, information about some local optima might be lost. However, it still provides information about some other local optima.

In Figure 5.9, it can be seen that the variable x_3 does not seem to influence the value of the objective function much as the high values can be found for almost all values of x_3. It does however have a small influence, which is most clear in Figure 5.9c. The objective function seems to be slightly higher for individuals with a value of x_3 between 0.04 μm and 0.05 μm. These might be individuals near to a local optimum with these values of x_3. The algorithm ultimately seems to converge to this optimum.

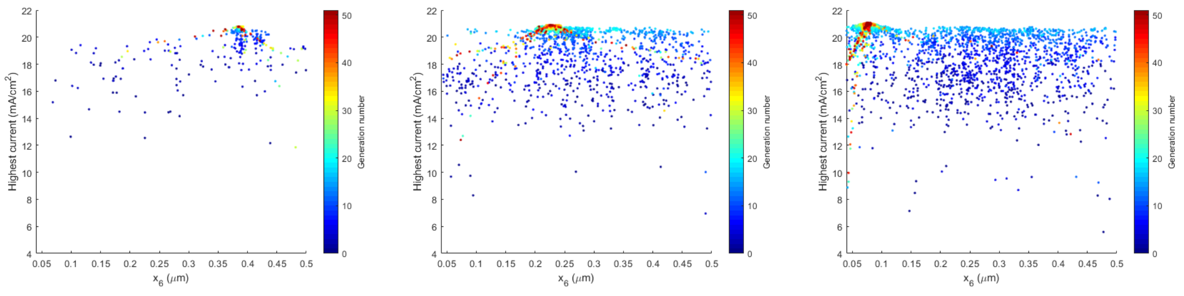
The genetic algorithms with population sizes 25 and 75 have clearly less points in the entire domain. Therefore, a smaller part of the domain is investigated. This can also be seen by the fact that these genetic algorithms do not have individuals with small values for x_3. This is most likely due to the fact that they converged to a local optimum with an x_3-value between 0.20 μm and 0.25 μm. However, this local optimum is hardly visible in Figure 5.9c. This could be explained by the fact that this run did not get trapped in this local optimum. Therefore, the individuals in the lager generations, which have the highest values due to the other variables, are not near this local optimum and it is not visible in the plot.

In Figure 5.10, it can be seen that the variable x_6 influences the current produced by the solar cell more. Figure 5.10c shows that there exist several local optima in which x_6 is involved. The clearest local optimum has an x_6-value around 0.09 μm. This optimum also results in the highest current found by the genetic algorithm. Other local optima have values of x_6 of around 0.25 μm and 0.35 μm. It can be seen in Figure 5.10b that the genetic algorithm with population size 75 converges to a local optimum with a value of x_6 around 0.25 μm. This might be the same local optimum as found in Figure 5.10c. The genetic algorithm with population size 25 also converged to a local optimum as can be seen in Figure 5.10a. This local optimum is however not clearly visible in Figure 5.10c and is also not that high. Figure 5.10a also shows that with a smaller population size, a smaller part of the domain is investigated by the genetic algorithm. Therefore, a larger population size would be preferable.

Figure 5.9 and Figure 5.10 showed local optima which are not that strongly present. This is not the case for all variables. For instance, the variable x_4 shows a clear relationship with the function value. This can be seen in Figure 5.11, where a clear optimum can be seen. This optimum has a value of x_4 of around 0.40 μm. The variable x_4 seems to be the same for the local optima found in Figure 5.9 and Figure 5.10, since these optima have currents clearly higher than 20 mA/cm^2 and all individuals with a current higher than 20 mA/cm^2 seem to have the same values for x_4 in Figure 5.11. Therefore, it can be said that the variable x_4 strongly influences the current and in a smooth way since it has the same values for all local optima.

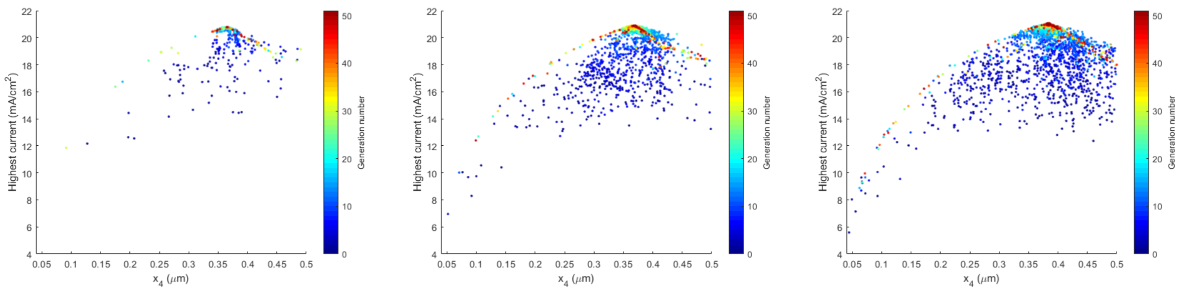
5.1.2. Population size

The first parameter of which the influence is studied is the population size. It can be expected that a larger population size results in a longer computation time as it needs to compute the fitness function for more variables. On the other hand, it also increases the chance of finding a better optimum as a larger part of the domain can



(a) Population size 25 (b) Population size 75 (c) Population size 125

Figure 5.10: The value of the objective function plotted against the x_6-value for all individuals in each generation of the genetic algorithms with different population sizes.



(a) Population size 25 (b) Population size 75 (c) Population size 125

Figure 5.11: The value of the objective function plotted against the x_4-value for each generation of the genetic algorithms with different population sizes.

be studied. The influence of the population size on the trade-off between the computation time and the quality of the result is studied in this subsection. This provides the possibility to make a good choice in this trade-off.

To study the trade-off, the function 'ga' was tested for the population sizes 25, 50, 75, 100, 125 and 150. For all runs, the generations and the corresponding fitness values are stored. These are used to analyze the performance of the genetic algorithms and provide insight in the way the genetic algorithms find a local optimum for this particular problem.

Highest found values

When the genetic algorithms terminate, they give the best individual and its corresponding fitness value as output. These results are shown in Figure 5.1 and the exact data can be found in Table 5.1. They show the highest found fitness values, which are the currents, of the runs for all population sizes. The highest value found for all populations sizes is 21.0759 mA/cm^2. This was found for a population size of 125. The worst performances on the other hand correspond to the algorithm with a population size of 25. This produced 20.7940 mA/cm^2 as the highest found value, which is clearly lower than the other values found. The graph in Figure 5.1 shows that the result of the genetic algorithm increases as the population size increases, except for the population sizes with values of 75 and 150.

For these population sizes, the highest current found by the genetic algorithm is lower than the result of a smaller population size. This could be due to the fact that genetic algorithms are stochastic algorithms. This means that every time it is used, they do different things. For instance, the mutations in the first generations can be different, which might influence the populations of all other generations. Therefore, the results are also variable for each time the genetic algorithms are run. This means that small differences in the results are not likely to be significant and it might even be the case that the run with population size 125 might not perform as well when it is repeated. More reliable information to study this relationship could be gained by repeating the run for all population sizes multiple times. However, it is not possible to realize this due to the limited time available for this thesis. Therefore, conclusions about the individual results cannot be dawn with certainty in this thesis.

On the other hand, conclusions about the general relationships are more likely to hold true as they are based on more measurements. As discussed in Section 2.3 and Section 4.5, a larger population size results in better results in theory. In this case, it can be seen in Figure 5.1 that the genetic algorithm slightly tends to find the best values for larger population sizes. Whereas the results strongly increase on average for population sizes up to 100, the results show a smaller increase for the population sizes of 100, 125 and 150. As the objective function is accurate up to 0.01 mA/cm^2, the values are still significantly different. Despite the uncertainty in the results due to the stochastic nature of the genetic algorithms, it seems to be the case that the results increase less for population sizes of 100 and higher.

In this thesis, the highest population sizes that is considered is 150. Since there does not seem to be a large increase in performance for population sizes of 100 and higher, it might be said that a genetic algorithm with a population size larger than 150 would not perform better than one with a population size between 100 and 150. However, a larger population size would still mean that the genetic algorithm investigates a larger part of the domain for possible optima. On top of that, larger population sizes could possibly decrease the variety in performance due to the stochastic nature of genetic algorithms. When the population size is small, one individual is more influential. This means that the stochastic operations such as mutations or crossovers have more influence when they are used on one individual. Therefore, the subsequent generations and even the results are subject to a larger variability. This could mean that it might be beneficial to have a larger population size, because the genetic algorithm is more likely to find good results. In order to investigate whether this is true, the implementations should not only be repeated for each population size in order to study the variability. It should also be studied for larger population sizes. For this thesis, this is not studied because of the limited time available, but it could be investigated in further research.

Where the differences in the results for the larger population sizes are small, the changes for population sizes smaller than 100 are larger. For instance, the population size of 50 gives a result which is higher than the results for the genetic algorithm with population size 75 and even higher compared to the result for the population size of 25. This could be explained by the variety in performance due to the stochastic nature of genetic algorithms. This performance can be even more influenced by chance due to the small population size as individuals have even more influence compared to larger population sizes. This could explain the large differences in result for the smaller population sizes.

The result of the genetic algorithm with a population size of 50 is much better than the results with population sizes of 25 and 75. The result is even close to some of the results of the implementations with population sizes of 100 and higher. It can be concluded that the genetic algorithm with population size 50 has the potential to give good results. However, nothing can be said about how likely a good result for this population size is, since the algorithm was only run once and the results of genetic algorithms are subject to variability as discussed.

It could also be noted that the population size does not influence the performance on finding the best value much for all variables. Investigating the whole domain of a variable is namely less important when the variable has a clear optimal value. This seems to be the case for variable x_4 as can be seen in Figure 5.11. It can be seen that a clear optimum is found with a value of x_4 around 0.4 μm by the genetic algorithms with all population sizes. Therefore, it is possible to use smaller population sizes of genetic algorithms when only the right values of the most influential variables are wanted.

To conclude, a larger population size in general seems to give better results. As the implementations for the different population sizes are only run once, no hard conclusions can be drawn which particular population size performs best. However, it can be concluded that the implementations with a population size of 100 or larger perform better than these with a population size of 75 and lower. This is because a large increase in result is obtained by a population size of 100. Since the better results of larger population sizes can be due to the stochastic nature of the genetic algorithm, this might be the best value in a trade-off between the computation time and quality of result. On the other hand, the best result is obtained with a population size of 125, which could therefore be favored, depending on how exact the best solution should be found. A population size of 150 or higher could even be better considered that the variability in the results due to the stochastic nature of the genetic algorithm might decrease, although this should be studied more detailed in further research. In general, the population size should be 100 or higher and further conclusions cannot be drawn from only the results of the genetic algorithms.

Number of function evaluations

The population sizes also influence the number of function evaluations, which represent the computational costs. As discussed in Section 4.5, it can be expected that a larger population size increases the computational costs. In Figure 5.12, the number of function evaluations which represent the computational costs are shown for each population size. The data is also included in Table 5.1. As could be expected, Figure 5.12 shows that a larger population size results in more function evaluations.

The graph even shows that this is a linear relationship. The fact that it is linear, is caused by the stopping criterion. All implementations stop after 51 generations. This implies that the stopping criterion might not perform well. This is further investigated in Section 5.2. For this case, it means that the 51 generations and the initial generation, which is not counted in these 51 generations, together are 52 populations. Therefore, the total number of individuals for which the objective function is evaluated is equal to 52 times the population size. This is in line with the number of function evaluations in Table 5.1. It can thus be said that number of function evaluations increases linearly with the population size.

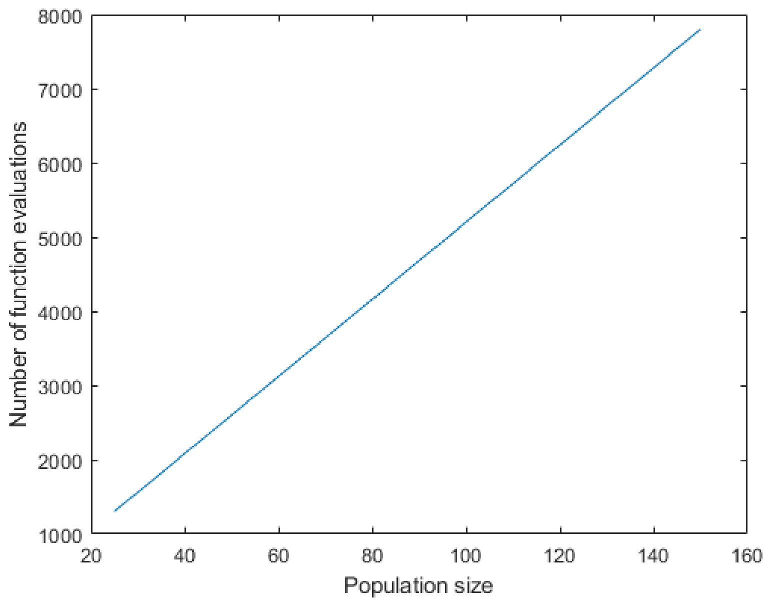


Figure 5.12: The number of function evaluations of the first genetic algorithm plotted against the population size.

In order to investigate the trade-off between the quality of the results and the computation time, Figure 5.13 is made. For each population size, a point is shown with the number of function evaluations as x-coordinate and the highest found current as the y-coordinate. Using this plot, it can be studied how much using an implementation with more function evaluations results in better results. If a population size has a relatively high value for the highest found current and a relatively small number of function evaluations, it is favorable. This is because it finds a relatively good result with small computational costs. In that case, it might be a good choice in the trade-off between the quality of the results and the computation time. As the function evaluations have a linear relationship with the population size, the function is only a scaled version of Figure 5.1. If the stopping criterion is changed, this might be different.

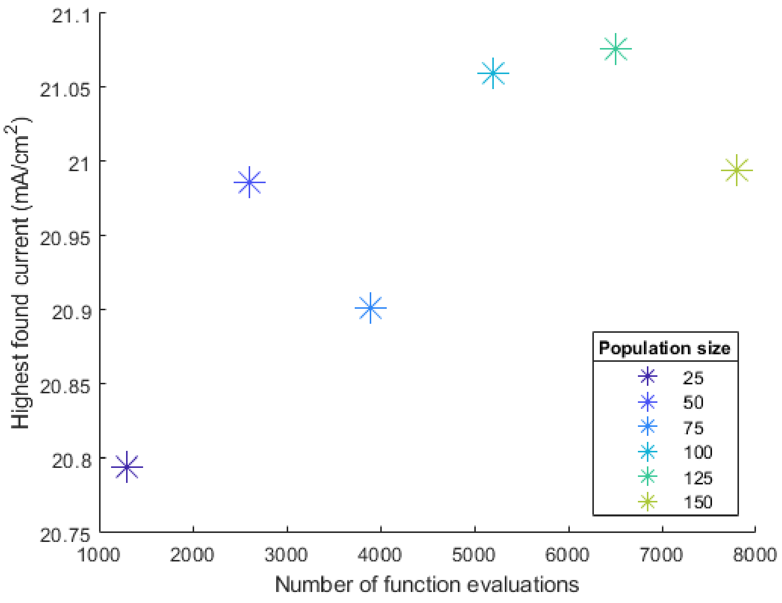


Figure 5.13: The results plotted for each population size against the number of function evaluations.

It can be seen in Figure 5.13 that the implementations with population size 25, 75 and 150 have a relatively low result compared to the computational costs. On the other hand, the implementations with population sizes 50, 100 and 125 have relatively high results for the computational costs. Therefore, these population sizes would be favorable to use, depending on whether a high quality of the result is wished or a short computation time. However, it is important to note that the values are not exact due to the stochastic nature of the genetic algorithm as is discussed earlier in this section. This means that the results might have a relatively high variability. Thus, it is not possible to draw conclusions with certainty about the performances of the population sizes for each individual run.

Nevertheless, it is possible to study the general relationship between the computational costs and the quality of the results for this case. It can namely be seen in Figure 5.13 that in general a high quality of the result has large computational costs. This means that there indeed seems to be a trade-off between a high quality and low computation time. However, more information would be necessary since it would reduce the impact of the stochastic nature of the genetic algorithms on the results. This could be obtained by running the genetic algorithms multiple times and taking averages. It should also be noted that a different stopping criterion might change the number of function evaluations. If the genetic algorithms for some runs have converged earlier than the current stopping criterion, this would mean that they would get a lower number of function evaluations for the same quality of the result. On the other hand, if the algorithms did not converge, a better stopping criterion might result in more function evaluations and possibly a better result. When a new stopping criterion is found in Section 5.2, it should therefore be studied what the consequences of a new stopping criterion are on the computational costs.

To summarize, the best results were found using a population size of 125 and resulted in a current of 21.0759 mA/cm^2. On top of that, a large population size gives a higher function value as output of the genetic algorithm at higher computational costs. This means that there is a trade-off between the quality of the result and the computational costs. A better stopping criterion might change the relationship which describes the trade-off.

5.2. Stopping criterion

As discussed in Section 4.6, the stopping criterion is an important setting to make a choice in the trade-off between finding a good result and having a short computation time for the genetic algorithm. When the stopping criterion stops an algorithm before it has converged, the algorithm is likely to have worse resulting solutions. This is due to the fact that the genetic algorithm is likely to find better solutions when it has converged. On the other hand, the stopping criterion should stop the genetic algorithm not too long after it has converged, because more generations do not contribute to finding better results and increase the computation time of the genetic algorithm. Therefore, a good stopping criterion is vital for a genetic algorithm.

As can be seen in Figure 5.2, all genetic algorithms stopped after 51 generations. It can also be seen that, at this point, some genetic algorithms still increase, and have therefore not converged yet. This is the case for the genetic algorithm with population size 125 and this means that the stopping criterion stops the algorithm too early in this case. On the other hand, some other genetic algorithms like the one with population size 25 do not show any improvement in the results for many generations before it is stopped. Such genetic algorithms have already converged and should be stopped many generations earlier. It can thus be concluded that the stopping criterion with $\epsilon = 0.01$ and $M = 50$ does not perform well and should be changed.

All algorithms stop after 51 generations, while the maximum number of stalling generations is 50. In Figure 5.2, it can be seen that the algorithms show large growth in the first generations. This growth is that large, the the stopping criterion is not met after 50 generations. However, when this growth is not considered anymore in the 51th generation, the average improvement over the generations 2 to 51 is apparently that much less that the stopping criterion is always met. This means that the stopping criterion for this optimization problem basically terminates the genetic algorithm after 51 generations. This stopping criterion is therefore the same stopping criterion as stopping after 51 generations.

As can be seen in Figure 5.2, most algorithms seem to have converged after 51 generations. Therefore, these runs provide enough information to determine the desired stopping points, which are discussed in Section 4.6. Only the genetic algorithm with population size 125 does not seem to have fully converged after 51 generations. This can be seen in Figure 5.5, as the standard deviation of the variable x_5 is high when the algorithm terminates. On top of that, in Figure 5.2, it can be seen that the results are still increasing at the end. This genetic algorithm is therefore executed again in such a way that it stops after 100 generations. Using the results of this execution, the desired stopping points can be determined for the algorithm with population size 125.

Highest found value	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
21.1049	0.1054	0.0445	0.0400	0.3821	0.0424	0.0758	2.0000	1.0000	1.0000	1.0000

Table 5.2: The best solution found by the genetic algorithm with population size 125 after it has ran for 100 generations.

When the genetic algorithm with a population size of 125 is run for 100 generations, the found solution after 100 generations was better than the solution after 51 generations. The new found solution is shown in Table 5.2 and this solution gives a current of 21.1049 mA/cm^2. This is a significant difference with the earlier found 21.0759 mA/cm^2, with respect to the accuracy of the objective function. It can therefore be concluded that the algorithm indeed did not converge yet after 51 generations and should not have terminated.

It can be seen that the solution in Table 5.2 is similar to the solution in Table 5.1 for the population size of 125. The main difference is in the variables x_3 and x_5. This could be expected as it is found that there is a large variation within the population for these variables after 51 generations as could be seen in Figure 5.5. Therefore, the genetic algorithm did not converge yet for these variables after 51 generations. Figure 5.14 shows the spread of the continuous variables of the genetic algorithm with population size run with 100 generations. It can be seen that the spread in variable x_5 strongly decreases after between the 55 generations, which means that the genetic algorithm found a good value for this variable. After 60 generations, all variables have a low standard deviation and the genetic algorithm seems to have converged. However, the standard deviation of x_3 increases around generation 95 and it does not decrease that much until the 100th generation. This means that it might be the case that the genetic algorithm still did not converge at this point. However, the function value does not increase much because of this change. In the last 10 generations, the increase in the best found objective function was just 0.0012 mA/cm^2. This means that these changes in x_3 do not influence the objective function much. Still, there exists a chance that the objective function increases significantly when the of x_3 converge. But, the improvement is likely to be low. Because of the high computational costs to run the genetic algorithm for more generations, the genetic algorithm is not run again for a higher number of generations.

Most genetic algorithms with other population sizes provide enough information to determine all stopping points in the way that is described in Section 4.6. However, for the genetic algorithms with population sizes of 50 and 100, the third type of stopping points, which is meant to have a very accurate with $\delta_j = 0.001$ mA/cm^2, are respectively generation 50 and 47. As discussed, the genetic algorithm with population size 125 did not fully converge after 100 generations. This results in the fact that the run of the genetic algorithm with a population size of 125 has its third type of stopping point at 96. This stopping point and those corresponding to the population size of 50 and 100 are near the last generation that is considered by the genetic algorithms. This means that the increase within the last generations is likely to be significant with respect to the highest desired accuracy. Therefore, the genetic algorithms have not converged enough with respect to this desired accuracy. As described in Section 4.6, the values $S_{i,j}$ for these runs are set to equal ∞ to show that they have not converged

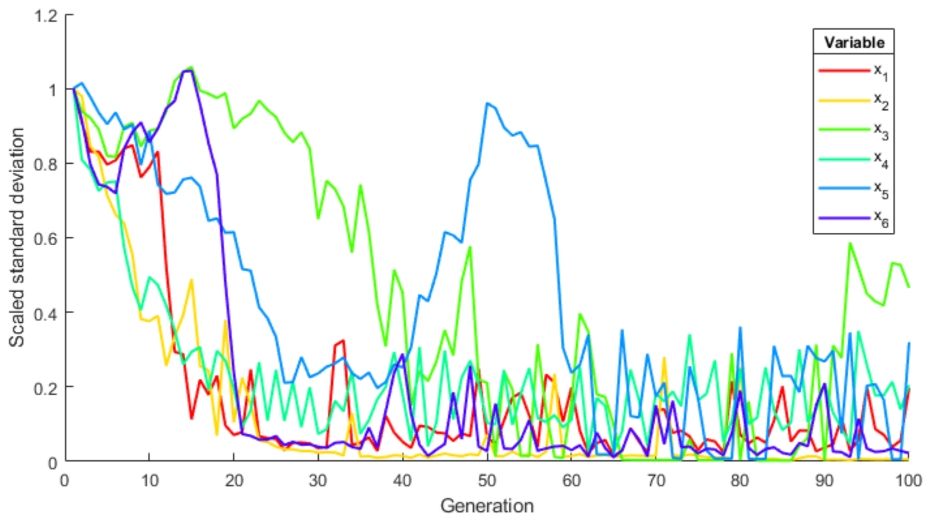


Figure 5.14: The scaled standard deviation plotted for each generation for the continuous variables of the genetic algorithm with population size 125 and 100 generations.

yet.

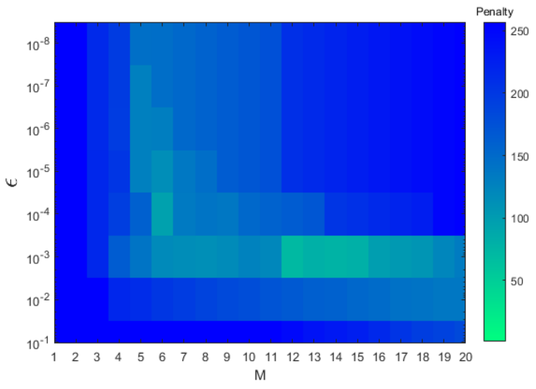
Population size	Best solution		Stopping points for given accuracy		
	Current mA/cm^2	First occurrence	0.01 mA/cm^2	0.005 mA/cm^2	0.001 mA/cm^2
25	20.9867	48	19	23	34
50	20.9867	51	38	∞	∞
75	20.9016	49	35	35	42
100	21.0599	51	33	41	∞
125	21.1049	100	73	78	∞
150	20.9948	50	20	24	38

Table 5.3: All three different types stopping points for all population sizes of the genetic algorithm.

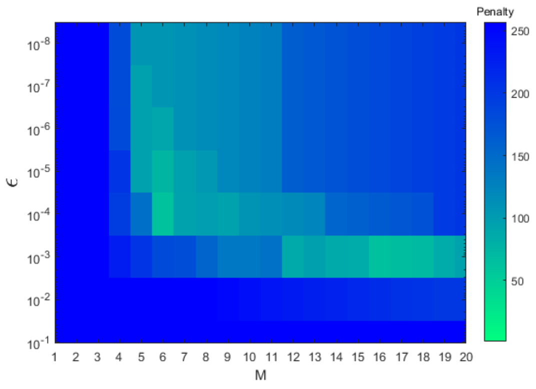
The genetic algorithm with a population size of 50 also has its second type of stopping point with a desired accuracy of $\delta_j = 0.005$ mA/cm^2 at generation 50. This gives the same problem as described for the most accurate type of stopping points corresponding to $\delta_j = 0.001$ mA/cm^2. This problem could be dealt with in the same way. Since this is only one implementation, the loss in information is relatively low compared to when the algorithm would be run again. Therefore, the same method is used to deal with the fact that this algorithm has not converged yet with respect to the desired accuracy.

The stopping points $S_{i,j}$ are determined and shown in Table 5.3. It can be seen that the first occurrences of the best solution in general happen at the end of the run. However, the stopping points are often at an earlier generation. This means that the increase between the stopping points and the last generation is small. It can also be seen that the desired generation to stop in general is a later generation as the desired accuracy increases. However, for the genetic algorithm with population size 75, it can be seen that both the stopping points for low accuracy and average accuracy are the same. This is most likely caused by a large increase in the values in this generation. This makes it will cause the stopping criteria to perform the same for both stopping points for this population size, which is not desirable. The influence effect can be decreased by using more runs of the genetic algorithm. However, this is has a long computation time and is therefore not possible to do for this thesis.

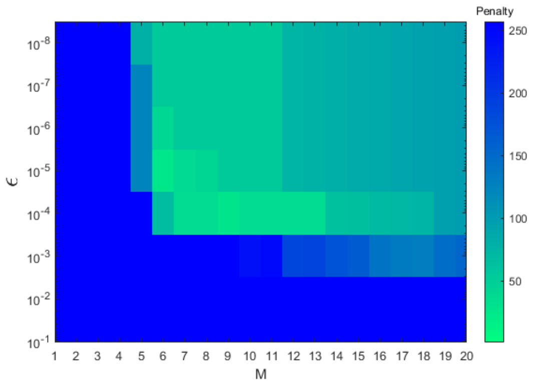
For all different combinations of M and ϵ, the values of $T_i(M,\epsilon)$ are determined in the way described in Section 4.6. Using this, the value of the penalty for each stopping criterion is found for all runs of the genetic algorithms with different population sizes calculated with the method described in Section 4.6. This is done for all three types of desired accuracy δ_j of the stopping criteria. The total penalty of the stopping criterion for every combination of M and ϵ is then obtained by summing up all penalties for the different runs. This value was calculated for all possible combinations of M, ϵ and stopping point type j and the results are visualised in Figure 5.15.



(a) Desired accuracy of $\delta_j = 0.01$



(b) Desired accuracy of $\delta_j = 0.005$



(c) Desired accuracy of $\delta_j = 0.001$

Figure 5.15: The value of the final penalty for each combination of M and ϵ for the three types of stopping criteria.

In Figure 5.15, the value of the penalty can be found for each desired accuracy δ_j. In general, it can be seen that a low value of either M or ϵ results in a bad performance. On the other hand, a high value of both of them also results in a high penalty. For all three types of desired accuracy, it can be seen that the points with low penalty values seem to have an L-formed shape. This means that a high value of ϵ needs a low value of M to perform well and vice versa. Where this shape occurs and at what point the best penalty is, differs for all three types of desired accuracy.

The values that are considered for the parameter ϵ are all smaller than the value that was initially used for the genetic algorithm to implement it for different population sizes. Figure 5.15 shows that the stopping criteria with this ϵ perform badly. This value of 0.01 was based on the fact that the objective function is accurate up to 0.01 mA/cm^2. However, the stopping criterion takes the average increase of the last M generations. If a significant improvement of 0.01 mA/cm^2 would be made over M generations, the average change would be $\frac{0.01}{M}$, which is clearly smaller than 0.01 mA/cm^2. This explains the bad performance of 0.01 mA/cm^2.

If an accuracy of the result of $\delta = 0.01$ is desired, Figure 5.15a shows the penalty of the different values of M or ϵ. This stopping criterion should terminate the algorithm relatively fast in order to reduce the computation time. It can be seen that the stopping criterion with the best performance has the parameters $M = 12$ and $\epsilon = 10^{-3}$. This stopping criterion performs clearly better than the other stopping criteria for this optimization problem. However, it can be seen that the stopping criteria still performs well for values of M near 12. On top of that, $M = 6$ with $\epsilon = 10^{-4}$ does not perform that much worse than the stopping criterion with parameters $M = 12$ and $\epsilon = 10^{-3}$.

Figure 5.15b shows the penalty for the different values of M or ϵ when a stopping criterion with an accuracy of $\delta = 0.005$ is wanted. This stopping criterion should focus on both a short computation time and a good quality of the final solution. In Figure 5.15b, it can be seen that the best combination is $M = 16$ together with $\epsilon = 10^{-3}$. Just as in Figure 5.15a, it can be seen that other values of M close to 16 also perform well. Moreover, Figure 5.15b shows that the combination of $M = 6$ and $\epsilon = 10^{-4}$ performs well too. Although the best values in Figure 5.15b are slightly different to those in Figure 5.15a, the figures show a similar pattern of the well performing stopping criteria

On the other hand, Figure 5.15c shows a clearly different behavior of the penalty when an accuracy of 0.001 mA/cm^2 is desired. It can be seen that the combination of $M = 6$ and $\epsilon = 10^{-5}$ performs the best in this case. However, other combinations with similar values of M and ϵ also tend to perform well. Combinations with for instance $\epsilon = 10^{-4}$ and M between 7 and 13 also seem to perform very well.

In general, the best values for $\delta_j = 0.01$ mA/cm^2, $\delta_j = 0.005$ mA/cm^2 and $\delta_j = 0.001$ mA/cm^2 show similarities. It can be seen that two regions in Figure 5.15 seem to perform well. These two regions describe two types of stopping criteria with different focuses. The first of these regions is the one with $\epsilon = 10^{-3}$ and values of M from 12 to 18. The first type of stopping criteria follows from this region. This stopping criterion has a relatively low desired accuracy and is called stopping criterion A. This results in the fact that it can be used when the stopping criterion should focus on a fast result at the cost of a lower quality of the result. The second region that seems to perform well is the region around $M = 6$ and $\epsilon = 10^{-4}$. This gives rise to the second type of stopping criteria that performs better when a high quality of the result is desired. This stopping criterion is called stopping criterion B.

The first of these regions is the one with $\epsilon = 10^{-3}$ and values of M from 12 to 18. This region performs best for both the desired accuracies $\delta_j = 0.01$ mA/cm^2 and $\delta_j = 0.005$. Therefore, these stopping criteria with these combinations seem to perform well when a lower accuracy is desired. This is the case when the genetic algorithm is implemented in order to get a fast result at the cost of some quality of the solution. The value of M that performs best for these two desired accuracies together is the value of $M = 12$, although higher values of M also seem to perform well. A stopping criterion with a lower value of M also tends to terminate the algorithm earlier. This is favorable when the focus is on finding a sufficient result in a short computation time. Therefore, the value of $M = 12$ is best for stopping criterion A.

The second region that seems to perform well is the region around $M = 6$ and $\epsilon = 10^{-4}$. This region performs best for both the desired accuracies $\delta_j = 0.005$ mA/cm^2 and $\delta_j = 0.001$. This region thus performs well when a high accuracy is desired. A high accuracy is desired when focus is mainly on finding a solution with a high function value and the computation time does not matter much. The combination of $M = 6$ and $\epsilon = 10^{-4}$ seems to perform well for both desired accuracies $\delta_j = 0.005$ mA/cm^2 and $\delta_j = 0.001$. Therefore, the stopping criterion with this combination is a likely to find a very accurate value due to the good performance for $\delta_j = 0.001$ mA/cm^2. The stopping criterion corresponding to $M = 6$ and $\epsilon = 10^{-5}$ also performs well when an accurate solution should be found. However, it is very close to the stopping criterion with $M = 6$ and $\epsilon = 10^{-4}$. As the stopping criterion also has a good performance when the focus is not solely on a high accuracy. This can be seen by the fact that it also performs well for $\delta_j = 0.005$ mA/cm^2. This means that this combination is a better value in the trade-off between a high quality of the final solution and a short computation time when the focus should be mostly on the high quality. Therefore, the stopping criterion with parameters $M = 6$ and $\epsilon = 10^{-4}$ is chosen as stopping criterion B, which focuses on finding a good solution rather than a short computation time.

Table 5.4 shows the resulting stopping points for these two types of stopping criteria if they would be implemented for each population size. It can be seen that the found stopping criteria sometimes terminate the genetic algorithm at significantly later generations than would be ideal. This is caused by the fact that a genetic algorithm often shows no growth for periods of times and then suddenly finds a better solution. This can for instance be seen in Figure 5.2. This means that taking a low value of M or a high value of ϵ causes the stopping criterion to stop at such places whereas it would be desired to stop at a later generation. Since the penalty function punishes such errors stronger as they result in significantly worse solutions, low values of M or high values of ϵ perform bad corresponding to the penalty. Therefore, some higher values of M with low value of ϵ are found as the best combinations. As the penalty is designed like this, it is not necessarily something bad.

However, it can also be noticed that the stopping criteria sometimes terminate the genetic algorithm at an earlier generation that is desired. In this case, the genetic algorithm shows no growth for a very long time and the stopping criterion stops the algorithm. Although this type of behavior results in a higher penalty, it still happens sometimes. However, does not happen often and at all moments that it does happen, it is just a few generations before the ideal stopping point. Therefore, the loss in quality of the solution is likely to be low. This means that it this behavior is not a significant problem.

As discussed in Section 4.6, it is also possible to use a stopping criterion that stops when there is almost no variation within the population. As can be seen in Figure 5.3 and Figure 5.5, the variation within the population for the continuous variables does not vanish. Mutations within the population cause the variation to increase. As this happens often, such stopping criterion is likely to stop at a very late generation. As this increases the computation time significantly and the default stopping criterion seems to perform well for the right parameters, a different type of stopping criterion is most likely not better for this optimization problem.

Population size	Ideal stopping points			Real stopping points	
	0.01 mA/cm^2 accuracy	0.005 mA/cm^2 accuracy	0.001 mA/cm^2 accuracy	Focus on computation time (*A*)	Focus on good solution (*B*)
25	19	23	34	31	40
50	38	∞	∞	48	∞
75	35	35	42	46	41
100	33	41	∞	39	39
125	73	78	∞	63	∞
150	20	24	38	32	30

Table 5.4: The ideal stopping points for the desired three types of stopping criteria and the stopping points of the found best stopping criteria for each of the desired types.

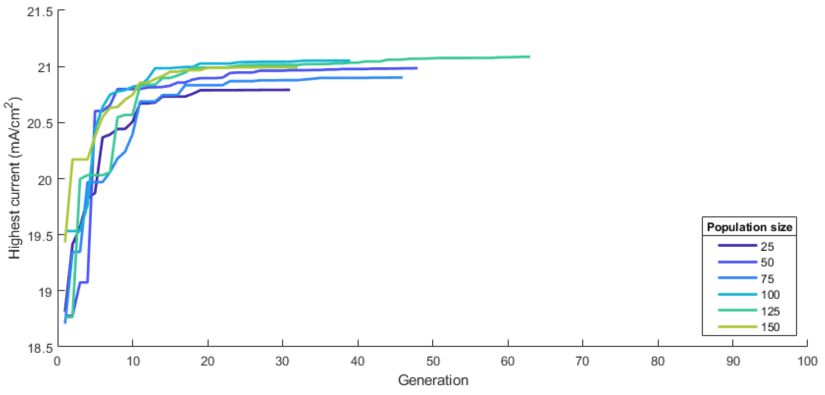


Figure 5.16: The highest found current plotted versus the generation of the genetic algorithm with different population sizes. The stopping criterion that focuses on a short computation time is used (stopping criterion *A*).

Now that the two types of stopping criteria are found, they can be applied to the results of the genetic algorithms implemented for the different population sizes. This means that the algorithms stop in the generations that are specified in Table 5.4. When these are used, the new results of the genetic algorithm can be formulated. How the genetic algorithms would have performed with the two new stopping criteria is visualised in Figure 5.16 and Figure 5.17.

In Figure 5.16 and Figure 5.17, it can be seen that the new stopping criteria indeed stop the algorithm earlier compared to Figure 5.2 for some of the population sizes. For some of the population sizes, the genetic algorithm is not stopped by the stopping criterion, and in these cases, the graph is exactly the same. It can also be seen that stopping criteria *A* terminates the algorithm earlier. This saves computation time as the subsequent generations do not have to be considered anymore.

As the computation times change for the different population sizes, the number of function evaluations also change. This means that the points in Figure 5.13 have also changed. The new number of function evaluations is determined for the different stopping criteria for each population size. With this, the graphs in Figure 5.19 are made.

Figure 5.19 shows that the better stopping criteria influence the choices in the trade-off between a short computation time and a good solution. This can be seen by the fact that it is clearly different than Figure 5.13. Most points have moved towards a lower number of function evaluations at the cost of a slightly lower found current. However, the decrease in the highest found current is in general not that large whereas the computation time does decrease significantly. Therefore, the stopping criteria seem to to perform well.

In both Figure 5.18a and Figure 5.18b, it can be seen that a better result in general comes at the cost of a longer computation time. It can also be seen that a larger population size is more likely to find a better result, although this is up to some variation. For determining the exact relationships between the quality of the results, the computation times and the population size, more information would be needed. However, this has a long computation time and is therefore not possible due to the limited time for this thesis. This could be a topic for further research.

With the information available, a possible choice in the trade-off between the computation time and the quality of the result could be made. Figure 5.18a shows that a population size of 50 or 100 seems to perform relatively well for the computation time it needs. However, the results of these population sizes could still vary. Therefore, nothing can be said about the choice for a specific population size. Nevertheless, it is possible to advice a range of population sizes. The populating sizes with values from 50 to 100 seem to perform relatively

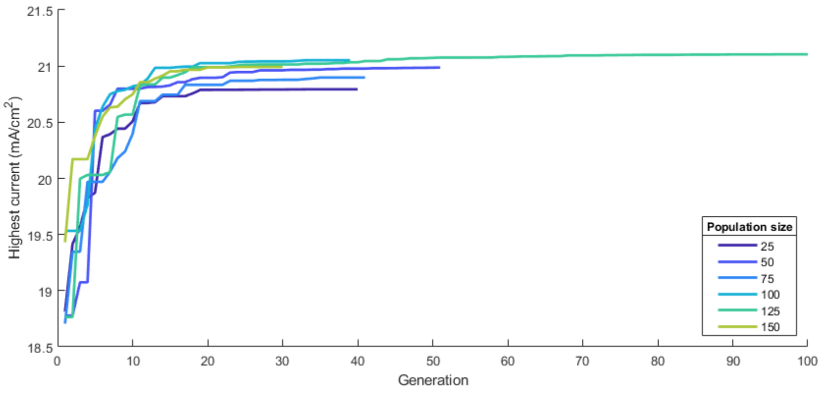
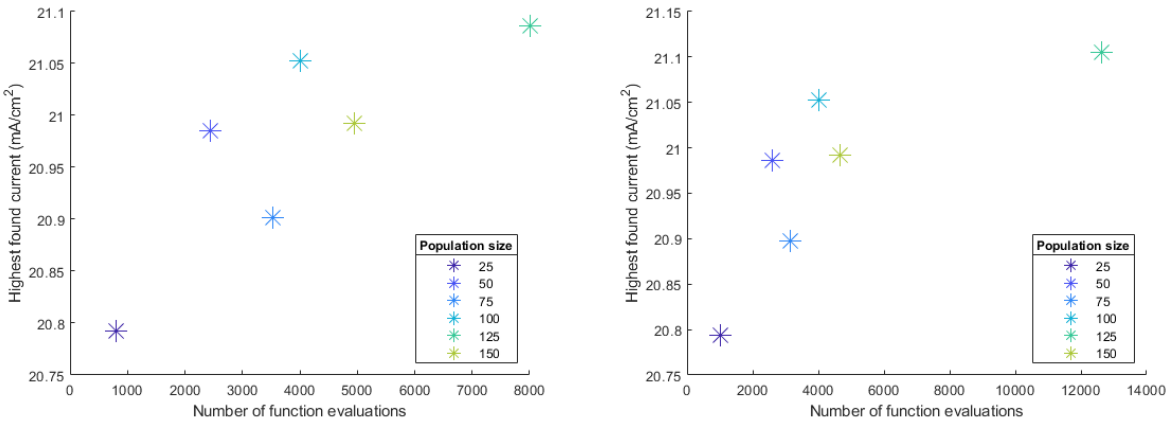


Figure 5.17: The highest found current plotted versus the generation of the genetic algorithm with different population sizes. The stopping criterion that focuses on a good final solution is used (stopping criterion B).



(a) Stopping criterion focused on short computation time (stopping criterion A).

(b) Stopping criterion focused on a good solution (stopping criterion B).

Figure 5.18: The current of the best solution of the genetic algorithm for each population size plotted against the number of function evaluations if the found stopping criteria would be used.

well for their computation time with the stopping criterion focused on a short computation time. Thus, if a good result is desired in a short computation time, the genetic algorithm should be implemented with a population size between 50 and 100 and stopping criterion A.

On the other hand, if a very accurate result is wanted, the choice should be different. The population size of 125 appears to perform very well and could therefore be suggested as best option. However, the performance of the genetic algorithm is subject to a stochastic process and varies. Nevertheless, it can be seen in Figure 5.18b that the genetic algorithm seems to give the best solutions for higher population sizes. It can be seen by the fact that the higher population sizes of 100 and higher give relatively high values for the found current. As a high quality of the solution is wanted, the computation time does not matter that much. This means that it the higher computation time for these population sizes is acceptable. In general, higher values are found using the stopping criterion meant for a high accuracy. Therefore, the best settings for a genetic algorithm are a population size of 100 or higher and stopping criterion B.

It should be noted that the data might be slightly misleading. This is due to the fact that information is missing. For instance, some genetic algorithms would not have stopped for the implemented stopping criteria as could be seen in Table 5.4. This means that the computation times most likely would have been higher for these genetic algorithms. On the other hand, the highest current might also have been higher. Since it is unknown how much any of these two quantities would have changed, the information cannot be filled in. The results should therefore be interpret with care as is done in this section by not making conclusions about specific settings. For instance, only conclusions are drawn about ranges of population sizes.

To summarize, the stopping criterion implemented in the genetic algorithm used to investigate the influence of the population size performs poorly. Two types of good stopping criteria for the genetic algorithm are found as good choices in the trade-off between a short computation time and a high quality of the solution. The first stopping criterion is stopping criterion A and has parameters $M = 12$ and $\epsilon = 10^{-3}$. This stopping criterion terminates the genetic algorithm relatively fast in order to reduce computation time, but it still manages to find relatively good solutions. The second stopping criterion is stopping criterion B and terminates the genetic algorithm at a later generation, which makes it possible to find better solutions. This is done at the cost of a longer computation time. The parameters stopping criterion B are $M = 6$ and $\epsilon = 10^{-4}$.

5.3. Parallel computing

As discussed multiple times, the most limiting property of the implemented genetic algorithms is the computation time. This is caused by the high computational costs for the objective function. A function evaluation of the objective function costs 21.8 seconds. The other actions performed by the genetic algorithm take multiple orders of magnitude less time. Therefore, the computation time of the genetic algorithm is almost solely determined by the computation time of the objective function.

As discussed in Section 4.7, parallel computing can be applied to decrease the average computation time of the objective function. This can be done by using parallel computing within *GenPro4*, which makes one function call faster. It can also be done by using parallel computing within the genetic algorithm. This makes it possible to compute multiple function calls at one time, which decreases the average computation time of the objective function.

To determine what implementation of parallel computing is the fastest, a genetic algorithm with a population size of 100 is executed for 10 generations. This resulted in 1056 function calls. This is higher than 1000 as the starting population is not considered as a generation by the function 'ga'. This would increase the number of function calls by 100. On the other hand, the genetic algorithm has some elitism, which means that the best individuals of a generation are automatically placed in the new generation. For these individuals, the objective function does not have to be called again as it is already known. This decreases the number of function calls from 1100 to 1056.

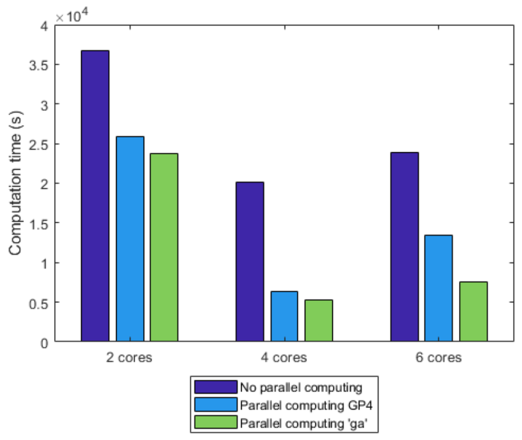
The genetic algorithms are run on three devices with a GPU of each 2, 4 or 6 cores. The specifications of these devices are given in Table 4.1. On all these three devices, the genetic algorithm is run with all different options of applying parallel computing. These three options are no parallel computing, parallel computing in *GenPro4* or parallel computing in the genetic algorithm using the function 'ga'. In total, the genetic algorithm is executed 9 times and all individuals, the resulting solution and the computation time are stored. This resulted in the computation times for each execution on each device. These results are shown in Figure 5.19a.

In Figure 5.19a, it can be seen that that the genetic algorithm is faster when parallel computing is used. Both ways of using parallel computing significantly decrease the computation time needed for the genetic algorithm. For all devices, the genetic algorithm is fastest when the parallel computing is applied within the function 'ga'. The fastest computation time is 5300 seconds, which is around 1.5 hour. This means that the average computation time of the objective function is around 5.0 seconds. This is a huge improvement compared to the 21.8 seconds for the original situation.

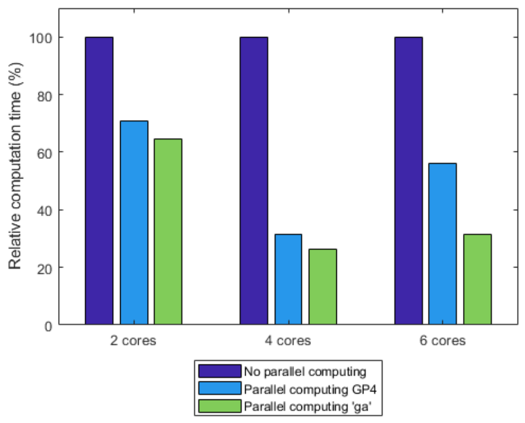
It can be seen that the genetic algorithm takes the longest to compute for the device with 2 cores. This could be due to the fact that this device has older hardware. It can also be seen that the device with 4 cores is faster than the device with 6 cores. This could also be caused by worse hardware. This means that the three devices do not all have the same computational speed. The fact that the objective function without parallel computing takes different computation times on the different devices makes it difficult to compare the influences of parallel computing on the computation time. In order to make the computation times comparable for the different devices, the computation times are seen as percentages of the computation time without parallel computing. The results can be seen in Figure 5.19b.

Figure 5.19b shows that the most decrease of computation time is gained for the devices with 4 or 6 cores. This is not strange as more tasks can be computed at these devices at the same time. It can also be seen that the effect of parallel computing in general is stronger for the device with 4 cores. This is strange as it could be expected that the effect would be the largest for the device with the most cores as this device can handle more tasks at the same time. This observation could be explained if the hardware of the device with 4 cores is build better for handling tasks in a parallel way. However, this is hard to determine.

Figure 5.19b also shows that the computation time between parallel computing with *GenPro4* and the genetic algorithm differs the most for the device with 6 cores. It could be the case that the software of the function 'ga' is better designed to run on more cores than *GenPro4*. However, this is not the case when the differences



(a) The absolute computation times



(b) The relative computation times

Figure 5.19: The absolute and relative computation times in percentages for the objective function using no parallel computation, parallel computation within *GenPro4* and parallel computation within the genetic algorithm. The computation times are shown for the three different devices with all different number of GPU cores.

for the devices with 2 and 4 cores are compared. The difference in the computation times is namely not clearly stronger for the device with 4 cores. On top of that, the results are based on measuring the computation time on just three specific devices. This does not necessarily make it generalble for other devices. It can therefore not be concluded that the difference in computation time between using parallel computing with *GenPro4* and the function 'ga' is larger when the device uses more cores.

For all runs of the genetic algorithm, the individuals and resulting solutions are stored and used to determine whether the different implementations of parallel computing changed the way the genetic algorithm performs. For all runs of the genetic algorithm, the best found solution within 10 generations had a function value of 20.7223 mA/cm^2. On top of that, all individuals and fitness values within the populations were the same. This means that parallel computing does not change the performances of the genetic algorithm. However, the genetic algorithm must be initialized slightly differently when parallel computing is used. The first function evaluation determines the scatter matrices. Therefore, this initial function evaluation is performed before the genetic algorithm is started. The implementations used for the genetic algorithms for the different population sizes as in Section 5.1 first draw the initial population and then have the first function call. This means that the random drawings are performed in a different order as the first function call is now made before the initial population is drawn. Due to the way the random generator works, this makes the genetic algorithms perform slightly differently. However, for new runs of the genetic algorithm, this does not influence the performance either negatively or positively.

In general, the data shows that using parallel computing within the genetic algorithm results in the shortest computation time. This means that an implementation with this type of parallel computing would be preferable. However, the function has to be called before the genetic algorithm in order to determine the scatter matrix. This matrix has to be used by all other function calls and should therefore be determined before the function calls are performed in parallel. As discussed, this is done before the genetic algorithm is implemented. However, this first function call takes around 30 minutes to compute as it is computed without parallel computing. The computation time shown in Figure 5.19a and Figure 5.19a do not include this computation time. Whereas it is not possible to compute the first function call within the parallel process of the genetic algorithm, it is possible to use the parallel computing of *GenPro4* to compute it. This means that this computation time can also significantly decreased.

Because of the fact that the genetic algorithm takes less time to compute, the choices in the trade-off between the computation time and the quality of the final solution are influenced. If the computation time is almost 4 times shorter, more accurate settings can be used within the same computation times. It is for instance possible to use a genetic algorithm with a larger population size while the computation time stays the same as when no parallel computing was used. This increases the chance to find the best optima of the genetic algorithm.

For instance, the best result for this optimization problem was found with a genetic algorithm with population size 125 after 100 generations. This means that approximately 12500 function evaluations are performed. Without parallel computing, this would take around 3 days to compute. However, this computation time could be reduced to around 17 hours using parallel computing in the ideal situation. If the genetic algorithm would be started at the end of the work day, the genetic algorithm would be done at the start of the next work day. In this way, the genetic algorithm could find good solutions within a good computation time.

On the other hand, it can still be possible that a result is wanted in a as short as possible computation time. In this case, the computation time is even shorter. This could for instance be done by letting the genetic algorithm with population size 25 run with the stopping criterion that focuses on a short computation time. Table 5.3 gives that it would terminate after 41 generations, which means that it would have around 1000 function evaluations. As is found in this section, the computation time of this could only be 1.5 hours when parallel computing would be used in the genetic algorithm. However, this genetic algorithm would only have found a current of 20.7934 mA/cm^2 due to the choice in the trade-off between a good computation time and a good final solution.

To summarize, it can be concluded that the computation time of the genetic algorithm is decreased when parallel computing is used in either *GenPro4* or the genetic algorithm itself. The shortest computation time is gained by applying it to the genetic algorithm itself and it proved the possibility to decrease the computation time to 25% of the original computation time in the ideal situation.

5.4. Case study

In this thesis, genetic algorithms are implemented in different ways. This is done in order to find the influence of certain settings on the performance of the genetic algorithm, which is assessed using the quality of the best found solution and the computation time of the genetic algorithm. It is found that these choices often result in a trade-off between a high quality of the solution and a short computation time. There are two runs of genetic algorithms that performed well. In this section, these two best runs are discussed.

The genetic algorithm with population size 50 finds a relatively good solution within a short computation time. On the other hand, the genetic algorithm with a population size of 125 finds the best solution of all implementations of genetic algorithms. However, this comes at the cost of a long computation time. These two different runs of the genetic algorithms perform well for two different possible choices in the trade-off. However, the settings of the stopping criterion and parallel computing that are used in these implementations are not the settings that are found to be well performing in Section 5.2 and Section 5.3. In this section, it is therefore also discussed what would have happened in these two best cases if the right settings for the would have been used.

5.4.1. Focus on high quality of the solution

The first case that is discussed is a case in which the best solution is desired and the computation time does not matter much. As discussed, the genetic algorithm with population size 125 appears to find the best solution. In Section 5.1, genetic algorithms are run for population sizes 25 up to 150. These genetic algorithms are stopped after 51 generations. The best solution is found using a population size of 125 at the cost of a long computation time. This solution can be seen in Table 5.1 and results in a current of 21.7059 mA/cm^2. In Section 5.2, the genetic algorithm with population size 125 is implemented again to run for 100 generations. During these extra generations, the genetic algorithm was able to even find a better solution, which is shown in Table 5.2 and results in a current of 21.1049 mA/cm^2. The information of this run is used to study this case in which the focus is on a high quality of the solution.

As discussed in Section 5.2, the stopping criterion that is implemented does not perform well. Instead, two better stopping criteria are found. Stopping criterion B focuses on a high quality of the solution of the genetic algorithm, which is exactly what is desired in this case. This stopping criterion stops the genetic algorithm if the increase in the best value over the last $M = 6$ generations is less than $\epsilon = 10^{-4}$. However, this stopping criterion does not terminate the genetic algorithm with population size before the 100th generation. But, as could be seen in Figure 5.14, the variation within the population is relatively low after 100 generations. Although this does not hold for variable x_3, it is found in Subsection 5.1.1 that this variable does not have as strong influence on the objective function. This means that despite the fact that the algorithm did not converge for this variable yet, the result of the genetic algorithm is not likely to improve much. Therefore, the first 100 generations are considered in this case study.

This genetic algorithm with population size 125 is originally not implemented with parallel computing. As the genetic algorithm takes 101 generations of each 125 individuals, approximately 12500 function calls are performed. Without parallel computing, this results in a computation time of around 3 days. As discussed in Section 5.3, parallel computing within the genetic algorithm could decrease this computation time up to 75%. This results in a computing of 17 hours. This means that if the genetic algorithm is started at the end of a work day, the algorithm is likely to have results with a very high quality the next morning.

In short, the genetic algorithm with a population size of 125 performs the best when a high quality of the solution is desired. In this case, it is best to use stopping criterion B, which stops the genetic algorithm if the improvement of the best value of the objective function over the last 6 generations is on average less then 10^{-4}. This implementation of the genetic algorithm finds a current of at least 21.1049 mA/cm^2 and this can be found within 17 hours when parallel computing is applied.

5.4.2. Focus on a short computation time

Another possible choice in the trade-off between a high quality of the final solution and a short computation time is to focus on a short computation time. In this case, it is still important to find a solution that is as good as possible within this short computation time. As discussed, the genetic algorithm with a population size of 50 found relatively good results within a short computation time. Therefore, this implementation is discussed now.

The genetic algorithm with population size 50 is originally implemented with a stopping criterion that stops after 51 generations. After this number of generations, the best solution results in a current of 20.9867 mA/cm^2. In Section 5.2, it is found that stopping criterion A, which terminates the genetic algorithm when the increase in the objective function over $M = 12$ generations is less than $\epsilon = 10^{-3}$ mA/cm^2 is good choice when a relatively good solution is desired in a short computation time. Therefore, this stopping criterion is implemented for a genetic algorithm with a population size of 50. As can be found in Table 5.4, this genetic algorithm is then terminated after 48 generations as it has converged at this generation. This highest current found after 48 generations is 20.9848 mA/cm^2.

With this population size and stopping criterion, around 2400 function evaluations are performed by the genetic algorithm. This genetic algorithm is originally implemented without any form of parallel computing. In that case, the run of the genetic algorithm takes around 15 hours. However, if parallel computing would have been implemented within the genetic algorithm, the computation time would be significantly lower. In that case, the genetic algorithm would take 3.5 hours to compute. This means that multiple runs can be executed within one working day. The current of 20.9848 mA/cm^{2}is relatively close to the best found value of 21.1049 mA/cm^2, but the result is obtained more than 5 times as fast. Therefore, this is a good result when the focus is on finding a relatively good solution within a short computation time.

In short, the run of the genetic algorithm with population size 50 shows how a relatively good result can be obtained within a short computation time. In order to do this, stopping criterion A is used, which focuses on a short computation time. To decrease the computation time further, parallel computing can be used. In this way, the run of the genetic algorithm with population size 50 shows that a current of 20.9848 mA/cm^{2}can be found within a matter of hours.

5.5. Discussion of fixed settings

To implement the genetic algorithms that are discussed in this chapter, choices for several settings had to be made. For instance, the selection and variation operators are fixed for all implemented genetic algorithms in this thesis. Chapter 4 explains these choices. However, it is possible that the choices that are made for these settings influenced the performance of the genetic algorithm. In this section, these choices in the implementations are discussed and evaluated.

5.5.1. Fixed settings of the objective function

The first choice that is made, is how to implement the objective function. The objective function is implemented using *GenPro4* which makes an estimation of the current. The accuracy of this estimation can be changed using several variables. In Chapter 3, it is found that the most accurate objective function that can be computed within reasonable time is the objective function in which *GenPro4* uses 30 angular intervals, 1000 rays and a step size of numerical integration of 0.005 μm. In general, more accurate settings result in a longer computation time. To reduce this computation time, the objective function reuses scatter matrices that have to be determined using a simulation. As this simulation takes a lot of time and does not have to be computed anymore, this shortens the computation time. This objective function has a computation time of 21.8 seconds and results in an accuracy of 0.01 mA/cm^2 or even more accurate.

The genetic algorithms are also implemented for an objective function with 18 angular intervals, 100 rays and a step size of 0.020 μm. This is briefly discussed in Appendix A. In general, it can be seen that the solutions found using the more accurate objective function give a higher current. This does not hold for the genetic algorithms with population sizes 75 and 150. However, this could be explained by the stochastic nature of the genetic algorithm. In general, the genetic algorithm with the less accurate objective function has a shorter computation time. Therefore, there also seems to be a trade-off for the choice of the accuracy settings between finding a good solution with the genetic algorithm and having a short computation time.

One of the things that makes the more accurate objective function perform better is the fact that it reuses the scatter matrices. By doing this, the simulation is not performed every function call. Because of this, every function call makes the estimation error in the scatter matrices. Therefore, the objective function that is given to the genetic algorithm is more smooth. This makes the genetic algorithm perform better. It is also shown in Chapter 3 that the number of rays has the largest influence on increasing the accuracy of the objective function. This setting determines how many light rays are simulated to determine the scatter matrices. That means that the computation time is not expected to increase much when more rays are used in a situation in which the scatter matrices are reused. Therefore, it could be possible that a genetic algorithm performs well when the objective function that reuses scatter matrices is implemented with 1000 rays, 18 angular intervals and a step size of 0.020 μm. This could be beneficial especially if a fast result is desired. Consequently, the performance of a genetic algorithm with these accuracy settings could provide a better choice in the trade-off between a result of good quality and a short computation time. This might be a topic of future research.

5.5.2. Fixed settings of the genetic algorithm

In Chapter 4, several settings of the function ‘ga’ are fixed for the implementations of the genetic algorithms. Some settings are fixed because they are a logical choice for this problem, while other methods are fixed to due to limitations of the function ‘ga’ in Matlab, which is used to implement the genetic algorithm. It is now discussed whether the choice for these settings was right and if changing these settings might improve the results of the genetic algorithm.

In Section 4.1, it is described that a real valued representation is used for the continuous variables $x_1, ..., x_6$ and a integer valued representation is used for the discrete variables $x_7, ..., x_{10}$. It could be possible to investigate how other representations, such as binary representations, would perform. However, genetic algorithms based on binary representations tend to have problems when continuous variables are involved [12]. Therefore, the choice for this representation is most likely the right choice.

Another choice that is made is the choice of the selection operators. In the implementation, a stochastic uniform sampling method is used combined with ranked scaling, which are both explained in Subsection 2.3.2. There is no result that clearly indicates that the selection method is right for this problem or not. Therefore, it could be studied how other selection operators would perform for this optimization problem. This is not covered in this thesis, due to limitations in time, but it might be a topic of further research.

The function ‘ga’ also differs in types of selection operators compared to other genetic algorithms. As discussed in Section 4.2, the genetic algorithm in the function ‘ga’ selects individuals to be parents from the old generation. All children of the parents are then directly placed in the new generation and no selection is applied to this process. In Section 2.3, another method for this process is discussed. In this method, a selection operator is applied to a generation with population size μ in order to determine which individuals are parents. The variation operators then create λ children using these parents. After this, another selection operator is used to select μ individuals for the new generation out of the $\mu + \lambda$ individuals, which consist of the parents and their children. It is not possible to implement this using the function ‘ga’ in Matlab. It is however possible to write a custom genetic algorithm that uses this method. As there is a possibility that it leads to better results, this could be further investigated.

Two important other choices for a genetic algorithm are mutation and crossover operators. The crossover operator that is used is scattered crossover. It is difficult to asses the performance of the crossover operator as the direct effect cannot be captured in plots as in Figure 5.9, Figure 5.10 and Figure 5.11. However, the crossover operators make new solutions by switching the entries of the parents. If both parents are good solutions it might be possible that their offspring are even better solutions. In this way, the crossover operators helps the genetic algorithm converge to an optimum. The genetic algorithm seems to converge well towards optima, and therefore, the crossover function is likely to perform well. Furthermore, as there is no direct evidence that this operator does not perform well, this crossover operator is probably a good choice for this optimization problem.

The mutation operator that is used is an adaptive Gaussian operator, which is discussed in Section 4.3.

The mutations have as main goal to make the algorithm investigate new parts of the optimization domain. This is done as a mutation causes an individual in the population to be different than the population. As this region of the domain is most likely not taken into account by the genetic algorithm at that moment, the genetic algorithm investigates new parts of the domain in this way. If the population is converging towards a local optimum, a mutation could find a good result other than the local optimum. In this way, a genetic algorithm has the property to escape local optima.

The code that performs the mutations for the discrete variables is made inaccessible by Matlab. Figure 5.4 and Figure 5.6 show that the standard deviation does not increase after it has decreased to 0 for a population. This means that the mutations do not introduce any differences within the population after this point. This could be the case if the mutation operators do not change the discrete variables. This would be a situation that decreases the performance of the genetic algorithms.

In general, the genetic algorithm converges to local optima for most implementations and not to a global optimum. This could be caused by the fact that the mutations are not large enough and this would mean that the mutations should be larger. This could be don by implementing another mutation operator. However, the function 'ga' does not provide the option to use another type of mutations for bounded mixed-integer optimization problems. Therefore, a genetic algorithm must be made in order to implement another mutation function. This is a possible topic for further research.

However, it would also be a possibility to change the scale and shrink parameters of the current mutation operator. The scale parameter that is used for this thesis, starts at the maximal value that are possible with the lower and upper bound of the optimization problem. Therefore, changing this parameter likely does not make the mutations larger. Using another shrink parameter on the other hand could increase the size of the mutations. This cannot be studied in this thesis due to the limited time available, but it could be studied in further research.

The last setting that is fixed for the genetic algorithms in Chapter 4 is the creation method. For this method, it is important that it creates individuals that cover a part of the domain that is as large as possible. This is discussed in Section 4.4. The creation method that is used is uniform creation, which means that the individuals are created by performing drawings from the uniform distribution. Figure 5.9, Figure 5.10 and Figure 5.11 show the spread of the population for the variables x_3, x_4 and x_6. The points of the first generation seems to be quite evenly spread over the possible values for these variables. It should however be noted that there seem to be only a few points with lower values of x_4 in Figure 5.11a. This is most likely due to the fact that the population size of this genetic algorithm is only 25. This means that only 25 points were drawn to cover the whole domain. With a small number of points it is likely that there are less points in certain parts of the domain. However, other creation methods also need to use a small number of points and are thus also likely to leave parts of the domain untouched. Therefore, it can be concluded that the choice for this creation method is most likely not a bad choice.

In general, some settings of the algorithm seem to perform well for this optimization problem. Other settings on the other hand could possibly be changed to improve the performance of the genetic algorithm. The influence of these settings can be researched in further studies. These are discussed in Chapter 7. However, the influence of these settings does not seem to disturb the performance of the genetic algorithms heavily. The genetic algorithms still seemed to perform well as discussed in earlier sections. Therefore, the algorithms that are implemented could still be used to study the influence of the population size, the stopping criteria and the use of parallel computing.

6 Conclusion

The goal of this thesis was to find how genetic algorithms could best be implemented to optimize the tandem solar cell. In this thesis, genetic algorithms were applied to this optimization problem with different settings. Due to the high computation time of the objective function, making choices for the settings of the genetic algorithm often come down to a trade-off between finding a good solution and having a short computation time. In this thesis, several important parameters were found that influence this trade-off.

Firstly, the genetic algorithm is an optimization algorithm and these perform better for a smooth objective function. Therefore, it is important that the settings used by *GenPro4* to calculate the objective function are chosen to have a high accuracy as this makes the function smoother. However, a higher accuracy often comes at the cost of a longer computation time of the objective function. When the settings of *GenPro4* are chosen to reuse scatter matrices, the computation time can be reduced. In this way, it only uses one simulation for all function calls, and therefore, the function calls have a similar error. This makes the function value even more smooth. A good choice of the accuracy settings of *GenPro4* is 30 angular intervals, 1000 rays and a step size of 0.005 μm when the scatter matrices are reused. This is a good choice since it provides a good accuracy of 0.01 mA/cm^2 for a relatively good computation time of 21.8 seconds for a single function evaluation.

The genetic algorithm itself also has settings which should be chosen in order to tune the performance in the trade-off between finding a good solution and having a short computation time. One of these settings is the stopping criterion. The stopping criterion of the function 'ga' terminates the genetic algorithm if the average increase in the highest function value over the last M generations is less than ϵ. Two well performing stopping criteria are found. Stopping criterion A could be used when the computation time should be as short as possible, which has settings of $M = 6$ and $\epsilon = 10^{-4}$. The second pair of parameters is $M = 12$ and $\epsilon = 10^{-3}$ and corresponds to stopping criterion B. This stopping criterion could best be used when a final solution resulting a current as high as possible is desired. However, genetic algorithms with a stopping criterion with these parameters are more likely to have a long computation time.

Another setting of the genetic algorithm that influences both the computation time and quality of the found solution is the population size. It can be concluded that a large population size results in a better chance of finding a good solution. On the other hand, it also increases the computation time significantly. The results of the genetic algorithms that were implemented show that a population size between 50 and 100 performs well in finding relatively good solutions in a short computation time. If the focus should be on finding a good solution only, population sizes between 100 and 150 seem to perform better. However, these also result in a longer computation time.

A third setting of interest is the use of parallel computing. Parallel computing can be implemented within *GenPro4* and within the genetic algorithm itself. It can be concluded that using parallel computing reduces the computation time greatly. The greatest time reduction is obtained when parallel computing is applied within the genetic algorithm itself. The computation time of the objective function can even be reduced from 21.8 seconds to 5.0 seconds on a typical 4 core CPU. This makes it possible to have more accurate results within reasonable time.

In general, it can be concluded that the function 'ga' seems to perform well in optimizing the genetic algorithm, although its options for mixed-integer problems are limited. In all implementations, it is best to apply parallel computing within the genetic algorithm to reduce the computation time. It is also best to compute the objective function with 30 angular intervals, 1000 rays and a step size of 0.005 μm and to reuse the scatter matrices. If a fast result is desired, the genetic algorithm can best be implemented with a population size between 50 and 100 and stopping criterion B, which has parameters $M = 6$ and $\epsilon = 10^{-4}$. If, on the other hand, a high quality of the solution is desired, it is best to use a population size between 100 and 150 in combination with stopping criterion A, which has parameters $M = 12$ and $\epsilon = 10^{-3}$.

7 Recommendations

In this thesis, the application of genetic algorithms on the optimization of tandem solar cells is studied. In this study, some problems occurred which could not be considered within the available time span. Therefore, these problems give rise to recommendations on further research topics.

The first and most important recommendation is to repeat the runs of the genetic algorithms multiple times. Some conclusions about the parameters had to be less specific due to stochastic nature of the genetic algorithms combined with the fact that they were only executed once. When the genetic algorithms are run multiple times, it will be possible to draw better conclusions.

In this thesis, the objective function is implemented with the accurate settings that are found in Section 3. In this way, the objective function is more smooth and the final result is more accurate. As discussed in Section 3.3 and Section 5.5, it might be better to use different settings in a situation in which a short computation time is preferred. It would for instance be possible to reuse the scatter matrices and use 1000 rays in the simulation, while the other accuracy settings are the default settings. In this way, a smooth function can be obtained with a relatively high accuracy. This objective function can be calculated in a significantly shorter time. Therefore, this might be a topic of further research that results in new insights in the trade-off between a high quality of the results and a short computation time.

The genetic algorithm was implemented with the Matlab function 'ga'. This provides a standard structure of the genetic algorithm, but it also gives limitations. Due to the fact that the optimization problem was a bounded mixed-integer problem, several settings could not be changed, such as the mutations, crossover and selection operators. In order to investigate the influence of such settings, the genetic algorithm has to be constructed from scratch. How this should be done could be a topic for further research.

The mutation operators of the function 'ga' could not be changed, but they had parameters which could be tuned to control the size of the mutations. As many genetic algorithms converged to a local optimum, a larger size of mutations could be beneficial. A topic for future research could be to study the influence of these parameters on the performance of the genetic algorithm.

For the genetic algorithms in this thesis, a proportional selection operator with ranked scaling was used. In Section 2.3, it was described that scaling of the fitness function is an important tool to control the selection pressure. Although it was not directly visible in the results of the genetic algorithms, it could be the case that the selection pressure was too low or too high. For further research, it would be possible to study the influence of the scaling of the fitness function. This can for instance be done by determining the takeover time for several combinations of settings and it would provide more insight in the influence of the selection operator on the performance of the genetic algorithm. The takeover time can be determined in an implementation in which the genetic algorithms are restricted to just making copies. The takeover time then is how many generations a genetic algorithm needs to fill the whole generations with only copies the best individual.

In this thesis, the effect of parallel computing on the computation time of the genetic algorithm is studied. This resulted in the conclusion that parallel computing decreases the computation time significantly. However, better tools for parallel computing exist such as the MATLAB Parallel Server [23]. This provides the possibility to run the code online on more processors, which might decrease the computation time even more. Therefore, the last recommendation for future research is to study the possibilities of parallel servers and their influences on the computation time.

Bibliography

[1] International Renewable Energy Agency. Perspectives for the energy transition. Technical report, International Renewable Energy Agency, 2017.

[2] Lujean Ahmad, Navid Khordehgah, Jurgita Malinauskaite, and Hussam Jouhara. Recent advances and applications of solar photovoltaics and thermal technologies. *Energy*, 207:118254, 2020. ISSN 0360-5442. doi: https://doi.org/10.1016/j.energy.2020.118254. URL `https://www.sciencedirect.com/science/article/pii/S036054422031361X`.

[3] Steve Albrecht, Michael Saliba, Juan-Pablo Correa-Baena, Klaus Jäger, Lars Korte, Anders Hagfeldt, Michael Grätzel, and Bernd Rech. Towards optical optimization of planar monolithic perovskite/silicon-heterojunction tandem solar cells. *Journal of Optics*, 18(6):064012, may 2016. doi: 10.1088/2040-8978/18/6/064012. URL `https://doi.org/10.1088/2040-8978/18/6/064012`.

[4] Aly Shahzada P. Hossain Mohammad I. El-Mellouhi Fedwa Tabet Nouar Alharbi Fahhad H. Baloch, Ahmer A. B. Full space device optimization for solar cells. *Sci Rep*, 7:11984, 2017. ISSN 2045-2322. doi: https://doi.org/10.1038/s41598-017-12158-0.

[5] C.B.Honsberg and S.G.Bowden. Photovoltaics education website, 2019. URL `www.pveducation.org`.

[6] Zehor Allam Chahrazad Boudaoud, Abdelkader Hamdoune. Simulation and optimization of a tandem solar cell based on ingan. *Mathematics and Computers in Simulation*, 167:194–201, 2020. ISSN 0378-4754. doi: https://doi.org/10.1016/j.matcom.2018.09.007.

[7] Duote Chen, Phillip Manley, Philipp Tockhorn, David Eisenhauer, Grit Köppel, Martin Hammerschmidt, Sven Burger, Steve Albrecht, Christiane Becker, and Klaus Jäger. Nanophotonic light management for perovskite–silicon tandem solar cells. *Journal of Photonics for Energy*, 8(2):1 – 13, 2018. doi: 10.1117/1.JPE.8.022601. URL `https://doi.org/10.1117/1.JPE.8.022601`.

[8] Richard Corkish. Solar cells. In Cutler J. Cleveland, editor, *Encyclopedia of Energy*, pages 545–557. Elsevier, New York, 2004. ISBN 978-0-12-176480-7. doi: https://doi.org/10.1016/B0-12-176480-X/00328-4. URL `https://www.sciencedirect.com/science/article/pii/B012176480X003284`.

[9] Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2):311–338, 2000. ISSN 0045-7825. doi: https://doi.org/10.1016/S0045-7825(99)00389-8. URL `https://www.sciencedirect.com/science/article/pii/S0045782599003898`.

[10] Kusum Deep, Krishna Pratap Singh, M.L. Kansal, and C. Mohan. A real coded genetic algorithm for solving integer and mixed integer optimization problems. *Applied Mathematics and Computation*, 212(2):505–518, 2009. ISSN 0096-3003. doi: https://doi.org/10.1016/j.amc.2009.02.044. URL `https://www.sciencedirect.com/science/article/pii/S0096300309001830`.

[11] Miha Filipič, Philipp Löper, Bjoern Niesen, Stefaan De Wolf, Janez Krč, Christophe Ballif, and Marko Topič. Ch3nh3pbi3 perovskite / silicon tandem solar cells: characterization based optical simulations. *Opt. Express*, 23(7):A263–A278, Apr 2015. doi: 10.1364/OE.23.00A263. URL `http://www.opticsexpress.org/abstract.cfm?URI=oe-23-7-A263`.

[12] Lozano M. Verdegay J.L. Herrera, F. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12:265–319, 1998. ISSN 1573-7462. doi: https://doi.org/10.1023/A:1006504901164.

[13] Manuela Hooghwerff. Optimization algorithms for improving the effciency of a tandem solar cell, July 2020.

[14] Abid Hussain, Yousaf shad Muhammad, and Nauman Sajid. Performance evaluation of best-worst selection criteria for genetic algorithm. *Mathematics and Computer Science*, 2:89–97, 11 2017. doi: 10.11648/j.mcs.20170206.12.

[15] National Human Genome Research Institute. Homologous recombination. URL `https://www.genome.gov/genetics-glossary/homologous-recombination`.

[16] Klaus Jäger, Lars Korte, Bernd Rech, and Steve Albrecht. Numerical optical optimization of monolithic planar perovskite-silicon tandem solar cells with regular and inverted device architectures. *Opt. Express*, 25(12):A473–A482, Jun 2017. doi: 10.1364/OE.25.00A473. URL `http://www.opticsexpress.org/abstract.cfm?URI=oe-25-12-A473`.

[17] Patrick Siarry Eric Taillard Johann Dréo, Alain Pétrowski. *Metaheuristics for Hard Optimization*. Springer, Berlin, Heidelberg, 2016. ISBN 978-3-540-23022-9. doi: https://doi.org/10.1007/3-540-30966-7.

[18] Xue Lingyun, Sun Lefei, Huang Wei, and Jiang Cong. Solar cells parameter extraction using a hybrid genetic algorithm. In *2011 Third International Conference on Measuring Technology and Mechatronics Automation*, volume 3, pages 306–309, 2011. doi: 10.1109/ICMTMA.2011.647.

[19] Glenn Muller Reinoud Segers Maria José Linders, André Meurink. Hernieuwbare energie in nederland 2019. Technical report, CBS, 2020.

[20] MathWorks. Genetic algorithm options, . URL `https://nl.mathworks.com/help/gads/genetic-algorithm-options.html`.

[21] MathWorks. Mixed integer ga optimization, . URL `https://www.mathworks.com/help/gads/mixed-integer-optimization.html`.

[22] MathWorks. How the genetic algorithm works, . URL `https://nl.mathworks.com/help/gads/how-the-genetic-algorithm-works.html`.

[23] MathWorks. Matlab parallel server, . URL `https://www.mathworks.com/products/matlab-parallel-server.html`.

[24] MathWorks. Fitness scaling, . URL `https://nl.mathworks.com/help/gads/fitness-scaling.html`.

[25] Takashi Suezaki Gensuke Koizumi Kenji Yamamoto Rudi Santbergen, Tomomi Meguro and Miro Zeman. Genpro4 optical model for solar cell simulation and its application to multijunction solar cells. *IEEE Journal of Photovoltaics*, 7(3):919–926, 2017. ISSN 2156-3381. doi: https://doi.org/10.1109/JPHOTOV.2017.2669640.

[26] Masashi Murata Takashi Minemoto. Theoretical analysis on effect of band offsets in perovskite solar cells. *Solar Energy Materials and Solar Cells*, 133:8–14, 2015. ISSN 0927-0248. doi: https://doi.org/10.1016/j.solmat.2014.10.036.

Appendices

A Genetic algorithm with inaccurate objective function

Before genetic algorithms were implemented as described in Chapter 5 with the objective function found in Chapter 3, genetic algorithms were applied to different objective function. This objective function was calculated using the default accuracy settings of *GenPro4*. The results of this implementation are discussed briefly in this appendix. The code of this objective function and the implemented genetic algorithms can be found on `https://github.com/KoenvanArem/BEP-Genetic-Algorithms-on-Solar-Cells`.

The objective function that is implemented using the default accuracy settings of *GenPro4*, which means that it uses 18 angular intervals, 100 rays and a step size of 0.020 μm. As found in Chapter 3, the results of an objective function using these settings follow a normal distribution with a mean of 20.9404 mA/cm^{2}and a standard deviation of 0.0295 mA/cm^2. These values are estimates of the current. As the standard deviation is high, the estimates are likely to be less accurate as discussed in Chapter 3. This makes the results of the genetic algorithm less accurate.

On top of that, this implementation of the objective function does not reuse the scatter matrices in order to reduce the computation time. This results in a computation time of one function evaluation of 18.6 seconds, which could be 3.9 seconds if the scatter matrices would be reused. Not only does the choice of not reusing the scatter matrices give a longer computation time, it also makes the objective function less smooth. This is caused by the fact that the simulation to determine the scatter matrices is performed every function call, which gives a different estimation error for every function call. Because the error is not constant, the objective function is likely to make jumps due to the different errors. Therefore, the objective function is less smooth. Although the genetic algorithm is a robust optimization method, it still performs better for a smooth objective function.

The genetic algorithms are implemented using the same representation, selection operator, variation operators and creation method as discussed in Chapter 4. It is also implemented for different population sizes. On top of that, a stopping criterion is used that stops the genetic algorithm if the average improvement in the best value of the last 50 generations is less than 0.01 mA/cm^2. This means that the algorithms also stop after 51 generations. The results are shown in Figure A.1 and Table A.1.

Population Size	Highest found value	Number of function evaluations	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
25	20.7179	1300	0.1139	0.0582	0.2019	0.3748	0.3459	0.4325	2	1	1	1
50	20.9663	2600	0.1142	0.0533	0.3909	0.4011	0.1391	0.0704	2	1	1	1
75	20.7189	3900	0.1106	0.0592	0.3364	0.3713	0.3295	0.2726	2	1	1	1
100	21.0170	5200	0.1065	0.0573	0.2015	0.3933	0.3476	0.0693	2	1	1	1
125	21.0241	6500	0.0981	0.0557	0.1594	0.3772	0.1830	0.0776	2	1	1	1
150	21.0374	7800	0.1001	0.0546	0.1537	0.3785	0.1859	0.0785	2	1	1	1

Table A.1: The results of the genetic algorithm applied to a less accurate objective function for each population size.

Figure A.1 shows that the genetic algorithm tend to perform not that bad at first sight compared to the results in Section 5.1. For instance, the highest current found is a current of 21.037 mA/cm^2. However, the estimates of the current are less accurate. As the standard deviation of the estimates is 0.0295 mA/cm^2, around 2% of the estimates will be 0.0590 mA/cm^2 higher than the mean. If these overestimation occur for an estimation

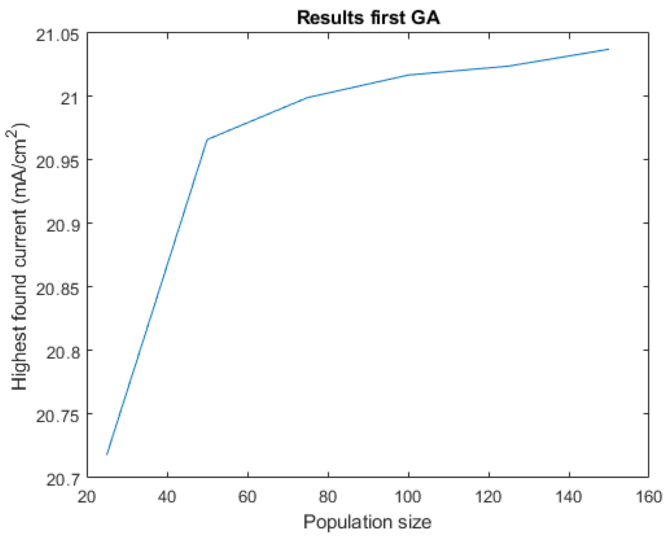


Figure A.1: The optimal value of the genetic algorithm applied to a less accurate objective function plotted against the population size.

of a solution near to a local optimum, this individual could get a too high current. As hundreds of function evaluations are performed, such overestimates are likely to occur. Therefore, the genetic algorithms are likely to be drawn towards local optima. On top of that, the results of the genetic algorithms are likely to be overestimation of the real current corresponding to this solution.

The influence of one individual is less when the population size is larger. Therefore, genetic algorithms are less likely to be drawn towards local optima in a situation as just described. Therefore, genetic algorithms with a larger population still perform relatively well. On the other hand, the final solution is still likely to be an overestimation. Especially since the genetic algorithms with larger population sizes performed more function evaluations. This means that the results of these genetic algorithms are less good than they seem.

In general, it can be said that the genetic algorithms implemented with this less accurate objective function give rise to some problems. Due to these problems, the accuracy of the objective function is studied in Chapter 3.