Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Optimization of the efficiency of tandem solar cells using simulated annealing

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

Pim van de Lest
4483642

Delft, Nederland
August 2021

TU Delft

BSc thesis APPLIED MATHEMATICS

"Optimization of the efficiency of tandem solar cells using simulated annealing"

Pim van de Lest

Delft University of Technology

Supervisor

Dr.ir. J.T. van Essen

Dr.ir. R. Santbergen

Other committee members

Dr. N.V. Budko

August, 2021

Delft

Abstract

In this thesis, a simulated annealing algorithm is implemented to optimize the efficiency of a tandem solar cell. The efficiency of a tandem solar cell is approximated by the current of the tandem solar cell. This current is determined by the simulation program GenPro4. Firstly, the general concepts of simulated annealing is described. After understanding these concepts, a simulated annealing algorithm is implemented for our specific problem. This algorithm includes the handling of discrete variables which describe the structure of the solar cell. The results contain multiple variants of the newly implemented simulated annealing algorithm, which differ in the used temperature schedule. Finally, a conclusion is made that a temperature schedule with a slower convergence gives the best results. However, one should consider a slightly faster convergence whenever there is a need to reduce the computational time. Moreover, one should write their own implementation of a simulated annealing algorithm when dealing with discrete variables, thus not use the provided function of MATLAB.

Preface

We have here, the bachelor thesis "Optimization of the efficiency of tandem solar cells using simulated annealing" which is a study on the implementation of a simulated annealing algorithm to optimize the efficiency of a tandem solar cell. This project is written as graduation project for the study Applied Mathematics in order to receive the Bachelor of Science at Delft University of Technology (TU Delft). This project took place from May to August 2021.

This project was chosen after my interests in the area of optimization in mathematics. This research is succession of other research on the use of optimization algorithms to make tandem solar cells more efficient. My supervisors, Rudi Santbergen and Theresia van Essen, provided a clear goal for me to achieve. Conducting this research was a long but interesting experience, and discussing my new ideas with my supervisors gave much gratification. The situation at the moment is not ideal due to COVID-19, however, the communication with my supervisors always went smoothly and nicely.

I would like to thank my supervisors for their moral support during the meetings and their guidance during the whole project. They gave good input, which not only helped me in the research aspect, but also helped me a lot in constructing this thesis with their comments in terms of academic knowledge. Finally, I would also like to thank my friends and family, for their support and interest in my project.

Contents

1 Introduction

Sustainable energy is a trending topic, much research has been done or is ongoing in the area of sustainability. These studies are mainly about the efficiency when generating energy. Solar energy is one of such areas in which much research is ongoing. In solar energy, the efficiency takes mainly place in the solar cells. There is a demand for the development of better and more efficient solar cells. This demand led to the development of a new and better solar cell, namely a tandem solar cell. A regular solar cell consists of multiple layers of different materials. This new and better tandem solar cell has two absorb layers instead of one, one perovskite and one silicon absorb layer. In a solar cell, the efficiency is dependent on various factors, namely, the choice of materials, design parameters, such as thickness of layers and coatings and the fabrication [8]. There are computer models that simulate an optic model of solar cells. These models have the materials and design parameters as input, and will calculate the efficiency with these inputs. To assemble these inputs, optimization algorithms are used to approximate the optimal matching of these dependencies.

1.1 Problem Definition

The tandem solar cell we are considering consists of a silicon substrate layer of a fixed thickness of 200 μm. On this layer, there are approximately 10 other layers of different materials installed. All of these layers have a significantly smaller thickness, which lies between 5 and 500 nm. The limited accuracy of the manufacturing methods can result in deviation of approximately $\pm 10\%$ from the desired layer thickness. In order to make the solar cell most efficient, the optimal combination of layer material with the according thickness has to be found. A change of material, or the smallest change in layer thickness can lead to a big change in the efficiency. The calculation of the efficiency is done in a 10-dimensional solution space. In this space, there are 4 of the 10 variables that determine the structure of the solar cell, whereas the other 6 determine the thickness of the layers in the solar cell. Therefore, there are multiple different structures possible, that each have their own optimal thickness of layers. One can imagine that there are a lot of different possibilities, and each of those possibilities can be calculated. All these calculations have a relatively long calculation time, hence finding an optimal solution with trial and error is not doable. Here is where optimization algorithms are coming into the picture. In this study, we use the simulated annealing optimization algorithm in order to save a considerable amount of computational time. The simulated annealing algorithm helps us to approximate the highest possible current, and thus the highest efficiency of the tandem solar cell.
This problem can be defined as an optimization problem, more specific a combinatorial optimization problem. In such problem, we search for an optimum object in a finite collection of objects [12]. In our problem, the highest efficiency is the optimum object, and the materials, structure and layer thickness are the finite collection. GenPro4 is a simulation tool that determines the current of the solar cell [18]. It simulates the tandem solar cell as a multi layer system and it is an optical model for simulations of solar cells. This function is based on another method, namely the extended net-radiation method, in which ray optics and wave optics are combined in a computationally efficient way. The output of GenPro4 consists of the reflectance, absorptance of each layer and the

transmittance as a function of wavelength. Another important note is that the result of GenPro4 can vary for the same input values. This happens due to the random nature of the simulation itself, which is caused by the use of Monte-Carlo simulation. Combating this randomness will not only provide a more accurate result, it also makes the algorithm run more rapid. More details on GenPro4 can be found in [18].
In this study, we are given some fixed materials, of which some have a fixed thickness. The others may vary in thickness considering certain constraints, moreover, some layer materials may vary in material and thickness. The aim of the study is to search the answer to the question: Can a simulated annealing algorithm be implemented with the use of MATLAB in order to determine which materials, and their according thickness, and structures contribute to a close to optimal current?

1.2 Structure of a (Tandem) Solar Cell

It is important to understand what the structure of a tandem solar cell is, such that we recognize what we can optimize within in a solar cell. In order to understand the structure of a tandem solar cell, we firstly consider the structure of a general solar cell. The general device structure of a solar cell can be seen in Figure 1. There are 3 main layers in this solar cell: layer 1 is the Electron Transport Material (ETM), layer 2 is the absorber layer and layer 3 is the Hole Transport Material (HTM). Besides these layers, there are also two contact points. A layer can consist of several materials, and for each layer, there exists an optimal matching of materials such that the highest efficiency is reached.

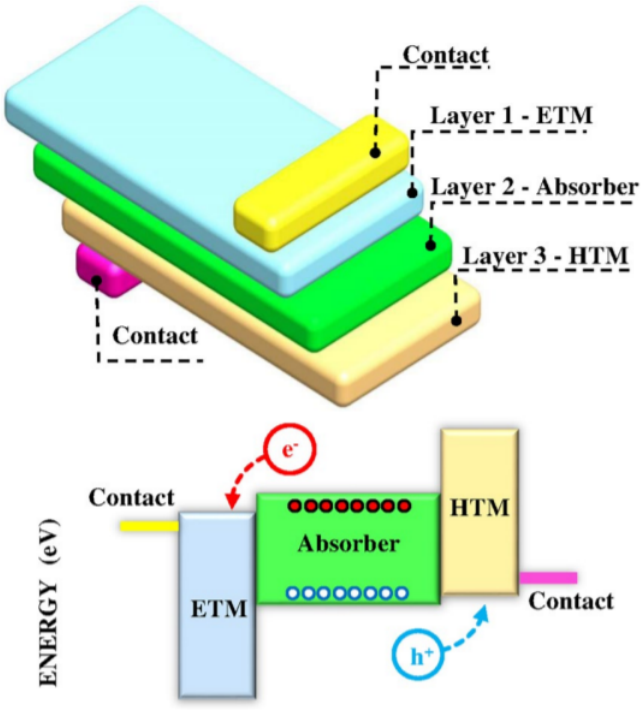

Figure 1: Device structure of general solar cells [3]

In order to find this matching, we first have to explain what a tandem solar cell consist of in terms of layers, and why certain optimization factors are important to be taken

into account. Firstly, the structure of tandem solar cell, a tandem solar cell has two absorber layers instead of one as in the regular solar cells. We consider that the tandem solar cell has two areas, the area of the perovskite layer, and the area of the silicon layer. We refer to these, respectively, as bottom and top half of the cell. In our problem, the layer thickness for these absorber layers are fixed. Now, we shortly explain the difference between 'thin' and 'thick' layers. In order to describe this importance, interference has to be described. Whenever the coherence length of incident light on a layer is greater than the layers optical thickness, an observable interference is caused, these layers are called coherent layers. Much thicker layers for which this does not hold are called incoherent layers [17]. GenPro4 treats these thin coherent layers as 'coating', the two thick incoherent layers as just 'layers'. In GenPro4, 'layers' are considered as incoherently and do not produce interference whereas 'coatings' are considered coherent and do produce interference. A small change in thickness in these coatings can lead to a great change of the current, hence, optimizing these coatings has to be done very carefully. GenPro4 uses equation (1) to calculate the current, around this equation, a brief explanation on it is given. Moreover, GenPro4 only calculates the current, hence not the voltage, thus we cannot conclude the efficiency. We already mentioned that the tandem solar cell consists of two absorbers, however, both of these absorbers have their own current that has to be calculated. The goal is to maximize both of these currents, nonetheless, these values of the currents have to be close to each other. This has to be the case because the maximum current of the solar cell is the minimum of both the currents of the bottom and top cell, more on this is described around equation (2).
Having explained all of this, we describe our problem in more detail. In the problem, we vary the thickness of six of these coatings, remember, in these variables small changes of $10nm$ can make a notable difference in the current due to interference. These coatings are continuous variables in our problem, and can vary between $40nm$ and $500nm$. Additionally, our problem has four discrete variables, two of these are on the choice of material in the ETM and HTM. The other two discrete variables are structural variables, one is to add an extra coating, namely, an Anti-Reflective(AR)-coating and the other one is the option to add a pyramid structure on the interfaces, which are the separation between layers. There are two possible interfaces on which pyramid structures can be applied.
There are three interface models included in GenPro4, these are models for interfaces with no, small and large textures. With these models, most types of silicon solar cells can be simulated accurately.

1.3 Thesis Outline

The goal of this thesis is to use the simulated annealing optimization algorithm, determine a high current of a tandem solar cell with a limited number of function evaluations. First, in Chapter 2, a previous literature is reviewed and discussed. Furthermore, in order to understand the simulated annealing algorithm more thoroughly, a deep explanation of the origins and the works of simulated annealing is given. Then, in Chapter 3, the methods used in order to implement a simulated annealing algorithm in MATLAB that fits our problem are provided. Next, the results are displayed and analysed for a case study in Chapter 4. Lastly, the overall results and conclusion on the methods used are given in Chapter 5, with an additional recommendation on Chapter 6.

2 Literature Review

In this chapter, we describe previous research on optimization methods and algorithms in order to achieve the highest efficiency in tandem solar cells. The analysis of previous optimization methods will show us that it is important to implement a simulated annealing algorithm for this problem. Furthermore, a brief explanation is given on what optimization methods exactly are with a more detailed view on the optimization method we are going to study, simulated annealing.

2.1 Previous Research

Jäger et al. [2] uses simulated annealing and the GenPro4 program in MATLAB. In the introduction of the article, they explain how the optimization of a tandem solar cell is improving one step at the time, and that they are building on previous research. They state that, optical optimization based on trying out different material stacks which are relevant and layer thickness combination are promising to increase the efficiency in tandem solar cells. The building piece for efficiency that this study aims to provide is to extend the previous optical optimization of the top-cell layer thickness of a perovskite-SHJ tandem solar cell. This is the main difference in the research that we are conducting, since we consider a change in other layers as well.
Now, to extend this optimization, in the study, they consider multiple architectures. They compare an inverted architecture with a regular architecture. In the inverted architecture, the top and the bottom cells of the perovskite-SHJ tandem solar cells are flipped with respect to their layers. Both the top perovskite and the bottom silicon cells are flipped upside down. In Figure 2, the regular and inverted layer stack are represented.

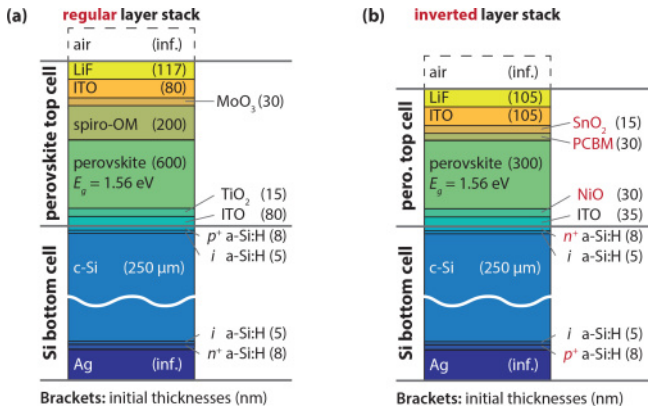

Figure 2: Regular and inverted layer stack architecture types [2].

As mentioned, the MATLAB program GenPro4 was used to calculate the absorption profiles, a discussion on GenPro4 will be given later on in more detail. These calculations are done for both the regular layer stack, and the inverted layer stack. In the domain of the simulation, the layer stack is enclosed by incoherent infinitely-thick air and silver layers. The high thickness of the silicon layer is more than the coherence length of sunlight, hence this layer is treated incoherently. Since the other layers are considerably more thin, they can be treated coherently. Furthermore, they consider that the interfaces are optically

flat, hence they do not scatter any incoming light. Equation (1) is used for the calculation of the absorption profiles of each layers current densities.

$$J_{ph,i} = -e \int_{350nm}^{1200nm} A_i(\lambda)\Phi(\lambda)d\lambda. \tag{1}$$

Here is e the elementary charge, A_i the absorption spectrum of the i-th layer, λ denotes the wavelength and Φ is the photon flux according to the AM1.5G solar spectrum [1]. The absorption spectra of the perovskite and silicon layer are used to calculate the reachable optimum for the current densities in their area. The current densities of the other layers are considered losses due to parasitic absorption. Now, for the 7 layers around the perovskite top cell, the layer thickness optimization is conducted for both the architectures. This is done with the most basic implementation of the simulated annealing algorithm. With this algorithm, they want to achieve a maximum of the current density, i.e.,

$$\min\left(J_{ph}^{top}, J_{ph}^{bottom}\right) \rightarrow \max. \tag{2}$$

During this optimization, the thicknesses of the bottom cell were kept constant. A good remark that they made is that the simulated annealing algorithm did not find *the* global optimum. This is true due to the pseudo-random nature of the algorithm. However, it finds solutions in its close proximity and is more accurate whenever the duration extends, in other words, the simulated annealing algorithm converges to the global optimum [4]. This only happens in theory, since the the running time of the algorithm is finite.
Their result showed the following for the two different architectures. In the regular stack layer, they achieved an optimum of $J_{ph} = 17.6$ mA/cm^2, and the inverted stack layer had a optimum at $J_{ph} = 19.0$ mA/cm^2. Furthermore, they also concluded that the full performance potential of perovskite-silicon tandem solar cells can only be exploited if the devices are optimised from an optical perspective. The result they have presented is of an optimization in layer thickness for planar inverted architecture. The parasitic absorbance in the front layers of the top cell is greatly decreased since the support layers of such an inverse architecture can be manufactured much more thinly. Hence, there is a potential that an additional cumulative of ≈ 2.8 mA/cm^2 can be generated in the inverse architecture in comparison with the regular one.
The most important differences this study has in comparison with the study that we now conduct are, firstly, that they consider a different type of architecture change. They consider two options, a regular and an inverted layer stack, whereas, we consider a regular or a pyramid structure. Secondly, the simulated annealing algorithm that they implemented is from the MATLAB Global Optimization Toolbox, this will be the same algorithm that we use, however, we use more of the optional inputs. Next, a small detail, their silicon layer is fixed on $250\mu m$, ours is fixed at $200\mu m$. And as last, we consider multiple materials for different coatings, with the potential addition of an extra layer.
In Baloch et al. [3], they introduce and deploy a full space material-independent optimization with the aim to increase the maximum possible conversion efficiency and focused functionalisation for any given layer by means of opto-electrical simulations. They made sure of this full-space optimization by only having essential physical constraints. There are many reports that optimize a cell design by determining the properties of the optimal matching materials for a given absorber, however, they do not span the full possible space. The full-space device design optimization in [3] is composed of two parts. The first part is the device simulation module and the second module is a numerical optimization tool.

For the first module. they use the Solar Cell Capacitance Simulator (SCAPS), which is a one dimensional device simulation. This simulation calculates the efficiency given two input vectors $\mathbf{v} \in \mathbb{R}^N$, where N is the number of parameters that need to be optimized and $\mathbf{a} \in \mathbb{R}^M$, where M is the number of fixed parameters. In the simulation, it solves the optically excited charge generation problem, transport equation, 1D Poisson's equation, and continuity equation and calculates the efficiency $\eta(\mathbf{v}, \mathbf{a})$. The second module maximizes the objective function $\eta(\mathbf{v}, \mathbf{a})$ numerically by varying $\mathbf{v}$ based on some physical constraints. This optimization is carried out by using toolboxes of MATLAB. The objective function is here the efficiency, and it is to be maximized as follows,

$$\eta_{max} = \max_{\mathbf{v}_L < \mathbf{v} < \mathbf{v}_U} \eta(\mathbf{v}, \mathbf{a}). \tag{3}$$

where η is

$$\eta = \frac{V_{OC} J_{SC} FF}{P_{in}} \tag{4}$$

Here, $\mathbf{v}_L$ and $\mathbf{v}_U$ are lower and upper bounds derived from acceptable physical ranges, V_{OC} the open circuit voltage, J_{SC} the short circuit current, FF the full factor and P_{in} the input power. In this work, they used initial guesses to start the optimization process which is iterative, with the convergence of the objective function within a tolerance of 10^{-6} as stopping criteria. The process outputs an optimized vector $\mathbf{v}$, and together with $\mathbf{a}$ it composes the optimal material data set for each layer in their design. In order to select the best suitable optimization algorithm, they compared seven different local, global and hybrid optimization methods. One last note before we can observe the results, in the study they considered two cases. The first one is a consideration of a solar cell without defects, in this case only inherent recombination properties are considered. And in the second case, several defects are introduced based on literature. For in the case without defects, the local gradient based method (one of the local optimization methods) had the fewest number of function evaluations, and obtained a nearly as good of an efficiency compared to the methods with the best efficiency, namely, 26.70%. In the case with the defects, the same holds, now with an efficiency of 23.42%. The point on which our study will be novel in comparison to this one is firstly the use of GenPro4 instead of SCAPS. The difference in this is that with the use of SCAPS an opto-electrical simulation is considered in order to achieve a high efficiency. Whereas, GenPro4 is only optic and not electrical, thus only the highest current in the solar cell is calculated. In other words, our approach is limited to optical modelling of the current, and does not include electrical modelling of the voltage. This can be seen as a downside of GenPro4. However, the optical model in GenPro4 is more advanced, since it can handle optic effects like the interference and scattering of light. This makes GenPro4 an overall better simulation. As last, they did not consider simulated annealing as one of the global optimization methods.

In El-Naggar et al. [11], a simulated annealing based approach is proposed for an optimal estimation of solar cell model parameters. This is done for different models of solar cells, namely single diode, double diode, and photovoltaic, but not for the tandem solar cell. The developed technique is used to solve a function that controls the current to voltage relationship of a solar cell. Several cases are investigated to validate the consistency of their parameter estimation. Additionally, other methods and models are used to test the outcomes. The parameters that are optimized are the photon current, diode saturation current, series resistance, shunt resistance, and diode ideality factor. We are interested in

their development of the simulated annealing algorithm. Their proposed method is a 10 step procedure, which contains the standard steps op a simulated annealing algorithm.
The accuracy of the results for the implemented simulated annealing algorithm was evaluated with the statistical root mean square error criterion. In the results, they compared simulated annealing to three other models from other literature, namely, the Newton model modified with Levenberg parameter, genetic algorithm (on the exact same problem), and a method with the Lambert W function.
They conclude that simulated annealing has the potential to be a valuable tool to estimate parameters for a system. This conclusion is made because simulated annealing relieves system modeling from the normal oversimplifying assumptions by other traditional estimation techniques.
As mentioned, in the reviewed study, they did not consider the tandem solar cell, moreover, they had a different methodology in order to achieve their results. However, the result that simulated annealing has been having its successes when used to optimize parameters in other types of solar cell, reveals how promising the algorithm can be.

2.2 Simulated Annealing

In this section, we describe the ideas and theory behind simulated annealing. Simulated annealing is an optimization method that we use in order to maximize the efficiency in a tandem solar cell. Before we describe our specific optimization method, we describe what an optimization method is and what the differences are among optimization methods.

2.2.1 Optimization Methods

In many fields of study, optimisation methods are used to find optimal solutions for a function where a certain objective function is maximised or minimised in a specific parameter space. Such as, minimizing the costs of a product or service, or in our case, maximizing the electric current in a solar cell [16]. Usually, one refers to a minimization problem, since a maximization problem on a function $g(x)$ can simply be considered as a minimization problem on the function $f(x) := -g(x)$. One can classify optimization methods into two types, global or local optimization methods. To understand why we want to use a global optimization method, we need to learn the differences between global and local methods. For this purpose, we consider the simple case that is illustrated in Figure 3. In this figure, there are three local minima of which one is a global minimum. When considering a local optimization method, having start point 1 as initial value will provide us with the minimum in the middle. With starting point 2 as the initial value, the global minimum will be reached. From this, we can conclude that it is possible with a local optimisation method to reach the global optimum.

However, whenever a domain gets more complex, i.e. more local optima, it is very unlikely that the global optimum will be reached. In contrast, a global optimization method will reach the global optimum value for more initial values than local methods. This makes a global optimization method distinguishable from local optimization, since its focuses on finding the optimum over the given set, as opposed to finding local optimum values. Finding an arbitrary local minimum is relatively straight forward by using general local optimization methods. The global minimum of a function is far more difficult to find. With analytical methods, it is very straight forward to find one local optimum. For global

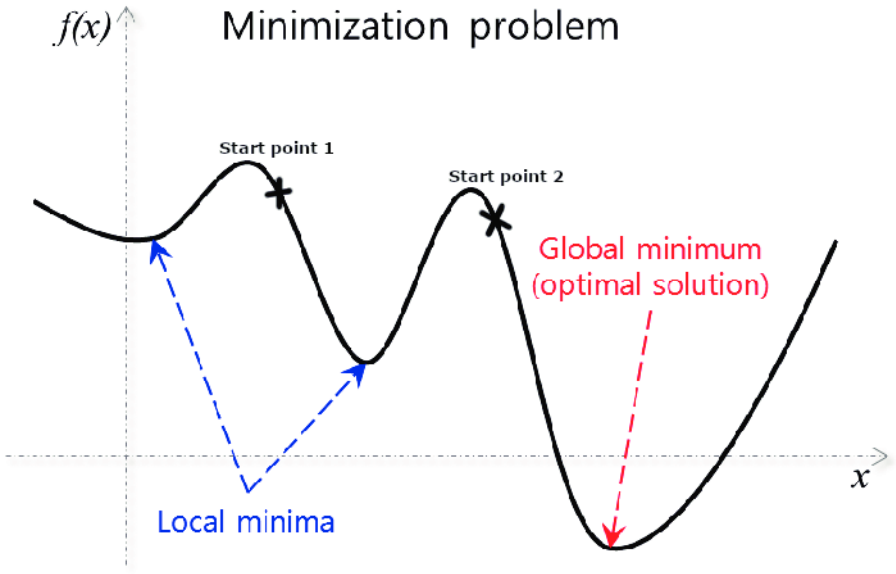

Figure 3: A graph with global and local minima [9].

optimization, analytical methods are able to find the global optimum in explicit optimal control problems, for which an analytic solution must exist for the objective. However, for our problem, there is no analytical solution for the objective function GenPro4. Thus, numerical optimization methods must be included. Finding an optimal solution with numerical methods can often be a very challenging task.
Translating the previous to our problem, we want to find the optimum value of the objective function provided by GenPro4. In our 10-dimensional parameter space, there are many local optima. If we use a local optimisation method, the probability of finding the global optimum, or a local optimum that is close enough to the global optimum, is small. Therefore, we want to use a global optimization method, where the technique of simulated annealing comes into play.

2.2.2 Simulated Annealing Algorithm

In order to get a better understanding of simulated annealing, we firstly discuss how an annealing process occurs in nature and physics, which is also referred to as real annealing. Here, physicists modify the state of a material with temperature as adjustable variable. Annealing is a technique where an optimum state can be approximated by controlling this temperature. The technique of annealing consist of heating a material to a high energy state, followed by a graduate slow cooling of the material, by maintaining at every state a temperature of adequate duration. At the start, thus high temperature, a material has many different optional states to which it can transform, which is proportional to the entropy of the material. In statistical thermodynamics, Boltzmann describes the entropy as being proportional to the logarithm of the number of possible microstates of the system that could cause the observable macrostates of the system. The Boltzmann constant (k_B), provides the rate of the proportion between these states [13].
Whenever the decrease in temperature is too fast, some faults may occur, which should be removed by local extra input. This system will lead to a stable state of the material, which corresponds to a minimum of energy. This idea of using the annealing technique

in order to deal with optimization problems in physics led towards the technique of simulated annealing.
In simulated annealing, an introduction of a control parameter is utilized, which plays the role of the temperature in the physical case. This temperature of the simulated system must be analogous to the temperature of the psychical system: it must determine the number of available states and. In the psychical system, this means that the temperature determines the number of energy states. Whereas, in the simulated system, the temperature determines the number of solutions, which are analogous to the energy states, that can be reached in the solution space. Moreover, it should result to the optimal state or solution. If the temperature is gradually and slowly lowered, as in the case of annealing, the entropy is rather the same, in simulated annealing entropy is a measure of disorder present in a system. In order to further convert this technique to its simulated version, an broadened analogy must be established between the optimization problem and the physical system. This analogy is represented in Table 1. This analogy is a comparison of simulated annealing and real annealing.
Having discussed this, we can dive deeper into the simulated annealing algorithm. Simulated annealing is a stochastic method and is based on two results of stochastic physics. Firstly, we conclude that the probability of an energy E that is given for a physical system is proportional to the Boltzmann factor: $e^{\frac{E}{k_B T}}$, where T is the absolute temperature and k_B denotes the Boltzmann constant. This is only the case whenever an entropy equilibrium is reached at a certain temperature. Whenever this is the case, the distribution at that temperature of the iteration for the energy states is the Boltzmann distribution. This Boltzmann constant will only occur in nature, in simulated annealing, we do not use the Boltzmann constant because we only need the form of the Boltzmann distribution. This distribution happens to capture the idea of accepting new worse solutions, provided that they are not extremely worse than the current solution. In theory, one can use other (similar) distributions.
Secondly, to further simulate the physical system, the Metropolis algorithm is utilized [15]. Now, with a certain given composition, the system is subjected to an elementary perturbation. Whenever the objection function decreases with this new result, it is accepted, otherwise, if it causes an increase, it can still be accepted, however with a probability $e^{\frac{-\Delta E}{T}}$, where ΔE is the change of energy between two states. To realise this condition, a uniformly distribution on the interval $[0, 1]$ is considered, and the objective function is accepted when this distribution is less than or equal to $e^{\frac{-\Delta E}{T}}$. Repeating this Metropolis rule of acceptance, a sequence of configurations is generated, which leads us to a Boltzmann distribution of the objective function at this temperature. At the initially high temperature, $e^{\frac{-\Delta E}{T}}$ is close to 1, therefore almost every move is accepted and the algorithm becomes equivalent to a simple random walk in the parameter space. This is important

Table 1: Analogy between an optimization problem and a physical system [14].

Optimization problem	**Physical system**
objective function	free energy
find a "good" configuration	find the low energy states
variables of the problem	coordinates of the particles
entropy; disorder of system	entropy; disorder of particles

because now practically any move can be accepted, which creates the possibility for the algorithm to explore the entire parameter space. Contrary, at low temperature, $e^{\frac{-\Delta E}{T}}$ is close to 0, now the majority of moves resulting in an increase of the objective function are the only ones that are going to be accepted. At a temperature in a middle state of the algorithm, the algorithm has a decent chance to accept a move that lowers the value of the objective function. This makes that the algorithm leaves a field for the system to be dragged out of a local optimum.
This previous provides knowledge on how the algorithm terminates and how it can escape some local optimal values. Knowing this, we can give a step by step representation of the simulated annealing algorithm [7].

- **Step 1** Start with an initial solution $s = S_0$, which has to be a solution inside the solution space of the objective function. Moreover, set the initial temperature T_0. Calculate the objective function (f) of the initial solution, $f(S_0)$.
- **Step 2** Set a cooling factor α and fix k to 1.
- **Step 3** For the current iteration k, loop N times through **Step 4**, after this loop, decrease the temperature T_k to the new temperature T_{k+1} according to the temperature schedule and the initiated cooling factor α. Break this loop whenever the stop conditions are satisfied, mostly either a good enough solution is found, or the algorithm has cooled down sufficiently.
- **Step 4** Pick one solution from a randomly selected neighbourhood of solution $N(s)$, and calculate the difference in the objective function between the old and the new solution.
- **Step 5** If the difference in the objective function between the old and the new solution is greater than 0, we accept it, moreover, we now compare it to the best found solution. If it is less than or equal to 0, then we compare it to a $\eta \cong Unif[0, 1]$ and accept it if $e^{\frac{-\Delta f(s)}{T}} \geq \eta$ holds.

Step 1 and 2 are initial steps and do not need any further explanation. In step 3 it is at first sight not fully clear how the decrease of temperature takes place. To elaborate a little more on this decrease, we consider multiple laws of decrease for the program of annealing. Ahead at the bullet point *"Cooling schedule"*, an elaboration of these choices for these laws will be described. In step 4, we consider a neighbourhood of solutions $N(s)$, which are the solutions around s that are reachable with the random perturbation function used to create the new solution. These random perturbations must be a size that fits the problem. For example, for a problem on a small interval, they have to be considerably small. In step 5, the overall best solution is stored, moreover, it considers the rules of acceptance, which is the principle of simulated annealing. An occasionally acceptance of an increase in the objective function of the current state enables the solution to be pulled out of a local minimum. At low temperature, the rate of acceptance becomes very low, which makes that the method becomes slowly ineffective. This is a common problem in simulated annealing.
Since we have now finished describing the steps of the algorithm and know what happens at each iteration, we can begin to think about how the code should be implemented and what should be taken into consideration when doing so. In [14], an example of simple

practical suggestions for implementing a simulated annealing algorithm is given. Furthermore, these suggestions are adjusted to make more sense when considering certain MATLAB functions.

- *Definition of the objective function*: The problem is formulated by an objective function of many variables, which are subjected to several constraints, which can have several origins. For example, they can be integrated from the objective function, and other from the parameter space and limitations of the problem itself.

- *Choice of the neighbourhood of solutions*: The choice of a neighbourhood must be precise, it cannot contain the current solution. Moreover, the calculation of the difference in objective functions must be as quick as possible.
  In MATLAB, a neighbour solution is chosen by an annealing function. One can use one of the two standard functions, or implement a custom one. With the standard ones, the new solution is generated based on the current solution and the current temperature using multivariate normal distribution. Moreover, MATLAB uses certain seeds for randomness, which makes it important to check whether or not our random neighbourhood is indeed always random. In order to make this certain, one can use the code line `seed('shuffle')` to ensure randomness.

- *Initial temperature* T_0: the initial temperature is important, whenever it is too high, the run time of the algorithm can be unnecessary long, contrary, if it is too low, the algorithm will focus too quickly on a certain area of the parameter space. A good rule of thumb when implementing the initial temperature is, that it has to be chosen in a way that the acceptance probability is close to a fixed value. In Ben-Ameur et al. [5], there is a description of several different methods to determine a fitting initial condition:

  - Method 1: set $T_0 = (\Delta f)_{max}$, where $(\Delta f)_{max}$ is the maximum difference in the objective function in the neighbourhood of solutions. This is a good indication for the upper bound of the initial temperature.
  - Method 2: calculate a number of strictly positive, hence worse, transformations of the objective function, and determine the average of the objective functions $\bar{f}$. Then, an educated guess of the initial temperature is $T_0 = -\frac{\bar{f}}{ln(x_0)}$, where x_0 is the desired initial acceptance ratio. For example, $x_0 = 0.8$ whenever we desire an acceptance of 80% at the start of the algorithm.

- *Acceptance rule of Metropolis*: Let η be a uniformly distributed random variable on the interval $[0, 1]$. Solution s is accepted if $\eta < e^{\frac{-\Delta f(s)}{T_k}}$, where T_k is the current temperature and Δf the difference between the old and current objective function. In MATLAB, a variation of the Metroplois rule of acceptance is used as default. There, a worse solution is accepted if $\eta < \frac{1}{(1+exp(\frac{\Delta f(s)}{T_k}}$ holds. One should determine which acceptance rule fits their problem the most.

- *Change in temperature stage*: happens as soon as one of the following conditions are satisfied during the temperature stages, with N_* as the number of variables of the problem, i.e., the length of the vector simulated annealing is running on:

- $a \cdot N_*$ neighbourhood solutions accepted
- $b \cdot N_*$ neighbourhood solutions attempted
  Here, a and b are integers, with $a < b$.

The MATLAB function has no options for this choice, thus, it should be implemented if used.

- *Cooling schedule*: is based on a cooling factor α which leads to a decrease of temperature that can be carried out according to multiple laws of decrease.
  - Geometrical reduction law: $T_{k+1} = \alpha \cdot T_k$. This is a widely used one, because of its simplicity.
  - Adaptive geometrical reduction law: $T_{k+1} = (T_k) \cdot T_k$, here, (T_k) is defined as in equation (5). This reduction law is potentially more effective than the previous one, since the new temperature only decreases when the new average of the objective function is less than or equal to the old average. The biggest drawback of this law is that can be very slow in practice [19].

$$(T_k) = \begin{cases} 1 & if \quad \langle U \rangle_{i+1} \leq \langle U \rangle_i \\ \alpha & if \quad \langle U \rangle_{i+1} > \langle U \rangle_i \end{cases} \tag{5}$$

    $\langle U \rangle_i$ is the average of the objective functions at the i^{th} iteration.
  - Linear reduction law: $T_{k+1} = T_k - \alpha$, which has a low entropy production rate at the beginning (high temperatures), which then grows rapidly at lower temperatures [10].

  In MATLAB, the default used function decreases every iteration according to a variant of the geometrical reduction law, where the cooling factor α get closer to 1 each step. A choice has to be made from the various laws proposed in literature on the initiation of the temperature.

- *Stop criteria*: The program can be terminated on several occasions. In the MATLAB function for simulated annealing, there are some build in options that decide the termination, which can also be altered. These are:
  - Function tolerance and max stall iterations: The function tolerance is a small value, standard $x_{tol} = 10^{-6}$, and max stall iteration is a value which is standard $x_{msi} = 500 \cdot N$, where N is the number of variables in the problem. A program is terminated whenever an average change in the value of the objective function over x_{msi} iterations is less than x_{tol}.
  - Max time: Standard on infinity, whenever the running time of the algorithm exceeds the max time, the program is terminated.
  - Max iterations or function evaluations: Program terminates whenever the maximum number of iterations or maximum number of function evaluations is exceeded.

- *Most important verification during the initial executions of the algorithm*:
  - different results of different executions should not differ considerably, moreover, they should be still close whenever:

    - different random number generators and seeds are used,
    - different initial conditions are used.

- *Parallelization of the simulated annealing algorithm*: Computation time can become a drawback. In order to reduce the computation time, a parallelization of the algorithm can help. This consists in simultaneously carrying out several calculations necessary for its realization.

- *Handling of discrete variables*: The implementation of discrete variables has to be done carefully. The convergence of the continuous variables must coincide with the method of handling the discrete variables.

3 Methodology

In this chapter, the method used to implement the simulated annealing algorithm in MATLAB to our specific problem is discussed. Firstly, a discussion on why the general simulated annealing algorithm in MATLAB does not suffice to our problem, moreover, what needs to be implemented in order for it to be. Secondly, the methods on how we have improved this general algorithm, with an explanation of the implementation of these improvements. Moreover, during the descriptions of the improved algorithm, references towards the codes are made. The pseudo code is provided in parts during the methodology.

3.1 MATLAB's General Algorithm

In order to understand why the most simple general function of simulated annealing is lacking, we first describe this function and how it operates. The MATLAB function for simulated annealing is called *simulannealbnd*. To initialize this function, an objective function and a starting solution have to be provided. Moreover, if the problem is bounded, a lower and upper bound must be given. Additionally, more options can be provided, whenever one decides to leave this empty, the default settings are used. Several settings can be provided, namely, the temperature function, acceptance function, how to determine a new solution, the stop criteria and more. Using this algorithm with some of the default MATLAB settings will not give the best results for our problem. The main reason for this, is that our problem contains some discrete variables and there is no standard function for creating a new solution in the simulated annealing algorithm in MATLAB that handles discrete variables. Moreover, plenty of other tweaking of the function has to be done in order to make the algorithm viable for our variable space.

3.2 Improvements to the Algorithm

The pseudo code of the algorithm is divided in three parts, Algorithm 1, is the description of the temperature function. Algorithm 2 describes the process of the creation of a new solution. As last, Algorithm 3 describes some of the initialization and the general structure of the algorithm, such as when to call certain functions. MATLAB gives the option to implement your own functions instead of the standard ones. Now, we briefly explain the new types of functions which have been implemented.
As acceptance function, the acceptance rule of Metropolis is implemented. The default function of MATLAB uses a variation of this rule. In the default function, a worse solution has a maximum chance of 50 percent to be accepted for every temperature. In result, this should not differ with the general Metropolis rule. However, in our method the initial temperature has to be decided on forehand. At high temperature, an acceptance of 80% is requested, this because we practically want to allow almost any movement in the solution. Thus, we choose the Metropolis rule of acceptance, even though the end results of both can be close to optimal.

```
1 Function Temperature_Fcn is
2 |  if N iterations have passed then
3 |  |  Lower the temperature as follows: T_{k+1} = αT_k;
4 |  end
5 |  N and α depend on the current state of the algorithm
6 end
```

Algorithm 1: Temperature Function: Simulated Annealing Algorithm

The standard MATLAB temperature function is a variation of the geometrical reduction law. The temperature is decreasing every iteration, however, every iteration it decreases a little less. Even with this slower decrease, the temperature decreases too fast for our variable space. The bounds in our solution space are very small, but the standard MATLAB functions are made for general problems. The size of our solution space and the standard MATLAB functions will be a repeating conflict in our problem. Now, as stated in the literature review, the temperature should decrease every N_* iterations. Moreover, at low temperatures, the same current temperature should be maintained on even more iterations to ensure a better convergence. In the new temperature function, Algorithm 1 Line 1, the regular geometrical law of reduction is implemented with a cooling factor α. Additionally, N is dependent on the current state of the algorithm. For example, whenever the temperature is above 1, the temperature gets reduced after a certain N amount iterations, if the temperature is less than or equal to 1, the temperature should decrease slower, thus N becomes larger. Moreover, α also differs in both of these cases as described in Algorithm 1 Line 5 of the pseudo code.
Our relatively small-bounded variable space is a problem for the general functions in MATLAB that create a new solution. In the MATLAB function, a random vector is created and multiplied by the current temperature. In the general function, nearly always the addition of this resulting factor to the current solution exceeds the boundaries of our problem because the temperature is too high, or the random elements of the vector are too big. Although those exceeded values will be properly scaled with another function, it is better for the algorithm to not almost always have new solutions that exceed the boundaries. Hence, a new function to generate new solutions has been written. Note that in this function also the discrete variables are decided. In our problem, a solution is a vector with 10 variables, of which 4 are discrete. The generation of a new solution is the same idea, however, we scale the resulting addition vector down. First we consider how the continuous variables are decided. A random vector with elements which are normally distributed with the standard deviation dependent on the boundaries is created, in the general MATLAB function a standard normal distribution is used. In order to make the perturbation vector smaller, the root of the temperature is taken before we multiply it with this perturbation vector, as last, we add this new perturbation vector to the old solution. The formula is given in Algorithm 2 Line 2 of the pseudo code. Now for the discrete variables, since the function for the algorithm itself only handles continuous variables, the discrete variables are first some continuous variables which are later on rounded. The discrete variables are uniformly distributed such that after they are rounded the chance to choose any discrete variable is the same for every discrete variable. In a later stage of the algorithm, most of the discrete variables are redundant to be checked still, later on an explanation on how this problem is tackled will be provided.
For all the additional options, these are implemented in order to test the algorithm. These are for example, when and what the algorithm should plot as information and if the result

after a certain step should be compared to another optimization method. The only one worth mentioning is the stop criteria. The only stop criterion that is important to include is the *Max Stall Iterations*, which is described in the literature review. Other stop criteria can have a negative contribution to the result of the algorithm. Other stop criterion can lead to a premature stoppage of the algorithm. For example, the use of 'Max Function Evaluation' or 'Max Time' can both lead to this early stoppage. Additionally, we are not interested in a maximum objective function value stop criterion, since we search a high as possible objective function value, which is unknown. Note that some of these option could be of value while experimenting with algorithm.

1 **Function** *create_new_Solution* **is**
2 | create a random addition vector y, and set $y = \frac{y}{||y||}$ to scale it down, the new solution is created with the following formula: $x_{new} = x_{current} + y \cdot (T_k)^{\frac{1}{2}}$;
3 | **if** *Manual_Reannealing == False* **then**
4 | | Discrete variables chosen equally random;
5 | **else**
6 | | Discrete variables are chosen equally random or will be set to the current best whenever $a > b$, where a is a uniformly random distributed between 0 and 1, and b a fixed point based on the current temperature T_k;
7 | **end**
8 **end**

Algorithm 2: New Solution: Simulated Annealing Algorithm

There are still some important implementations of the algorithm that have to be mentioned. These implementations partly have some code outside of the algorithm its main function *simulannealbnd*.
The first additional implementation is the calculation of the initial temperature. The method used is described in the literature review as Method 2. In order to get a better accuracy, this method has been conducted several times, and the average of these initial temperature has been chosen as the main initial temperature.
In order to explain the second implementation, a new definition has to be explained, namely, reannealing. Normally, reannealing starts whenever the algorithm has accepted a certain number of new solutions. Reannealing raises the temperatures, which is a method to avoid being caught in a local minima. In our until now described algorithm, the discrete variables will always be chosen with the same probability. In the later parts of the algorithm, this will be counter productively. Because, the continuous variables have already been cooled down and settled in, but still every discrete variable will be tested with equal probability which leads to a lot of unnecessary testing of clearly worse solutions, since the algorithm in this state has an idea which discrete variables are the best candidates for the best discrete variables. To counter this, we introduce a manual reannealing that happens once. Our algorithm now has two parts, one before the manual reannealing, and one after. In the part before, a relatively rapid convergence takes place. Here, the discrete variables are still random with the same chance to be chosen, Algorithm 2 Line 4. This is done until the algorithm reaches a temperature of 0.05. At this time, the current best solution has the highest probability to contain the best combination of discrete variables compared to other combinations. Now, the manually reannealing occurs, we set the temperature high again, and we implement a code that gives the algorithm a probability to choose the at that time best found discrete variables, instead of some

randomly chosen variables. This probability depends on the temperature at that time, the lower the temperature, the higher the probability to choose the best found discrete variables, thus the discrete variables now converge in probability, Algorithm 2 Line 6. Note that whenever a new best solution is found, its discrete variables also become the new best found.

```
1  Run one simulation and save scatter matrices;
2  Initialize Temperature T_0 and starting solution x_0;
3  Calculate func_val(x_0);
4  while Stop conditions == False do
5  |  iteration ← iteration +1;
6  |  Call function create_new_Solution() to create x_k;
7  |  Calculate func_val(x_k);
8  |  if current solution is better then
9  |  |  Accept the solution;
10 |  |  if Manual_Reannealing == True & System cooled down sufficiently then
11 |  |  |  if accepted solution is better than best solution then
12 |  |  |  |  Calculate func_val(x_k) a number of times and take the average;
13 |  |  |  end
14 |  |  end
15 |  |  Check to update best found solution;
16 |  else
17 |  |  Metropolis rule of acceptance;
18 |  end
19 |  Call function Temperature_Fcn();
20 end
```

Algorithm 3: Initialization & Structure: Simulated Annealing Algorithm

As last, a method to handle the randomness of the GenPro4 simulation must be implemented. In order to tackle this problem, whenever the algorithm is sufficiently cooled down, and finds a new best solution, this new best solution is tested several times and the average is taken as the objective function value of that solution. Trivially, whenever this average is better than the current best, it will become the current best, and if it is not, the solution will be accepted according to the rule of acceptance, which happens in Algorithm 3 Line 10. For an additional method to combat the randomness of the simulation, one can reuse the scatter matrices that are calculated. In both of the interfaces, every time the regular GenPro4 function is called, a scatter matrix is calculated and these differ from the previous one, even if they have the same input variables due to the randomness of the simulation. Theoretically, only when some changes in the coatings of the interfaces occur, the scatter matrix must be recalculated. In the second interface, the only changes we make are the choice whether or not we use a pyramid structure. This means that before the algorithm, we need to run the simulation twice to calculate the scatter matrices, once with and once without the pyramid structure. These scatter matrices can now be reused in each iteration of the algorithm. Moreover, this first calculation can be done with more accuracy, since the calculation time is less of an important factor, because we only calculate them once. Additionally, while reusing the scatter matrices, especially when the pyramid structure is used, the duration of each iteration is decreased by a considerable amount. This is because most of the iteration is spend calculating the scatter matrix, and again, especially for a pyramid structure.

4 Results

In this chapter, we describe the results of optimizing the current in a tandem solar cell by using the simulated annealing algorithm as described in Chapter 3, on the objective function GenPro4, which is on itself a simulation of a tandem solar cell. The code of the implementation of the algorithm and its initialization can be found in a external file [6]. The results conclude the best settings for the simulated annealing algorithm, where we also consider whether or not a loss in accuracy is worth the time that is won in computation time. In our results, a super computer has been used, here every iteration only takes 1 to 2 seconds. However, when using a computer with a considerable weaker processor, the computation time can take up to 30 seconds each iteration. To achieve this conclusion, the initialization must be conducted carefully, and multiple temperature schemes in different stages of the algorithm must be compared.
In the objective function, the optimization takes place in a 10 dimensional solution space. Six of these variables are continuous and four are discrete. The implemented objective function is an abridgement for a tandem solar cell, which calculates the current of the tandem solar cell. The solar cell consists of 1 thick layer, and 10 thinner layers named coatings, with an additional option for an extra coating. The bottom half of the cell contains the thick silicon substrate layer and 3 coatings which all have a fixed thickness, hence do not require optimization. However, the coatings of the bottom half form the second interface, for which we do consider whether or not to use the pyramid structure. This is the only optimization factor of the bottom half of the cell. The top half of the cell consists of the first interface and contains 7 coatings. The thickness of 5 of these coatings have to be optimized. Moreover, this interface also contains the more important coatings, the HTM and ETM. The optimization of these coatings do not only require an adjustment in thickness, but also material. As last, in the top half of the cell, one can choose if an extra coating should be implemented, which also has a variable thickness. In Figure 4, a representative structure of what needs to be optimized in a tandem solar cell is provided. The coatings that need to be optimized are coloured, the blue colour implies that both the material and the thickness must be optimized, and for the yellow only the thickness. Again, in the bottom half no thicknesses are optimized, and only the choice to use a pyramid structure is optimized.

There are three coatings that only differ in thickness, and three coatings that additionally differ in material. The first coating that differs in material is the extra anti-reflective coating. The choice here is slightly different than the choice for the other materials, since we choose here whether or not we use the extra coating. If we do not use the extra coating, the material is set to be air, and if we do us it, the material is set to magnesium fluoride (MgF_2). For the ETM, three different materials are to be tested, namely, spiro-OMeTAD, fluorine doped tin oxide (FTO) and titanium dioxide (TiO_2). For the last material variable coating, the HTM, there are two material choices, poly(triaryl amine) (PTAA) and nickel oxide (NiO). The last variable that needs to be optimized is the discrete choice for using a pyramid structure or not.
These 10 variables are represented in the objective function. The first six variables determine the thickness of the first 6 coatings of Figure 4, respectively. The seventh to ninth variable determine the materials of the ETM, HTM and anti-reflective coating. And the

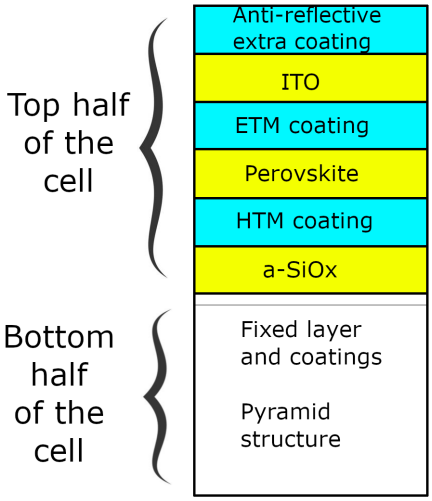

Figure 4: Structure of variables that need to be optimized

final, the tenth variable decides the use of the pyramid structure.

4.1 Initialization

In order to get reliable results from the simulated annealing algorithm, some initialization has to take place. The first initial calculation that has to take place is that the two scatter matrices have to be calculated, which is done by running the simulation GenPro4 once. As standard, the result of the scatter matrix is stored in the workspace of MATLAB. However, when inside a function, specifically our objective function, the workspace can not be reached.
Secondly, the initial solution has to be decided. With simulated annealing, the best found solution should not be dependent on different initial solutions. We generally do not want to have a close to optimum initial solution. Taking these factors into account, the continuously variables are chosen at random. In theory, the discrete variables can also be chosen at random. And multiple random initialisation variables have been tested prior to these end results. However, in the results, we use the same initial solution for all the cases, in order to have a completely fair comparison of the temperature schedules. Here, the continuous variables are set around middle of each boundary interval, the initial solution with the initial objective function value can be found in Table 2.
Next, the initial temperature must be calculated, here the scatter matrices and initial solutions are already to be used, hence it is important to calculate them beforehand. The initial temperature must be recalculated after every major change in the implementation of the algorithm. Remember that the initial temperature is determined from the average of the objective functions of 100 strictly worse solutions after one step and the initial acceptance rate of 80%. In the later states of the study, every calculation of the initial temperature has been close to 13.

4.2 Results of Different Temperature Schedules

Here, we provide the resulting choices of the temperature schedules. The temperature schedules contribute a crucial factor towards the end results of a simulated annealing algorithm. The consideration of the schedules is a careful process in which some trial and error can be used. Remember that the temperature schedules consist of a temperature function, and a step size which decides whenever, after how many iterations, the temperature should be changed. First, we consider certain schedules before reannealing, and conclude which schedules fits our problem the best. After this, we consider some temperature schedules for the part of the algorithm after the reannealing. The temperature schedules determine how quickly and accurate the algorithm converges.

4.2.1 Temperature Schedules Before Reannealing

Remember, the aim of this part of the algorithm, before the next reannealing, is to find a candidate for the best discrete values. This part must be relatively quick because it decides whenever we start the reannealing process. However, more importantly, it has to be accurate since ideally the optimal discrete values have to be found. Reannealing takes place whenever the temperature reached a value of 0.05 for the first time, this values is chosen since at a temperature of 0.05, the algorithm has decently converged. Thus, combined with a fitting temperature schedule, the algorithm has a good idea on what a candidate for the best discrete variables could be. Hence, the following schedules run until the temperature reaches this value. Understanding what result we are aiming to achieve, we test the following temperature functions. All of these temperature schedules have been tested 6 times in order to get a first glance on how good the methods are. We analyse the methods which use these temperature schedules and we reject two obvious dropouts, and the two remaining are tested further. The first schedule consists of a slow convergence, the second of a medium convergence, and the third of a fast convergence. Additionally, a schedule which decreases every iteration is implemented, here, the cooling factor is tweaked such that the last iteration is around the same as the one from the temperature schedule with a medium convergence speed.

1. $T_{new} = 0.95 \cdot T_{old}$ decreases every 10 iterations
2. $T_{new} = 0.95 \cdot T_{old}$ decreases every 7 iterations
3. $T_{new} = 0.95 \cdot T_{old}$ decreases every 5 iteration
4. $T_{new} = 0.9935 \cdot T_{old}$ decreases every iteration

Firstly, a very slow decreasing temperature schedule is tested, namely $T_{new} = 0.98 \cdot T_{old}$ with a decrease every 20 iterations. This is done in order to conclude whether or not the discrete values of the other schedules are close to optimal. This very slow schedule gives a fairly accurate estimation on what the discrete variables should be. From now on, we refer to this schedule as the Test-Schedule. Figure 5 depicts the results of the Test-Schedule, it indeed shows a very slow convergence, where even in the tail every discrete variable is tested equally. It is assumed that the discrete values of the best found solution are close to optimal, and probably even optimal.

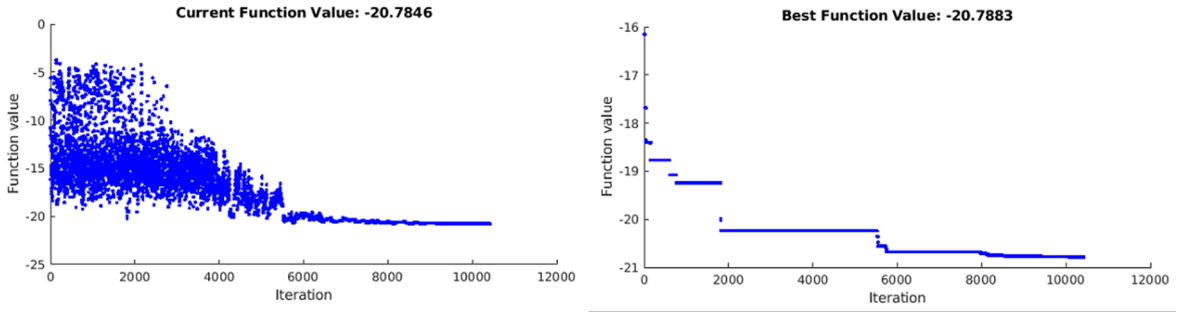

Figure 5: The Test-Schedule: A very slow convergence. Left: the objective function value at each iteration. Right: the current best found objective function value.

Temperature Schedule 1 is represented in Figure 6, here, a) displays the objective function value at each iteration, and b) the current best found objective function value, this will be the case for all schedules. In this schedule the desired convergence is slowly. When compared with the upcoming schedules, one will see that the convergence is indeed slow. Because the method with this schedule has found the same discrete variables as the Test-Schedule on every run, the chances of finding a good candidate are high. If, however, there is a faster method that has a similar result, this method becomes redundant.

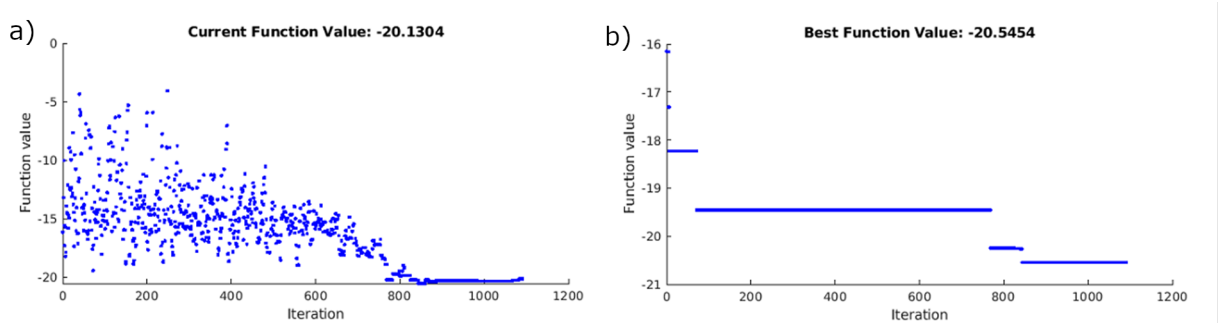

Figure 6: Temperature Schedule 1: A decrease every 10 iterations according to the following function: $T_{new} = 0.95 \cdot T_{old}$. Left: the objective function value at each iteration. Right: the current best found objective function value.

Temperature Schedule 2 is represented in Figure 7, in this schedule the convergence is more rapidly than in the previous schedule, because the temperature decreases faster. Now, one could argue that this will influence the accuracy of the schedule used. To counter this, the best methods will be tested more thoroughly. This schedule has always found the same discrete values as the Test-Schedule, thus it, for now, performs better than the previous one.

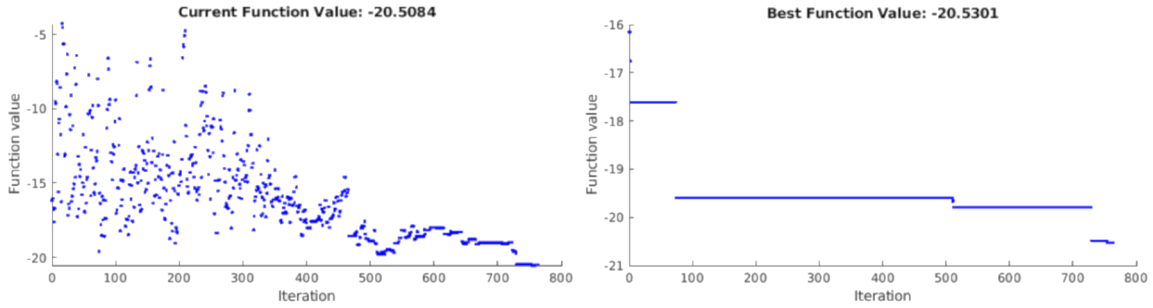

Figure 7: Temperature Schedule 2: A decrease every 7 iterations according to the following function: $T_{new} = 0.95 \cdot T_{old}$. Left: the objective function value at each iteration. Right: the current best found objective function value.

Temperature Schedule 3 is represented in Figure 8, this schedule did not find the same values for the discrete variables as the Test-Schedule in 1 of the 6 tests. The figure displays a test which did not find them, here one can clearly see that the objective function value is much lower than the other methods, the wrong discrete variables influence this error heavily.

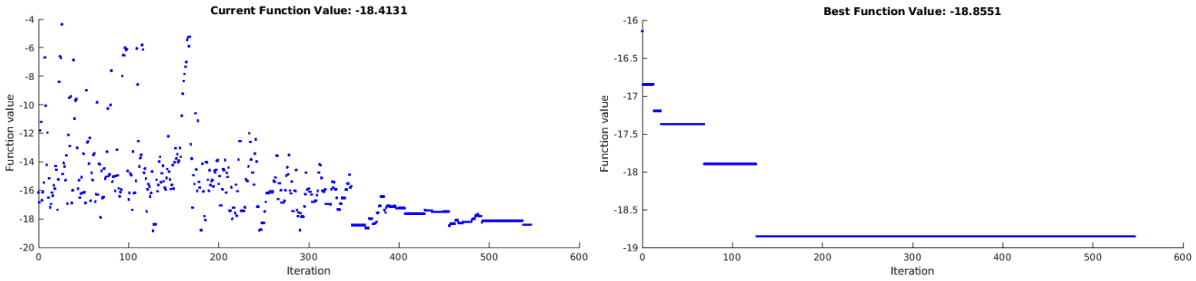

Figure 8: Temperature Schedule 3: A decrease every 5 iterations according to the following function: $T_{new} = 0.95 \cdot T_{old}$. Left: the objective function value at each iteration. Right: the current best found objective function value.

Temperature Schedule 4 is represented in Figure 9, this schedule is much alike schedule 2 in comparison. They both end around the same iteration, and both found the best solution in every test of the 6.

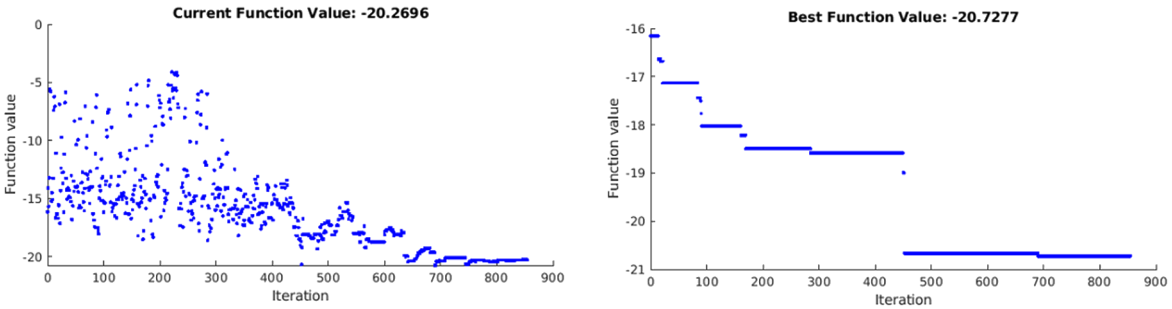

Figure 9: Temperature Schedule 3: A decrease every iterations according to the following function: $T_{new} = 0.9935 \cdot T_{old}$. Left: the objective function value at each iteration. Right: the current best found objective function value.

As noticed, the third schedule is clearly not good enough, thus this schedule can not be a candidate for the schedule of the algorithm part before reannealing. The first one is probably the most accurate, however, until now, the other tested schedules have never gotten a different result than the Test-Schedule. The second and the fourth schedule have been tested more thoroughly, if they showed that they made too many errors, the first schedule can still be chosen.
After testing both the second and the fourth iteration more than 20 times each, we can make a conclusion on which schedule we can use. The second schedule always found the same discrete values as the Test-Schedule, and the fourth was once one variable off. This does not automatically decides that the second schedule is better, however, the second schedule does fit our method better anyways since it makes use of a type of schedule which is used also after reannealing, that is, a decrease after a certain number of iterations. Thus, the second schedule is chosen to be the schedule of the part of the algorithm before reannealing.

4.2.2 Temperature Schedules After Reannealing

In the second part of the simulated annealing algorithm, a fitting temperature schedule has to be implemented as well. It is assumed that the part before reannealing has given us a good candidate for the optimum discrete values, however, we cannot assume that these are already the best discrete values, thus, they cannot be considered as fixed values. The MATLAB function for simulated annealing, has an option input in which several optimization options can be given, this function optimizes a provided objective function. The algorithm runs with the previously decided temperature scheme before the reannealing. Here, three runs of the algorithm will be compared with each other. These algorithms have been each been ran three times, the results of the best runs are included. Additionally, a short comparison with the previously used Test-Schedule is made, in order to show that the manual reannealing does provide some benefits. The results are also more in depth, in addition to the figures on the behaviour of the algorithms, the exact values of the continuous and discrete variables are provided in tables.
Then, firstly, the adjusted simulated annealing algorithm as described in the methodology with a slow decreasing temperature schedule is conducted. In this temperature schedule, when the temperature is above 1, the following schedule holds, $T_{new} = 0.95 \cdot T_{old}$ with a decrease every 7 iterations, otherwise, $T_{new} = 0.97 \cdot T_{old}$ with a decrease every 12 iterations. In order to stop the algorithm at a certain point, we set the function tolerance at 10^{-2} and the *MaxStallIterations* at 2000. This means, that whenever the best found solution does not vary more than 10^{-2} in 1500 iterations, the program will be terminated. Before running this optimization algorithm, we expect to see a lot of function evaluations due to the slow decrease of the temperature. However, we do expect a very close to optimal solution. All of the methods are represented in a figure, which displays the behaviour of the algorithm, and a table in which the exact values are given. In Figure 10, the behaviour of the algorithm is displayed in a similar style as seen before, the part before and after reannealing are very distinctive of each other. First note that, if these results are again compared with the Test-Schedule of the previous part, it shows that the manually reannealing helps in shortening the running time of the algorithm, additionally, the accuracy also increases. All of this happens because the discrete values now converge in probability as well, whereas before, they only converged in their contribution to the value of the objective function. This happens because in the Test-Schedule, at any point

in the algorithm all of the discrete variables had the same probability to be chosen when creating a new solution.

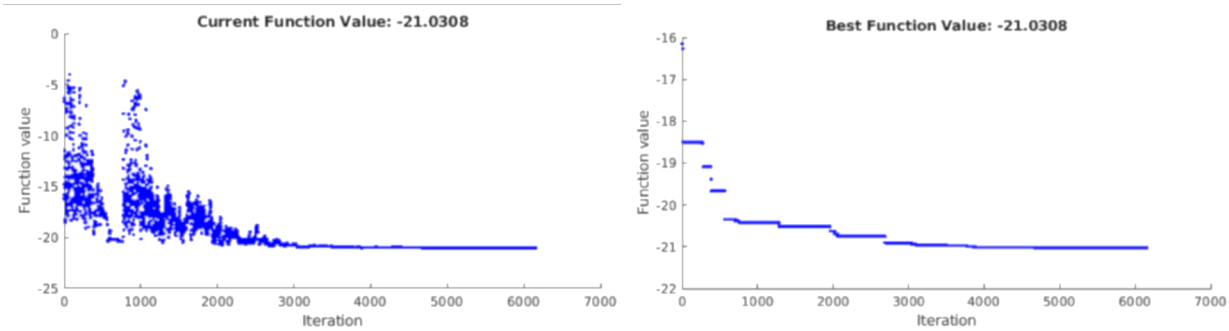

Figure 10: Algorithm's behaviour with a slow decreasing temperature schedule after reannealing. Left: the objective function value at each iteration. Right: the current best found objective function value.

In Table 2, the results of this algorithm are represented as follows, the first column represents the variable names, the second column the values of the initial solution, and in the last column the final results of the optimal found solution. The optimal found solution is here 21.0308, and it is found in 6160 iterations. The program was terminated because the best found objective function value did not change more than 10^{-2} in 1500 iterations. In comparison with the Test-Schedule, this program concluded its optimum value in less than half the iterations, moreover, the optimum value is considerable better.

Table 2: Results of the slow converging temperature schedule.

Variables	**Initial solution**	**Best found solution**
Pyramid texture	yes	yes
Anti-reflective coating material	air	MgF_2
ETM material	TiO_2	TiO_2
HTM material	PTAA	NiO
Anti-reflective coating thickness [μm]	0.15	0.1011
ITO thickness [μm]	0.15	0.0569
ETM thickness [μm]	0.15	0.1541
Perovskite thickness [μm]	0.3	0.3829
HTM thickness [μm]	0.15	0.1731
a-SiOx thickness [μm]	0.15	0.0721
Total current [mA/cm^2]	16.1471	21.0308
Function evaluations		6160

In the second case, we use the same objective function, but with different settings, the scatter matrix used in both cases are the same to make the comparison of the schedules most legitimate. Here, we test a medium speed convergence, the best way to do this is to change the temperature function to a faster decrease. In this temperature schedule, when the temperature is above 1, the following schedule holds, $T_{new} = 0.95 \cdot T_{old}$ with a decrease every 5 iterations, otherwise, $T_{new} = 0.97 \cdot T_{old}$ with a decrease every 8 iterations. The function tolerance is kept at the same value, 10^{-2} with the same *MaxStallIterations*.

Again, the behaviour of the algorithm is represented with Figure 11.

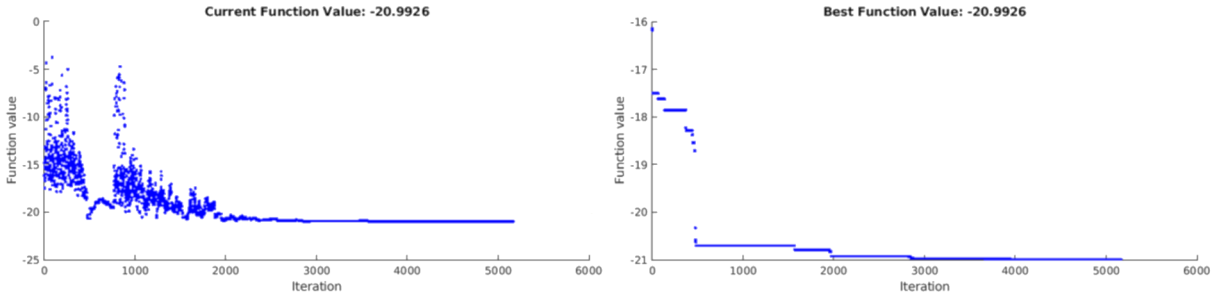

Figure 11: Algorithm's behaviour with a medium speed decreasing temperature schedule after reannealing. Left: the objective function value at each iteration. Right: the current best found objective function value.

In Table 3, the results of this slightly slower converging algorithm are given, which has the same representation as the previous table. The most important values to examine are the *Total current* and the *Function evaluations*. In this method, the algorithm had a slightly worse best function value, namely 20.9926, however, the program terminated after iterations 5163, which is almost 1000 iterations earlier. Furthermore, comparing the two tables, some values are fairly close to each other, and some are far apart. In the case of the slightly faster convergence the algorithm possibly got stuck in one local optimum.

Table 3: Results of the medium speed converging temperature schedule.

Variables	**Initial solution**	**Best found solution**
Pyramid texture	yes	yes
Anti-reflective coating material	air	MgF_2
ETM material	TiO_2	TiO_2
HTM material	PTAA	NiO
Anti-reflective coating thickness [μm]	0.15	0.1026
ITO thickness [μm]	0.15	0.0564
ETM thickness [μm]	0.15	0.3124
Perovskite thickness [μm]	0.3	0.3827
HTM thickness [μm]	0.15	0.5000
a-SiOx thickness [μm]	0.15	0.0777
Total current [mA/cm^2]	16.1471	20.9926
Function evaluations		5163

Lastly, a temperature schedule with a more rapid convergence is implemented, again on the same scatter matrix, and with the same temperature schedule for the part before reannealing. In this temperature schedule, when the temperature is above 1, the following schedule holds, $T_{new} = 0.95 \cdot T_{old}$ with a decrease every 5 iterations, otherwise, $T_{new} = 0.97 \cdot T_{old}$ with a decrease every 3 iterations. The stopping criterion is again the same, and the behaviour of the algorithm is represented in Figure 12. Note that there is indeed a fast convergence, since the best found function value at iteration 2000 does not

differ much from the best found function values at the end.

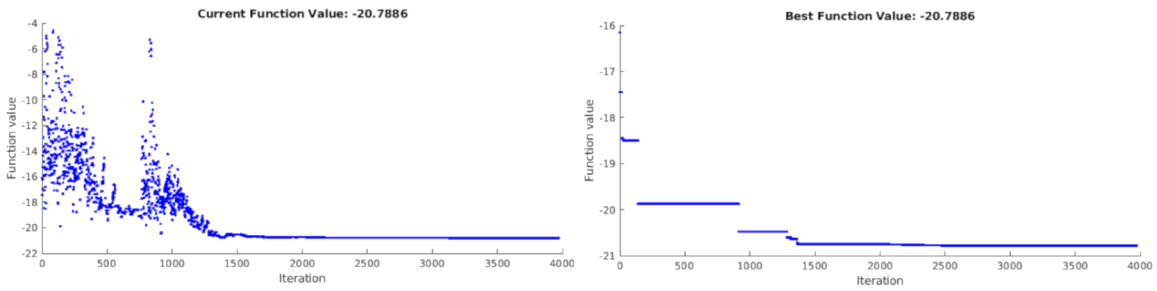

Figure 12: Algorithm's behaviour with a fast decreasing temperature schedule after reannealing. Left: the objective function value at each iteration. Right: the current best found objective function value.

Table 4, presents the results of the temperature schedule with a fast convergence. The best function value is 20.7886, and the algorithm terminated in 3974 iterations. This result is considerably worse than the last two found solutions.

Table 4: Results of the fast converging temperature schedule.

Variables	**Initial solution**	**Best found solution**
Pyramid texture	yes	yes
Anti-reflective coating material	air	MgF_2
ETM material	TiO_2	TiO_2
HTM material	PTAA	NiO
Anti-reflective coating thickness [μm]	0.15	0.1017
ITO thickness [μm]	0.15	0.0536
ETM thickness [μm]	0.15	0.1783
Perovskite thickness [μm]	0.3	0.3639
HTM thickness [μm]	0.15	0.4310
a-SiOx thickness [μm]	0.15	0.4762
Total current [mA/cm^2]	16.1471	20.7886
Function evaluations		3974

5 Conclusion

In this research, a simulated annealing algorithm is implemented to optimize the efficiency of a tandem solar cell. The goal of this research was to find the best implementation of the simulated annealing algorithm to a specific problem, namely, the optimization of the current in a tandem solar cell. There are different settings that ought to be fine-tuned during the implementation in order to make the simulated annealing algorithm find a close to optimal solution. Not only the best result in function value is considered, but also how rapid the algorithm can conclude this result. This was important to be taken into account, since the computation time of the objective function is high. The aim was to find a perfect matching of result and computation time.
Some attention has to be shed on how and where the simulated annealing algorithm was implemented. The objective function is GenPro4, this function is a simulation of a tandem solar cell which has the total current as output. GenPro4 is implemented in MATLAB, hence the simulated annealing algorithm is easiest to be implemented in MATLAB as well. MATLAB has a function that runs a simulated annealing algorithm, namely *simulannealbnd*. Firstly, some light is shed on the creation of a new solution. The default setting of *simulannealbnd* is developed to handle a broad range of problems, thus also on large intervals. In our problem, the variables are on relatively small intervals, and while using the default setting of the function, some inefficiencies will occur. In the newly implemented function for the creation of new solutions, the solutions are created with more respect towards the boundaries of the intervals. Secondly, the function *simulannealbnd* only allows a smooth objective function. This conflicted with our objective function GenPro4, since it has some discrete variables. The solution to this problem is to handle the discrete variables first as continuous variables, which are later rounded to discrete values. The discrete variables were at first chosen at random, this made it that they are never converging in probability. In order to combat this, a manual reset of temperature is implemented, which is called reannealing. This reannealing takes place whenever the temperature reaches a value of 0.05 for the first time. After this reannealing, the chance of using the best found discrete variables is higher and converges to 1 when the temperature lowers.
Next, the temperature schedules used have the greatest impact on the result of the simulated annealing algorithm. We consider two temperature schedules, since the reannealing divides the algorithm in two parts, one part before, and one part after the manual reannealing. In the part before reannealing, a temperature schedule which converges rapidly, but accuratly has to be implemented. Some cases have been considered, the result of the best temperature schedule was the one with a decrease each 7 steps using the following function: $T_{new} = 0.95 \cdot T_{old}$.
Finally, it rests to decide the best temperature schedule after reannealing, in combination with the part before reannealing, this provides the end result. Temperature schedules that conduct a slow, medium and fast convergence were tested, multiple times, the best found results are shown. The temperature schedule with the slower convergence had the best result, a total current of 21.0308 mA/cm^2 had been found in 6160 iterations, it concluded its end results always around 6000 iteration and always found an objective function value higher than 21. A slightly faster temperature schedule resulted in a slight loss of total current, the result was 20.9926 mA/cm^2 in 5163 iterations. Here, the algorithm most of

the time ended around 5000 iterations, and always found a best objective function value higher than 20.90 The temperature schedule that resulted in a fast convergence had a best found value that was diverged too much to be considered. At last, the consideration whether or not the reduction of the optimal current found in the faster temperature schedule is worth the decrease of 1000 iterations depends on a couple of factors. These factors are the computational speed for calculating the objective function, a computer with a low computational speed could desire the 1000 less iterations, since it could save plenty of time. Additionally, one must consider the significance that should be attached to a slightly better objective function value.

6 Recommendation

In this research, a simulated annealing algorithm has been implemented in order to optimize the objective function GenPro4, which calculated the current of a tandem solar cell. There are several recommendations for future research on the use and implementation of the simulated annealing algorithm to this topic specifically. Firstly, the most important recommendation is to not use the MATLAB function *simulannealbnd*. Despite the fact that this function is well written and works very good in the right circumstances with the right settings, for this problem, and any problem with unique constraints, such as discrete variables, it should be recommended to write a new implementation of a simulated annealing algorithm. Studying the code of the *simulannealbnd* forges a good idea on the structure of a simulated annealing algorithm, and when using discrete variables, the method described in this study can be implemented.
Moreover, if one decides to make use of the manual reannealing as described in this study, the part before the reannealing could be considered separately in an algorithm from the part after. Whilst running our algorithm with no real stopping criterion the following goes unnoticed, however, when the maximum stall iteration stopping criterion is implemented, the part before reannealing can lead to a premature stoppage of the algorithm. This happens because there is a probability that a close to optimum solution is found in the part before reannealing, and whenever the part after does not find a better solution in the provided max stall iteration amount, the program will terminate prematurely, which leads to a considerable amount of time wasted.

References

[1] IEC 60904-3. "Photovoltaic devices - Part 3: Measurement principles for terrestrial photovoltaic (PV) solar devices with reference spectral irradiance data". In: (Feb. 2019).

[2] K. Jäger; B. Rech; S. Albrecht. "Numerical optical optimization of monolithic planar perovskite-silicon tandem solar cells with regular and inverted device architectures." In: *Opt. Express* 25 (2017), A473–A482. DOI: https://doi-org.tudelft.idm.oclc.org/10.1364/OE.25.00A473.

[3] A. A. B. Baloch; S. P. Aly; M. I. Hossain; F. El-Mellouhi; N. Tabet; F. H. Alharbi. "Full space device optimization for solar cells". In: *Scientific Reports* 7 (2017). DOI: https://doi.org/10.1038/s41598-017-12158-0.

[4] C. J. P. Bélisle. "Convergence Theorems for a Class of Simulated Annealing Algorithms on R^d". In: *Journal of Applied Probability* 29.4 (1992), pp. 885–895. DOI: https://doi.org/10.2307/3214721.

[5] W. Ben-Ameur. "Computing the initial temperature of simulated annealing". In: *Computational Optimization and Applications* 29.3 (2004), pp. 369–385.

[6] P. van de Lest. *Simulated Annealing Code in MATLAB*. Aug. 2021. URL: https://github.com/RudyNiet/Simulated-Annealing-Algorithm-MATLAB-PimVanDeLest.

[7] F. Liang. *Optimization Techniques - Simulated Annealing*. Apr. 2020. URL: https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7.

[8] F. Pelanchon; P. Mialhe. "Optimization of solar cell performance". In: *Solid-State Electronics* 33 (1990), pp. 47–51. DOI: https://doi.org/10.1016/0038-1101(90)90008-3.

[9] B. Jeon; J. Kim; K. Lim; Y. Choi; M. Moon. "Calculation of Detector Positions for a Source Localizing Radiation Portal Monitor System Using a Modified Iterative Genetic Algorithm". In: *Journal of Radiation Protection and Research* 42 (Dec. 2017), pp. 212–221. DOI: 10.14407/jrpr.2017.42.4.212.

[10] Y. Nourani and B. Andresen. "A comparison of simulated annealing cooling strategies". In: *Journal of Physics A: Mathematical and General* 31.41 (Oct. 1998), pp. 8373–8385. DOI: 10.1088/0305-4470/31/41/011.

[11] K.M. El-Naggar; M.R. AlRashidi; M.F. AlHajri; A.K. Al-Othman. "Simulated Annealing algorithm for photovoltaic parameters identification". In: *Solar Energy* 86.1 (2012), pp. 266–274. ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2011.09.032. URL: https://www.sciencedirect.com/science/article/pii/S0038092X11003586.

[12] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003. ISBN: 3-540-44389-4.

[13] J.P. Sethna. *Statistical mechanics : entropy, order parameters, and complexity*. Oxford ; New York : Oxford University Press, 2006, pp. 99–134. ISBN: 9780198566779.

[14] J. Dréo; A. Pétrowski; P. Siarry; E. Taillard. *Metaheuristics for Hard Optimization*. Springer, 2006. ISBN: 978-3-642-06194-3. DOI: https://doi-org.tudelft.idm.oclc.org/10.1007/3-540-30966-7.

[15] N. Metropolis; A. W. Rosenbluth; M. N. Rosenbluth; A. H. Teller. "Equation of State Calculations by Fast Computing Machines". In: *J. Chem. Phys.* 21 (1953). DOI: https://doi-org.tudelft.idm.oclc.org/10.1063/1.1699114.

[16] G. Alonso; E. del Valle; J. R. Ramirez. *5 - Optimization methods*. Woodhead Publishing, 2020, pp. 67–76. ISBN: 9780128200216. DOI: https://doi.org/10.1016/B978-0-12-820021-6.00005-3.

[17] R. Santbergen; A. H. M. Smets; M. Zeman. "Optical model for multilayer structures with coherent, partly coherent and incoherent layers". In: *Opt. Express* 21.S2 (Mar. 2013), A262–A267. DOI: 10.1364/OE.21.00A262. URL: http://www.opticsexpress.org/abstract.cfm?URI=oe-21-102-A262.

[18] R. Santbergen; T. Meguro; T. Suezaki; G. Koizumi; K. Yamamoto; M. Zeman. "GenPro4 Optical Model for Solar Cell Simulation and Its Application to Multijunction Solar Cells". In: *IEEE Journal of Photovoltaics* 7.3 (2017), pp. 919–926. DOI: https://doi.org/10.1109/JPHOTOV.2017.2669640.

[19] M. Ortner; X. Descombes; J. Zerubia. "An adaptive simulated annealing cooling schedule for object detection in images". In: (Jan. 2007).