Multi-objective differential evolution optimization of ion beam analysis spectra

MSc. Thesis Computer Science

Simon Mariën

Delft University of Technology

TU Delft · imec

Multi-objective differential evolution optimization of ion beam analysis spectra

by

Simon Mariën

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday March 22, 2024 at 13:15.

Student number:	420333	
Project duration:	September 3, 2023 – March 22, 2024	
Thesis committee:	Dr. ir. N. Yorke-Smith,	TU Delft, supervisor
	Dr. J. Meersschaut,	Imec, supervisor
	Dr. A. Panichella,	TU Delft

Cover: Fractal Background Abstract
Style: TU Delft Report Style

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

In the rapidly evolving semiconductor industry, precise material characterization is crucial. This thesis focuses on enhancing Ion Beam Analysis (IBA), a pivotal tool in semiconductor characterization, through the integration of differential evolution optimization. This research proposes a single and multi spectra optimization algorithm approach controlled by a web application. Central to this work is a comparative analysis of the proposed algorithms to simulated annealing and the DE algorithm proposed by Heller et al. [5]. This analysis shows a good performance of the proposed algorithms and a potential for industry application. The web application offers users a robust, user-friendly, and scalable interface for IBA optimization. By proposing and testing new IBA optimization methods, this thesis contributes significantly to semiconductor technology, offering new methods for material characterization at microscopic level.

This project is a joint project between Imec (Leuven, Belgium) and HZDR (Dresden, Germany).

Contents

1 Introduction

In 1965, when Gordon Moore predicted that the amount of transistors on an integrated circuit would double every two years [4], it would have been hard to belief that this prediction still holds almost 60 years later. Moore's prediction, known as Moore's law, has since been used in the semiconductor industry to guide long-term planning and to set targets for research and development, thus functioning to some extent as a self-fulfilling prophecy.

The will to keep up with Moore's law made semiconductor technology, the cornerstone of modern technological progress, rapidly evolve. At the core of nearly every digital device, semiconductors become increasingly smaller and complex. This rapid evolution has been largely driven by the continuous scaling down of semiconductor components, and there seems to be no sign of slowing down. The miniaturization trend has lead to unprecedented progress in terms of computing, communication, and automation. Increasing miniaturization leads to efficiency and better performance but it also presents the need for significant challenges in material characterization at microscopic level.

Ion Beam Analysis (IBA) stands as a critical tool in this area. It offers a precise method to analyse the structure and composition of materials, offering precision and depth at atomic level. This microscopic precision makes ion beam analysis a perfect fit for semiconductor material characterization. However, as semiconductors become smaller and complex, the limitations of traditional IBA methods become evident. Using IBA optimization techniques, we aim to determine material characteristics.

This thesis work is situated at the intersection of two pivotal fields, both having the goal to improve current IBA techniques.

Differential Evolution (DE) is an optimization algorithm that belongs to the family of evolutionary algorithms. DE is known for its simplicity, speed, and robustness, which could make it a powerful method to improve IBA optimization.

Software Engineering (SE) entails the systematic application of engineering approaches to the development of software. Software engineering plays a crucial role in creating sophisticated tools and applications that can implement advanced algorithms like DE. The development of user-friendly, scalable, and robust software solutions enables researchers and professionals in the semiconductor industry to conduct more effective and precise material characterizations.

This thesis aims to explore how Differential Evolution optimization and Software Engineering techniques can be combined to enhance Ion Beam Analysis, addressing the challenges of microscopic material characterization.

1.1. Problem Statement

Heller et al. [5] developed Ruthelde, a software package, to perform IBA using differential evolution optimization. While it significantly contributed to the field of material characterization, its practical application has brought some challenges to light that underscore the need for innovation.

Ruthelde supports single spectra IBA where the optimization values are evaluated using only one measurement. With multiple spectra measurements, the optimization values are evaluated to multiple measurements thus making the search space smaller. This will be more apparent for a complex material. The current algorithm's architecture, tailored for single spectra analysis, constrains the software's capability in more complex, real-world applications. Next, the core optimization algorithm of Ruthelde is custom-built using Java which could make it less optimized than well-tested and open-source alternatives from a software library.

Another significant hurdle Ruthelde presents, is the fact that Ruthelde is a desktop application. This software engineering choice imposes restrictions on the usability and scalability of the software. The installation process can be tedious and could require specific requirements and dependencies that may not be available on every system and device. Next, complex experiments require a lot of computation. Ruthelde is limited to the hardware of the device the desktop application is installed on. This limitation poses a significant challenge in research environments where scalability and flexibility are important.

1.2. Research Questions

This research aims to explore several pivotal questions to advance the application of differential evolution optimization in ion beam analysis. This research mainly focuses on solving the problems that are identified in Ruthelde.

These questions include:

1. How can Differential Evolution optimization be effectively applied to multi-spectra Ion Beam Analysis to improve material characterization accuracy and efficiency?
2. What are the key software engineering challenges in developing an advanced Ion Beam Analysis platform that incorporates Differential Evolution optimization, and how can these challenges be addressed?
3. In what ways do the newly developed optimization algorithms improve upon Ruthelde in terms of accuracy and efficiency?

Previous questions guide the research towards a comprehensive understanding and improvement of ion beam analysis methods, assessing both theoretical and practical aspects.

1.3. Contribution

This thesis work makes several significant contributions to the field of Ion Beam Analysis by using techniques from the fields of Differential Evolution and Software Engineering. The author's contributions can be summarized in:

- Selection, implementation and comparison of suitable DE mutation strategies.
- Implementation and analysis of the single spectra DE optimization algorithm.
- Implementation and analysis of the multi spectra DE optimization algorithm.
- Implementation and analysis of the simulated annealing optimization algorithm.
- Comparative analysis of the single spectra DE optimization algorithm to the simulated annealing optimization algorithm.
- Comparative analysis of our proposed optimization algorithms to the algorithm proposed by Heller et al. [5].
- Design and implementation of a web application to control the proposed algorithms.

1.4. Thesis outline

This report is organized as follows. Chapter 2 provides an introduction to relevant background topics Ion Beam Analysis, Differential Evolution, and Ruthelde. Chapter 3 presents the optimization part of this research. First we look at the motivation behind IBA optimization followed by the optimization's requirements. After, the chosen library is discussed followed by details on the DE optimization model for both single and multi spectra optimization. At the end of the chapter, simulated annealing, as

an alternative optimization model is discussed in detail. Chapter 4 comprises a description of the software engineering part of this research. Followed by the motivation and requirements for the web application, the chosen technology stack and interesting design choices are discussed. Chapter 5 shows the experiments and results that test the optimization models presented in Chapter 3. The chapter ends with a thorough comparative analysis between all proposed models and Ruthelde by Heller et al. [5]. Finally, Chapter 6, concludes the thesis with a review of the key points and a look at future work.

2

Prior work and background

This chapter presents the fundamentals of ion beam analysis and differential evolution. Furthermore, Ruthelde, a software package for simulation and automated fitting of IBA spectra is discussed. Ruthelde will function as the baseline of this research.

2.1. Ion Beam Analysis

Ion Beam Analysis is a suite of analytical techniques used in material science to find the composition and structure of materials by analyzing the interaction between a high-energy ion beam and the target material. IBA is known for its depth profiling, non-destructive and quantitative nature.

The core principles of IBA involve an ion beam targeted at a sample material. The ion beam is generally consisting of protons or helium ions. The particles of the ion beam hit the target material atoms causing scattering, energy loss and nuclear reactions. These interactions provide a wealth of information about the sample's structure and composition. The most frequently used techniques that utilize this information are Rutherford Backscattering Spectrometry (RBS), Particle-Induced X-ray Emission (PIXE), Nulear Reaction Analysis (NRA) and Elastic Recoil Detection Analysis (ERDA). This research focuses on improving Rutherford Backscattering Spectrometry.

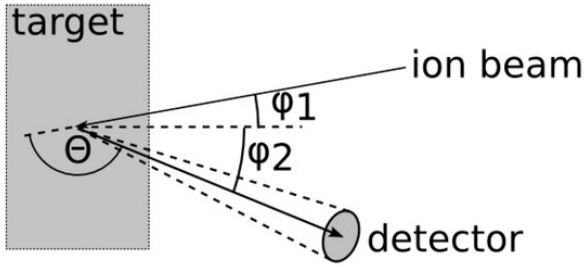


Figure 2.1: Rutherford Backscattering Spectrometry [16]

Rutherford Backscattering Spectrometry (RBS) is a technique where Lord Rutherford laid the foundation in 1911 [11]. In RBS, a high-energy beam of ions are impinging the nuclei of the target atoms sample and this causes the ions being scattered. A detector capable of measuring the energies of backscattered ions is used to gather the spectrum measurement. Figure 2.1 gives a simple depiction of the ion beam, target material and scattered ions caught by the detector.

This measurement can then be used to determine the composition of the sample, more specifically the areal density of each layer and each element ratio in the layer. The areal density is calculated as the mass per unit area. The SI derived unit is atoms per square centimeter (atoms·cm^{-2}). The ratio of an element in a layer is the percentage of the layer that is that specific element. The areal density of an element in a layer is found by multiplying the areal density of the layer with the ratio of the element.

Rutherford Backscattering Spectrometry (RBS) can be solved using different algorithms. Jeynes et al. [6] used simulated annealing to approach RBS. Simulated annealing performs well early in the optimization but struggles in the end. Barradas et al. [1] approached RBS with an artificial neural network. Neural networks give instantaneous results but they have to be trained first. Training a neural network is a long process and for every small change in the setup, all training has to be redone. Finally, Heller et al. [5] used differential evolution and demonstrated that their implementation is robust and achieved good precision and accuracy. Section 2.3 will give a more in-depth explanation of Ruthelde.

2.2. Differential Evolution

Differential evolution (DE) is a powerful and efficient algorithm for optimization problems, introduced by Storn and Price in 1997 [13]. DE is a member of the evolutionary algorithms family where it is distinct from other algorithms by two main points. The genotype is some form of real-valued vector and the mutation/crossover operations make use of the difference between two or more vectors in the population to create a new vector. DE is particularly efficient in non-linear, non-differentiable, and complex multidimensional problems with a large searchspace. The main drawback of DE is that the optimization can get stuck in a local optima which causes a sub-optimal solution.

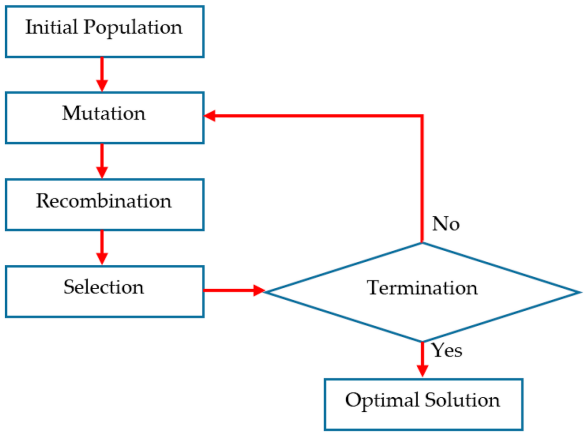


Figure 2.2: [2]

Figure 2.2 gives a diagram of the different steps in DE. In the next couple of subsections, every step is discussed in depth.

2.2.1. Initial population

In the first step of the DE optimization algorithm we decide on a set of parameters. The first thing to consider is the amount of variables we want to find and the minimum and maximum value each variable can be. The minimum and maximum tuple of a variable is called the bound. The next thing to choose are the stopping conditions. Examples of stopping conditions are discussed in subsection 2.2.5. The last thing to choose is the population size. The population size N is the amount of vector solutions each generation encompasses.

After all DE parameters are chosen, the initial population is constructed. Each generation, also the initial generation holds a population S^G of N solutions. A solution consists of individuals. An individual is a P dimensional vectors $X = \{x_1, x_2, ..., x_P\}$ where each element represents the fit variable of the problem. The individuals of the initial generation are chosen randomly with the constraint that the values are between the variables bounds.

2.2.2. Mutation

In the mutation phase, new solution vectors are generated by combining existing solutions of the current generation. In this step, each individual of the current generation can be used to generate new solutions. There are different mutation strategies, but typically, for each individual in the current population, a

mutated vector V is generated using equation 2.1.

$$V = X_1 + F * (X_2 - X_3) \tag{2.1}$$

$F \in [0, 1]$ is the differential weight and X_1, X_2, and X_3 are randomly chosen individuals of the population that are distinct. A typical setting of F is 0.8.

2.2.3. Recombination

In the recombination phase, mutated solutions are combined with existing solutions to form crossover individuals. This step ensures that old and new information and properties are both part of the next population. This step also helps diversification of the solutions. Typically, equation 2.2 is used in the recombination step.

$$U = \begin{cases} V & \text{if rand}(0,1) \leq CR, \\ X & \text{otherwise,} \end{cases} \tag{2.2}$$

$CR \in [0, 1]$ is the crossover probability and rand(0,1) is a uniformly distributed random number $r_i \sim U(0, 1)$. A typical setting of CR is 0.9.

2.2.4. Selection

In the selection phase, a decision is made which solutions are kept for the next generation. The DE algorithm evaluates both the existing as the new solutions for their fitness. The best solution vectors are then chosen to be in the next generation. Equation 2.3 is typically used in the selection step.

$$X = \begin{cases} U & \text{if } f(U) \leq f(X), \\ X & \text{otherwise.} \end{cases} \tag{2.3}$$

f is the fitness function that needs to be minimized. The fitness function is problem dependent. The fitness function influences the runtime of the DE algorithm a lot. An inefficient or complex fitness funtion can lead to long runtimes.

2.2.5. Termination

The DE algorithm loops through the mutation, recombination, and selection step until a termination criteria is met. There are different termination criteria to be chosen and combined. Typically the following two criteria are used for termination.

- Max iterations: when the DE algorithm reaches a predefined number of generations, the program terminates and returns the solution with the best fitness value of the last generation.
- Adequate fitness reached: when solution m has a fitness equal or smaller than a predefined desired fitness, the DE algorithm terminates and returns solution m as the result.

An optimal solution m is the solution where $f(m) \leq f(p)$ for all p in the search-space. Because the DE algorithm is often used for problems with a complex shaped search space, the exact real-valued solution m is very hard to find.

2.3. Ruthelde

Ruthelde is a software package for simulation and automated fitting of IBA spectra developed by Heller et al. [5]. Material characterization is done using Rutherford Backscattering (RBS). Ruthelde is an open-source software application that uses differential evolution for the fitting of RBS-spectra. This tool tackles different challenges in the analysis of RBS data, facilitating deeper understanding of material properties at microscopic level. Screenshots of the most important pages of Ruthelde can be found in Appendix B.

2.3.1. Functionality

The first feature of Ruthelde is the ability to simulate RBS-spectra on predefined target models. This allows users to predict how changes in material composition and structure affect the spectrum. This feature is essential for interpreting experiment data and fitting a target model.

Ruthelde can also do automated target model fitting. Using differential evolution, Ruthelde automates the process of fitting simulated spectra to experimental data. This reduces the time and expertise needed to analyse RBS-spectra. Ruthelde makes the analysis of RBS-spectra more accessible to researchers and professionals.

The second feature is the ability to asses the uncertainties associated with RBS measurements. Understanding these uncertainties is crucial for the interpretation of material properties and the reliability of these.

Finally, Ruthelde offers a variety of tools to support different aspects of IBA. These include a stopping calculator, an IBA kinematics calculator, and an ion penetration depth plotter. The tools provides ways to get additional insights into the interactions between ion beams and target materials.

2.3.2. Technical details

Ruthelde is written in the programming language Java, which makes it usable on different operating systems. This platform independence makes Ruthelde a valid option in the scientific community, regardless of the preferred computer environment. The codebase of Ruthelde is publicly available on GitHub [10] which makes the software accessible and possible to make custom changes. Being written in Java, all calculations and optimization is performed on the user's local machine. The performance of Ruthelde is thus limited to the performance of the local machine and there is no way to use Ruthelde on a more performant server or super computer.

2.3.3. Differential evolution

Target model fitting in Ruthelde is done using differential evolution. The mutation function used is the same as in the general differential evolution algorithm. The mutation vector V_j^G can be found by formula 2.4.

$$V_j^G = X_{r1}^G + F * (X_{r2}^G - X_{r3}^G) \tag{2.4}$$

$F \in [0, 1]$ is the scaling factor and X_{r1}^G, X_{r2}^G, and X_{r3}^G are randomly chosen individuals of the population of this generation G and $r1 \neq r2 \neq r3$.

In the selection step, the fitness function of a solution is achieved by minimizing the sum of squared residuals (χ^2) between the simulated and the experimental spectra using formula 2.5.

$$\chi^2 = \sum_{i=0}^{N} \left(x_i^{experimental} - x_i^{sim} \right)^2 \tag{2.5}$$

N is the amount of energy channels, this is typically 1024. $x_i^{original}$ is the height of the ith channel of the original spectrum. x_i^{sim} is the height of the ith channel of the simulated spectrum.

This sum is then normalized by formula 2.6.

$$f : \frac{\chi_{SG}^2}{\chi^2} \tag{2.6}$$

χ_{SG}^2 is the sum of squared residuals of the experimental spectrum and the simulated spectrum smoothed by applying a Savitzky-Golay filter.

Selection is done by maximizing the fitness value found by the above formula. A more in depth discussion of the selection step can be found in Section 3.5.2.

3

Optimization

This chapter will look at the optimization part of the research. First we discuss the motivation of this research in terms of optimization. Next, we look at the requirements for IBA optimization. Then we discuss the chosen Python library. After we have chosen a library, we look at the mutation and selection step of the algorithm. Next, we discuss at how multi-spectra optimization is implemented and we end the chapter with an explanation of the technique we also compare our approach to, namely simulated annealing.

3.1. Motivation

Precise material characterization at microscopic level is crucial in the semiconductor industry. Ion beam analysis, as a technique, allows analysis at this precision. This research build further on the work done by Heller et al. [5] in which they proposed an approach for IBA where differential evolution is used. By moving to a well-tested and widely used open-source library for differential evolution, this study aims to further improve the performance and usability of DE in the context of IBA. In principle, multiple composition depth profiles may lead to an identical RBS spectrum. We hypothesize that moving to multi-objective DE optimization by adding multiple experimental spectra would limit the search space and increase the precision to give better results.

This challenge is a necessary step to meet the ever-increasing demands of material characterization at microscopic level. By pushing the boundaries of existing techniques, this research work helps build the characterization tools of the future, essential for contiuous advancements in technology and research.

3.2. Requirements

The optimization algorithm of Ruthelde is written from scratch in Java and is thus potentially not as efficient as well-tested and used open source libraries. The results by Heller et al.[5] are very promising. This research builds on top of the work by Heller et al. by implementing differential evolution using a popular open source library. The first requirement is thus that the optimization is done by a well-tested and popular open source library that supports differential evolution. Being popular and open source makes the library less prone to bugs and performance bottlenecks.

The second requirement is that the programming language used is already known by the author and allows for fast development. Time on this research work is limited and avoiding the need to learn a language from scratch allows for more time for experiments. The possibility for fast development in the language also allows for more time elsewhere in this work.

The next requirement is the possibility to modify the DE algorithm to the requirements of the problem. The mutation step will be modified to create mutation vectors that better suit our problem. The fitness function in the selection step is also problem-specific and for this problem it requires the integration of a IBA simulator solver.

3.3. SciPy

Based on the previous requirements, Python library SciPy[1] [15] is chosen. SciPy is a libary that, among other things, provides algorithms for optimization. SciPy wraps highly-optimized implementations written in low-level languages, like C. This offers the speed of compiled code with the flexibility of Python. The library is open source and publicly maintained on GitHub by an active community. SciPy supports differential evolution natively using a dedicated function call. The mutation strategy and fitness function are fully customizable and allow the integration of a IBA simulator.

3.4. Vector parameters

Parameter choice is a vital part of solving a problem with differential evolution. It is essential that only the necessary parameters are selected while still incorporating all essential parameters to solve the problem. For IBA, the vector parameters can be split in two groups: target material parameters and experiment parameters.

Target material parameters refer to the variables IBA aims to determine. The main goal is to find the areal densities of elements in a target layer. The areal density refers to the amount of atoms per square cm. In Figure 3.1, a thin layer with strontium, titanium and oxygen is shown on top of a Silicon substrate. For this example, the target is to find the areal densities of strontium, titanium, and silicon. These areal densities can be found by the multiplication of the ratio of each element with the total areal density of the SrTiOx layer. The DE parameters for the target material would then consist of the areal density of the SrTiOx layer, the ratio of strontium, the ratio of titanium, and the ratio of oxygen.

Figure 3.1: Thin film of SrTiOx with Si substrate

Experiment parameters refer to the variables that link to IBA experiment setup. These 4 parameters need to vary between predefined bounds to allow DE to converge to the correct target material parameters. The first parameter is the total applied beam charge (μC). The detector calibration factor and offset are the second and third parameters. The detector resolution is the last parameter.

There is an improvement to be made in the amount of parameters necessary. The ratio of each element but one can also be substituted by using proportions to other elements. This further reduces the amount of DE parameters by 1 for each material layer. For the SrTiOx layer in Figure 3.1, we could represent the ratio of strontium as parameter A and the ratio of titanium parameter as B. Then the ratio of oxygen, C, can be found using equation 3.1.

$$C = \begin{cases} 1 - A - B & \text{if } 1 - A - B > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

Using these parameters for the SrTiOx layer, the amount of parameters necessary for DE optimization would be 1 less than before.

3.5. Differential evolution

3.5.1. Mutation

The mutation step is an important step in the DE algorithm. In this step, new solutions are generated by combining existing solutions of the current generation. Scipy offers 12 different mutation strategies to choose from. Qiang et al. [9] outlined all available strategies in detail. This research chooses and compares 3 of the most promising strategies for the ion beam analysis problem.

[1] https://github.com/scipy/scipy

During each generation, for each target vector i $X = \{x_1, x_2, ..., x_P\}$ a new mutant vector is generated by following the strategy. We chose the following mutation strategy equations.

$$rand1bin : V_i = X_{r1} + F_{xc} * (X_{r2} - X_{r3}) \tag{3.2}$$

$$best1bin : V_i = X_b + F_{xc} * (X_{r1} - X_{r2}) \tag{3.3}$$

$$randtobest1bin : V_i = X_{r1} + F_{cr} * (X_b - X_i) + F_{xc} * (X_{r2} - X_{r3}) \tag{3.4}$$

F_{xc} is a real scaling factor that controls the differential evolution variation. F_{cr} is a weigt for the combination between original target vector and the best parent vector. X_b is the best solution among the current population. r1, r2, or r3 are chosen randomly from the interval [1, P].

The first strategy, rand1bin, is the most widely used mutation strategy proposed by Storn and Price [13]. The second strategy, best1bin, is the default strategy used by Scipy. This strategy takes advantage of the best solution found in the previous generation. Best1bin allows for a faster convergence towards the optimal solution according to Mezura-Montes et al. [8]. The last strategy, randtobest1bin, compromises between exploitation of the best solution while also exploring the parameter space. By choosing a randomly selected parent vector instead of the current parent vector, larger diversity should be achieved.

In Section 5.2, all 3 aforementioned strategies are compared in an experiment on a sample material. The most suitable mutation strategy is chosen and used in further experiments.

3.5.2. Selection

In the selection phase, a decision is made which solutions are kept for the next generation. Solutions are compared by feeding the DE parameters in a fitness function. The vector with the smallest fitness value is the best solution and will be kept for the next generation. The fitness function is problem specific and influences for a large part the runtime of the DE algorithm. The fitness function used is very similar to the fitness function used by Heller et al. [5].

For IBA, the fitness will be calculated by comparing the original spectrum measurement to a simulated spectrum measurement. This simulated measurement is found by feeding the DE parameters to an IBA simulator. In Figure 3.2 the simulation process is shown. The DE parameters, together with a list of fixed experiment and material specific values, is sent to the simulator. The simulator analytically calculates the IBA spectrum according to the underlaying physical processes. Developing our own IBA simulator is out of scope for this project. Ruthelde, developed by Heller et al. [5] is used as a simulator using a WebSocket connection.

Figure 3.2: IBA simulation

The fitting is achieved by minimizing the sum of squared residuals (χ^2) between the simulated and the experimental spectra. Equation 3.5 is used for this calculation.

$$\chi^2 = \sum_{i=0}^{N} \left(x_i^{experimental} - x_i^{sim} \right)^2 \tag{3.5}$$

N is the amount of energy channels, this is typically 1024. $x_i^{experimental}$ is the counts of the ith channel of the experimental spectrum. x_i^{sim} is the counts of the ith channel of the simulated spectrum.

Savitzky et al. [12] proposed the Savitzky-Golay filter, a mathematical technique to smooth a list of data points. This increases the smoothness of the data without distorting the signal tendency. A Savitzky-Golay filter of width equal to the detector resolution is applied to the original spectrum. The squared

residuals between the filtered original spectrum and the simulated spectrum is calculated using equation 3.6.

$$\chi^2_{SG} = \sum_{i=0}^{N} \left(SG(x_i^{experimental}) - x_i^{sim} \right)^2 \tag{3.6}$$

N is the amount of energy channels, this is typically 1024. $x_i^{experimental}$ is the counts of the ith channel of the original spectrum. x_i^{sim} is the counts of the ith channel of the simulated spectrum. SG is the Savitzky-Golay filter function which is used for smoothing the original spectrum.

The sum of squared residual for a detector with 1024 channels can be very large and is dependent of the noise and shape of the experimental spectra. A normalized fitness function is needed that makes the fitness more intuitive. Fitness equation 3.7 ensures normalization of the data.

$$f : \frac{\chi^2_{SG}}{\chi^2} \tag{3.7}$$

The Savitzky-Golay filter ensures a very smooth representation of the input data points. The squared residuals of smoothed data is inherently smaller than not smoothed data. χ^2_{SG} is thus always smaller than χ^2. A good solution has a higher fitness value $\frac{\chi^2_{SG}}{\chi^2}$ than a worse solution. Because Scipy only minimizes fitness functions, the fitness function for minimization is the inverse. The fitness function used for single spectra optimization in this research is equation 3.8.

$$f : 1 - \frac{\chi^2_{SG}}{\chi^2} \tag{3.8}$$

3.5.3. Multi spectra optimization

Going from single spectra to multi spectra optimization requires changes in the amount of DE vector parameters and the fitness function. In multi spectra optimization, there is more than 1 experimental spectra for the same target material. This extra spectra gives the optimization algorithm additional information to find the optimal solution. For each additional spectra, there are 4 additional experiment parameters which have to be optimized.

The selection step requires some changes in multi spectra optimization. For each experimental spectra, a simulated spectra is generated using the IBA simulator. Such as in single spectra optimization, the inverse normalized squared residuals are used to find the fitness of a solution. The total fitness in multi spectra optimization is the average fitness of all spectra. Equation 3.9 is used to calculate the fitness in multi spectra optimization for n spectra.

$$f : 1 - \frac{1}{n} \sum_{i=0}^{n} \left(\frac{\chi^2_{SG_i}}{\chi^2_i} \right) \tag{3.9}$$

An important thing to note is that the amount of simulations needed in multi spectra optimization scales linearly with the amount of spectra. When we compare single with multi spectra optimization using n spectra, single spectra is $\mathcal{O}(1)$ and multi spectra fitting is $\mathcal{O}(n)$. This means that there is 1 simualion per individual for single spectra and n simulations per individual for multi spectra optimization. IBA simulation is relatively time consuming thus there has to be a balance in the amount of extra information we want to add and how much time it will take to simulate all spectra.

3.6. Simulated Annealing

Simulated annealing (SA) is a probabilistic technique for finding the global minimum of an objective function. Simulated annealing is inspired by the process of annealing in metallurgy, where heating and cooling of a material is used to reduce defects. Simulated annealing mimics this process by allowing random variations in the solution with a decreasing probability of accepting suboptimal solutions over time. This makes the algorithm good for avoiding local optima and finding the global optimum.

Simulated annealing uses a temperature parameter that gradually decreases using a cooling schedule. A high temperature makes the algorithm more likely to accept suboptimal solutions which helps

for exploration of the search space. When the temperature parameter decreases, fewer suboptimal solutions get accepted. This makes the algorithm better for exploitation of good solutions and reduce the search space at the end of the algorithms runtime.

The objective of the simulated annealing process is to find a state s where the function $E(s)$ is to be minimized. The goal is to go from an arbitrary inital state to the state with the minimum possible energy.

Each iteration, the simulated annealing algorithm considers a neighbouring state s^* and decides with a probability to move there or not. This step is then repeated until a good enough solution is found or a maximum amount of iterations is reached.

Standard simulated annealing in SciPy is deprecated since 2014. For our project we use the Dual Annealing optimization method from SciPy which replaced simulated annealing. This approach combines the generalization of Classical Simulated Annealing and Fast Simulated Annealing coupled to a strategy that does local search on accepted locations [17]. The algorithm uses a distorted Cauchy-Lorentz visiting distribution with its shape controlled by paramter q_v calculated using equation 3.10. This visiting distribution is used to generate a trial jump distance $\Delta x(t)$ of variable $x(t)$ under artificial temperature T_{q_v} for an artificial time t. For an in depth explanation of this distribution we refer to the work done by Xiang et al. [18].

$$g_{q_v}(\Delta x(t)) \propto \frac{[T_{q_v}(t)]^{-\frac{D}{3-q_v}}}{\left[1+(q_v-1)\left(\frac{\Delta x(t)}{T_{q_v}(t)}\right)^2 \frac{2}{3-q_v}\right]^{\frac{1}{q_v-1}+\frac{D-1}{2}}} \tag{3.10}$$

Next to the used distribution, 2 main equations are important for this algorithm. These are the chance of acceptance and the cooling schedule.

3.6.1. Chance of acceptance

The chance P that a new solution is accepted, even if the solution is worse than the current solution is given by equation 3.11.

$$p_{q_a} = \min\left\{1, [1-(1-q_a)\beta\Delta E]^{\frac{1}{1-q_a}}\right\} \tag{3.11}$$

ΔE is the difference in energy between the current and next state. β is the Lagrange parameter and q_a is the acceptance parameters coming from the Cauchy-Lorentz distribution in equation 3.10.

For $q_a < 1$ zero acceptance probability is assigned to the cases where $[1-(1-q_a)\beta\Delta E] < 0$.

The energy e used for ΔE is determined using the fitness function from the DE algorithm. The fitness function is calculated using equation 3.8.

Equation 3.10 and 3.11 ensure that at a high temperature T_{q_v}, more suboptimal choices are accepted. When temperature T_{q_v} decreases, the chance of acceptance of a suboptimal solution is low.

3.6.2. Cooling schedule

The cooling schedule determines the speed of decrease of the temperature parameter T_{q_v}. Equation 3.12 shows the cooling schedule used by the Dual Annealing optimization method of Scipy.

$$T_{q_v}(t) = T_{q_v}(1)\frac{2^{q_v-1}-1}{(1+t)^{q_v-1}-1} \tag{3.12}$$

$T_{q_v}(1)$ is the starting temperature, t is the artificial time, and q_v is the visiting parameter.

4

Web application

This chapter will look at the web application part of this research. First the motivation and requirements of the web application are defined. Then, a technology stack is chosen that adheres to the requirements. Next, different interesting software engineering design choices are elaborated further on. Finally we discuss the DevOps choices that help to validate and deploy the software package.

4.1. Motivation

In chapter 3, the need for efficient optimization techniques in IBA is substantiated. For an optimization technique to be useful, it needs to be configurable and usable in practice. This research builds further on the work done by Heller et al. [5] on Ruthelde. The codebase of Ruthelde is too complex and has a lot of component dependency which makes it hard to build new features. Extending and altering the actual Ruthelde code is out of scope for this project. We thus have decided that building a new application will be more time efficient. As such, we could tackle some challenges that Ruthelde currently has.

Ruthelde is a desktop application developed in Java. This has as a consequence that the application is limited to the resources available on the users machine. Having a desktop application also has the problem that there is an install procedure necessary. The user has to install the application on his machine and has to have all necessary data available to him at any time. Another problem of a desktop application is that the machine has to be on for the full duration of an experiment. This could pose difficulty for complex problem instances that take a long time.

According to finding by Gartner [3], worldwide cloud services are forecast to grow by 19.6% each year until 2027. There is a broader trend in software development to shift towards cloud computing and platform independent solutions. In this work, a step towards these new technologies can be made to make IBA methods more up to date to the current state of the art in software engineering.

4.2. Requirements

The graphical user interface that we will build to communicate with the optimiziation models has some important requirements.

1. The first requirement is that it has to be a web application. This eliminates the requirement for the user to install the application before use. A web application also has the advantage that the application can be used from more devices and theoretically anywhere in the world where there is an internet connection. A web application also enables the user to have the necessary data on a shared server which makes it less likely to get lost and everyone has the same version of the data.
2. Another requirement is that the amount of boilerplate code is minimized. There is a fixed amount of time for this research work and we should aim for a framework that works as efficiently as possible.

3. The next requirement is that the final software package should be easy to install and use. This research work requires dependencies such as e.g. an IBA simulator. The aim for this project should be to set up the final product as such that install on a server is easy and straightforward.
4. Next, the web application should be compatible with the file types and structure as Ruthelde. This enables researchers to use their data from Ruthelde in this web application.
5. Finally, the web application should be easy to work with. The amount of bugs should be reduced as long as time is sufficient. Unit tests or test coverage could help achieve this goal.

4.3. Streamlit

Streamlit[1] is a Python package that turns Python data scripts in a shareable web application. Streamlit is developed and maintained in an open source nature on GitHub. The package emerged as a tool for the development of web applications at the intersection of data science and software engineering. Streamlit allows the developers to create web applications with minimal effort and boilerplate code, accelerating the transition from data analysis to an interactive application.

The decision to use Streamlit for this project was driven by its ability to streamline the development process by keeping everything in Python. With Streamlit, the complexity of building a web application is reduced so most of the focus of this project can be on the optimization part rather then being slowed down by web development details.

The first advantage of Streamlit is the ease of use. The straightforward syntax and simple workflow makes it accessible for the developer or data scientist. Another advantage is the large library of quality components made by the community such as e.g. sliders and buttons. Finally, Streamlit allows for rapid prototyping enabling quick iterations and new features.

Based on the requirements defined in section 4.2, Streamlit is a perfect fit. The library is made to build web applications, does not require a lot of boilerplate, and is easy to work with. The library supports our needs seamlessly and easily integrates with other Python data science libaries to enable advanced features.

In Appendix C, screenshots of a subset of pages of our Streamlit application can be found.

4.4. Docker

Docker has revolutionized the way developers package, deploy, and manage applications by encapsulating them in containers. These containers are lightweight, standalone, and executable software packages that include everything needed to run the application.

This project is complex, containing a lot of dependencies. The project consists of a web application built using Streamlit and other Python libraries, the Ruthelde executable built in Java that is used for IBA simulation, and a DE algorithm built using SciPy and other libraries. This diverse collection of dependencies and programming languages requires a complex environment. We chose to encapsulate each complex part of the project in a Docker container. Docker Compose is used to orchestrate and connect the containers. The robust environment isolation capabilities are essential for achieving consistency across development, testing, and production environments. By using Docker and Docker Compose, we ensure that our web application and associated services run seamlessly regardless of where it is deployed.

We define three containers as can be seen in Figure 4.1. The first container encapsulates the Streamlit web application and core Python logic. That container is the only container accessible to the user and the default access port is port 8501. The second container contains the Ruthelde server application. This container is accessible on port 9090. Finally, the DE container contains the DE optimization algorithm implemented using SciPy. The DE container is accessible on port 9080. There is a direct connection between the DE and Ruthelde container to perform IBA simulations on Ruthelde to aid the DE algorithm. Connections between containers are set up using WebSockets, offering two-way communication at all times. All shared data is stored in the files directory which is mounted to each container to allow for shared access.

[1] https://github.com/streamlit/streamlit

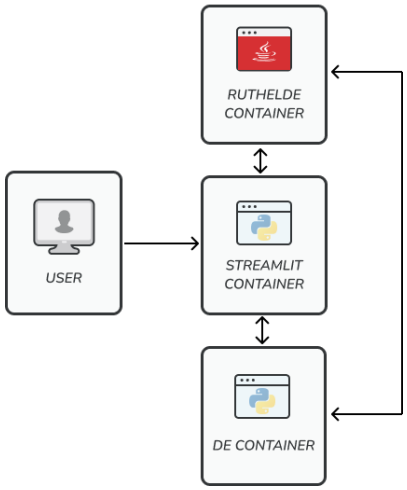


Figure 4.1: Container architecture diagram

Docker has been a an appropriate choice for this research project. It allows the user to install the software without worrying about dependencies and installations. A great advantage of Docker in this project is that the web application and optimization models are divided. When the user closes the web application or Streamlit crashes, the optimization on the other containers can continue without a problem. Docker not only simplified the development workflow but also ensured that our application is portable, scalable, and consistent across all deployment environments.

4.5. Ruthelde integration

In section 3.5.2, the IBA simulator from Ruthelde is discussed. Our DE optimization algorithm needs to integrate the IBA simulator from Ruthelde to evaluate possible solutions. In this section, the different methods of connecting the DE and Ruthelde containers that were tried are discussed.

4.5.1. Command line

The first method uses the command line functionality of Ruthelde. Using the subprocess libary in Python, the following terminal command is executed that starts the Ruthelde Client jar using command line options.

```
$ java -jar Ruthelde_Client.jar requestType input.json output.json ip 9090
```

requestType depends on the desired functionality and can be SIMULATE, OPTIMIZE or OPTIMIZE_MS, input.json is the path to the input file, output.json is the path to the output file, ip is the ip address of the machine the server is running on, and 9090 is the port the server is listening to.

There are some disadvantages with this method. The Java client has to exist in the same container as the Python code and have access to the input file. The duo setup of Java and Python makes the container more complex and larger in size. Another disadvantage is that if the Python container crashes, the Ruthelde Client also crashes and the server does not have a return point to send the result to. Finally, with this method there is no way of tracking optimization or simulation progress from the Python application. After the command is executed, there is no connection between Python and Ruthelde Client.

4.5.2. WebSockets

In the Streamlit application it is necessary to track the progress of a simulation or optimization. This is not possible with the command line method as there is not a fixed connection between Python and Ruthelde Client. We chose to investigate the possibility of replacing the Ruthelde Client with a Python client. This Python client will directly connect to Ruthelde Server using a WebSocket.

WebSockets provide a full-duplex communication over a single, long-lived connection. This allows real-time data exchange between a client and a server. Unlike with HTTP, where the client has to start all

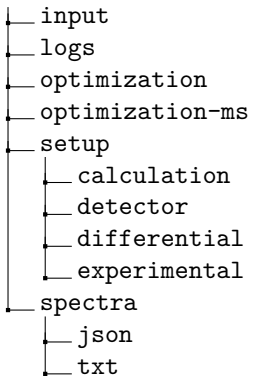
requests, in WebSockets the server can push data to the client when it comes available. The client does not have to request data repeatedly. For our project, WebSockets are a good choice because the server would then push the progress or intermediate data to the client when it becomes available.

In the public codebase of Ruthelde [10], it can be seen that ObjectOutputStream and ObjectInputStream methods are used to communicate with Ruthelde Client. Unfortunately, the code for Ruthelde Client is not public. After reverse engineering the Ruthelde Client and trying numerous Python libraries such as but not limited to pickle, multiprocessing.Pipe, python-javaobj we learned that Java uses a unique protocol for ObjectOutputStream and ObjectInputStream. Back and forth connection from Python to Java ObjectOutputStream and ObjectInputStream is at the time of this research, an unsolved problem but there is some progress done by developers on Github [14].

Luckily, René Heller, the main developer of Ruthelde [5], agreed to make a change to the server so a WebSocket connection from a different programming language is possible. The socket library from Python communicates to Ruthelde Server from the Streamlit or DE containers.

4.6. Storage setup

In our web application we need a way of storing data and information. Our application has to be compatible with Ruthelde data files and communication to the Ruthelde simulator module is through json files. Based on this information we chose to store data using flat-file storage. In this approach, data is stored in text or json files rather than in a structured database like SQL. Flat-file systems are easy to setup, simple to maintain, and have lower overhead. Flat-file storage is a great choice when advanced features of SQL databases are not a requirement. This method allows us to efficiently store data in the same format as Ruthelde. As data is stored in files, it is easy to implement a upload or download functionality in the web application.



In above diagram, the file structure is visualised. In the input directory, the data files are temporarily stored which will be sent through Ruthelde for a simulation or optimization. The logs directory stores logs for optimizations using the DE container. In the optimization and optimization-ms directory, all result data is stored after respectively a single or multi spectra optimization. In the setup directory all configuration files are stored necessary to start a simulation or optimization. The configuration files are split in there different sub categories so the user can create and combine different setup configurations. Finally, in the spectra directory experiment spectra and simulated spectra data is stored grouped by their file type being json or txt.

4.7. DevOps

In a software engineering project there should be an emphasis on validation of the codebase. The components and methods should do what they were implemented for. Tests are an important measurement method to validate functionality in code.

In this project we chose to use pytest[2] as the main testing framework. This project has external dependencies such as the file system and Docker containers. Therefore, we used unittest.mock to mock

[2] https://github.com/pytest-dev/pytest

these dependencies. Mock objects are simulated objects that mimic the behaviour of real dependency objects in controlled ways. This way we can focus on the code being tested and not on the behavior or state of external dependencies. With the coverage library, we can see which functionality is tested and generate a coverage report. The coverage report can be found in section 5.7.

The codebase and used data is stored in a public repository on GitHub [7]. GitHub actions runs all the pytest tests at every commit or pull request. This ensures no breaking changes are made in the development process. The repository also includes install documentation and relevant shell scripts to help the user install and use the software package.

5

Experimental Results

This chapter will talk about the experiments performed on the optimization models. We start with an experiment on the best1bin, rand1bin, and randtobest1bin mutation strategies proposed in chapter 3. Next, single spectra optimization experiments are performed on different materials. After, experiments on the multi-spectra optimization approach are conducted and discussed. Next, an experiment on the simulated annealing technique is discussed. Finally all previous results are compared to the results from Ruthelde [5]. This chapter ends with a discussion of the tests performed on our code and the test coverage.

All experiments are conducted on a 2020 Mac Mini with an 8-core M1 CPU and 8GB of RAM.

5.1. Metrics

When we want to evaluate the performance of algorithms, good comparison metrics are crucial. Before we discuss all experiments, we will first define the metrics used. We chose these metrics because they are also used in the work done by Heller et al. [5]. The use of the same metrics makes the comparison fair.

5.1.1. Standard deviation

Standard deviation is a metric for the spread of a set of values around the mean. It indicates the degree of consisteny of results between runs. A low standard deviation signifies high reliability of the algorithm. This is important for complex problems with large search spaces like in IBA. Consistency is key for the reliability of optimization results. The standard deviation can be calculated using equation 5.1.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}} \tag{5.1}$$

σ is the standard deviation. N is the number of values. x_i is the ith value. $\bar{x}$ is the mean of all values. In a standard deviation graph, the standard deviation for some vector parameters is visualised over the number of evaluations for an experiment. Ideally, the standard deviation of a parameter converges to 0, which means that all runs have identical parameter values at that point in time. It is important to note that when calculating the standard deviation at a specific function evaluation, terminated experiment runs at that point do not get included.

5.1.2. Fitness

Fitness is the next metric that is used for comparison in the chapter. Fitness is a measure of how well a particular solution meets the objective of the optimization problem. In DE, fitness is used to assess the quality of solutions within a population. A lower fitness value means that we have a better solution, The definition of the fitness function is problem specific and further elaborated on in section 3.5.2 and 3.5.3. In a fitness graph, the average fitness over all runs is visualized in function of the amount of

function evaluations. Ideally, the fitness shows a downward trend when function evaluations increase.

5.1.3. Survival rate

A faster optimization algorithm that finds an optimal solution is preferable. There should be a balance between speed of convergence and fitness. The survival rate metric shows the time it takes for an algorithm for each run to terminate. A run stops when a (sub)optimal solution is found. Ideally, all run instances terminate before the predefined maximum amount of iterations. When a run is still converging but exceeding the predefined maximum amount of iterations, the algorithm is too slow to find the optimal solution. In a survival rate graph, the amount of running instances in function of the amount of function evaluations is shown. Ideally, all instances terminate before the maximum amount of iterations is reached.

5.1.4. Simulated spectra

The last metric to evaluate a solution is a spectra simulation plot. In this plot, the original experiment data is visualised in function of the counts. On top of that, the simulated spectra data for the best solution found is generated and plotted on the same graph. We can then visually compare the experimental and simulated spectra for inconsistencies. Ideally, the simulated data perfectly aligns with the experimental data.

5.2. Mutation strategy

In the mutation step in the DE algorithm, new solutions are generated by combining existing solutions of the current generation. The choice of a mutation strategy is important and there are a lot of strategies to choose from. Below, three viable mutation strategy options are defined. In the following experiments, we will compare each of these strategies using the standard deviation, fitness, and survival rate. Finally, we compare the results and decide on the most suitable choice that will be used in further experiments.

For the mutation strategy experiments, a thin film of strontium-titanium-oxide is used on a silicon substrate. This material is visualised in Figure 5.1. It is important to note that these experiments are conducted using the old configuration. This means that the amount of vector parameters is not optimized and contains more parameters than necessary. For these mutation strategy experiments we will optimize 8 DE vector parameters. These being the 4 experiment parameters, the areal density of the strontium-titanium-oxide layer and a separate parameter for each element ratio in the strontium-titanium-oxide layer. For each mutation strategy, 10 experiment runs were conducted. We chose for 10 runs to get a balance between a good average result over multiple runs while also keeping the total experiment time low. The DE parameters and average runtime for each strategy can be found in Table A.1.

Figure 5.1: Thin film of strontium-titanium-oxide with Si substrate

It is important to note that the convergence termination criteria is 0.001 instead of 0.01. This means that the algorithm does not stop as fast if the new solutions hardly improve. The reason for this choice is to have the algorithm reach a large number of generations to better assess the mutation strategy quality. The consequence is that the maximum amount of function evaluations reached for a run is higher than for the other experiments.

5.2.1. Rand1bin

The first mutation strategy tested is the Rand1bin strategy. It is the most widely used mutation strategy and can be calculated using equation 5.2.

$$rand1bin : V_i = X_{r1} + F_{xc} * (X_{r2} - X_{r3}) \tag{5.2}$$

F_{xc} is a real scaling factor that controls the differential evolution variation. r1, r2, or r3 are chosen randomly from the interval [1, P].

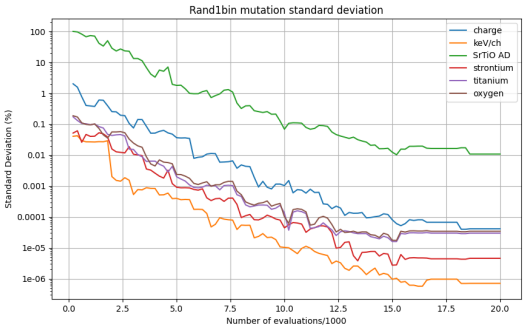


Figure 5.2: Thin film of strontium-titanium-oxide on a silicon substrate standard deviation using rand1bin mutation strategy

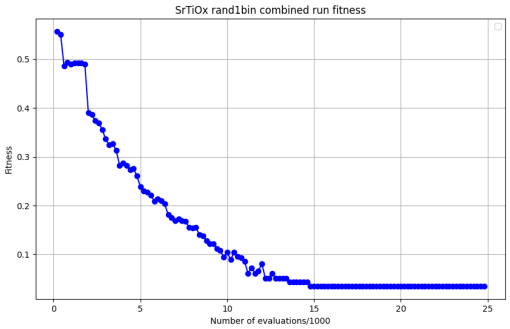


Figure 5.3: Thin film of strontium-titanium-oxide on a silicon substrate fitness using rand1bin mutation strategy

5.2.2. Best1bin

The next strategy, best1bin, is the default strategy used by Scipy. This strategy takes advantage of the best solution found in the previous generation. Best1bin is calculated using equation 5.3.

$$best1bin : V_i = X_b + F_{xc} * (X_{r1} - X_{r2}) \tag{5.3}$$

F_{xc} is a real scaling factor that controls the differential evolution variation. X_b is the best solution among the current population. r1, and r2 are chosen randomly from the interval [1, P].

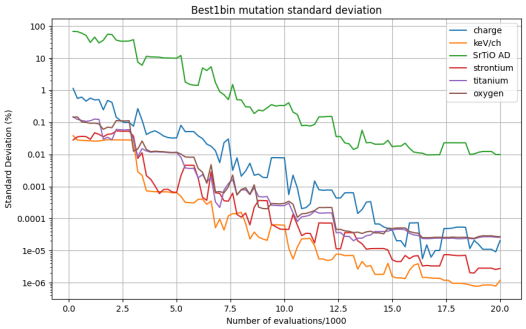


Figure 5.4: Thin film of strontium-titanium-oxide on a silicon substrate standard deviation using best1bin mutation strategy

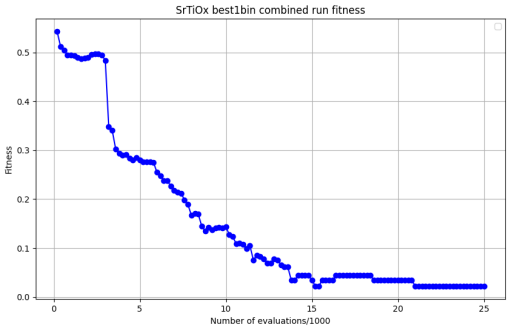


Figure 5.5: Thin film of strontium-titanium-oxide on a silicon substrate fitness using best1bin mutation strategy

5.2.3. Randtobest1bin

The last strategy, randtobest1bin, compromises between exploitation of the best solution while also exploring the parameter space. By choosing a randomly selected parent vector instead of the current parent vector, larger diversity should be achieved. Randtobest1bin can be calculated using equation 5.4.

$$randtobest1bin : V_i = X_{r1} + F_{cr} * (X_b - X_i) + F_{xc} * (X_{r2} - X_{r3}) \tag{5.4}$$

F_{xc} is a real scaling factor that controls the differential evolution variation. F_{cr} is a weigt for the combination between original target vector and the best parent vector. X_b is the best solution among the current population. r1, r2, or r3 are chosen randomly from the interval [1, P].

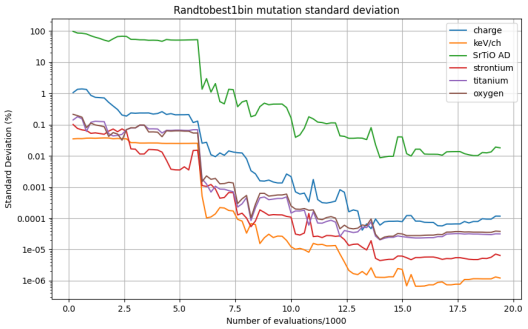


Figure 5.6: Thin film of strontium-titanium-oxide on a silicon substrate standard deviation using randtobest1bin mutation strategy

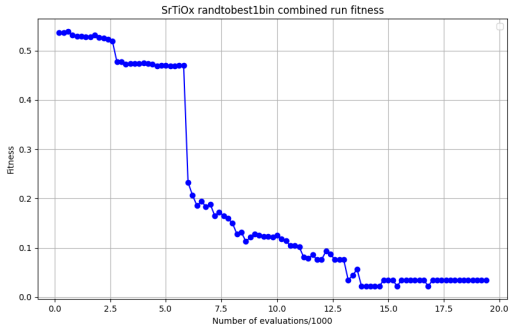


Figure 5.7: Thin film of strontium-titanium-oxide on a silicon substrate fitness using randtobest1bin mutation strategy

5.2.4. Survival rate

Such as discussed in section 5.1.3, a faster optimization algorithm that finds an optimal solution is preferable. Figure 5.8 shows the survival rate of each mutation strategy.

Figure 5.8: Thin film of strontium-titanium-oxide on a silicon substrate mutation strategies survival rate

5.2.5. Discussion

If we compare the standard deviation graphs of the mutation strategies in Figures 5.2, 5.4, and 5.6 we can conclude that all strategies converge quite well. The randtobest1bin strategy in Figure 5.4 gives the worst result. The rand1bin and best1bin standard deviations follow a comparable trend with the rand1bin strategy in Figure 5.2 giving the best result.

All mutation strategies have a fitness that converges to a sufficient value which can be seen in Figure 5.3, 5.5, and 5.7. Again, randtobest1bin has the worst convergence. Rand1bin and best1bin follow the same trend and both perform well. The rand1bin and randtobest1bin mutation strategies both converge to a fitness value of 0.0346 . The best1bin mutation strategy converges to a fitness value of 0.0218 . This means that the best1bin strategy found on average the best solution.

If we look at the survival rate in Figure 5.8, we can conclude that best1bin finds a solution the fastest. An interesting observation to be made is that the rand1bin strategy seems to have a hard time to find a sufficient solution and terminate the algorithm. The randtobest1bin strategy lays somewhere in the middle in terms of survival rate.

The best1bin strategy is the default mutation strategy in SciPy which makes it an often used and tested strategy. Research by Mezura-Montes et al. [8] proves that the best1bin strategy allows for a faster

convergence towards the optimal solution compared to other strategies like the rand1bin and randtobest1bin strategies. The best1bin strategy has a slightly worse standard deviation trend then rand1bin in our tests but in terms of fitness it is comparable and converges to a better solution. The survival rate metric shows that best1bin finds a solution the fastest of all the tested strategies. Based on all this information, the best1bin strategy is chosen to be the best choice for further experiments.

5.3. Single spectra optimization

In this section, we test our DE algorithm for single spectra optimization on different material configurations. All DE experiments are conducted using the best1bin strategy, which was proven to be the best choice in the previous section. For each material we ran the experiment 20 times to have a good and representative average result.

5.3.1. Au SiO2 Si

The first material we want to determine the material composition for is a gold film on a silicon-oxide layer on a silicon substrate. This material is visualised in Figure 5.9. We want to find the areal densities of both the gold film and the silicon and oxygen of the silicon-oxide layer. We have 7 DE vector parameters being the 4 experiment parameters, the areal density of gold, the areal density of the silicon-oxide layer and the ratio of oxygen. We performed 20 experiment runs for this material which took an average of 6 minutes each. The DE parameters used can be found in Table A.1.

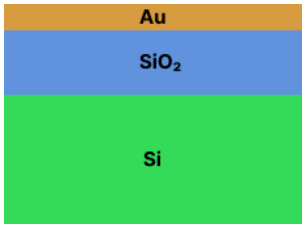


Figure 5.9: Gold film on a silicon-oxide layer on a silicon substrate

Figure 5.10: Gold film on a silicon-oxide layer on a silicon substrate standard deviation

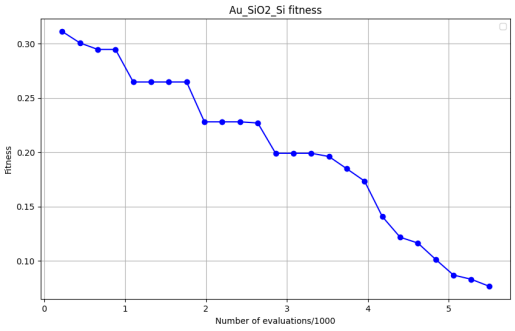


Figure 5.11: Gold film on a silicon-oxide layer on a silicon substrate fitness

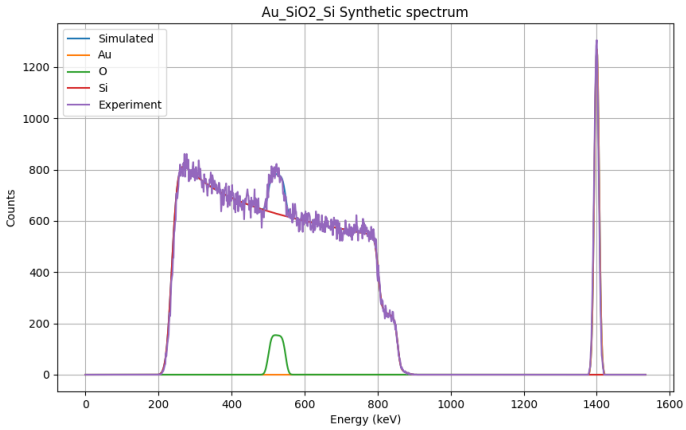


Figure 5.12: Gold film on a silicon-oxide layer on a silicon substrate synthetic spectrum

In Figure 5.10, the standard deviation of the charge, calibration factor, gold areal density, and oxygen areal density are plotted. We can see that all variables converge to a lower standard deviation. The algorithm has a relatively hard time finding the areal density of the silicon-oxide layer. In Figure 5.11, the average fitness of all runs is shown in function of the amount of function evaluations. We can see that the algorithm is able to minimize the problem efficiently and it converges to a small fitness value. The simulated spectrum generated using an optimal solution of one of the runs is plotted in fig 5.12. It can be seen that the simulated spectrum follow the experiment spectrum, even at difficult points like e.g. at energy level 1400.

From these metrics we can conclude that the DE algorithm finds a good solution for this material in sufficient time. The predefined maximum amount of iterations is never reached, all vector parameters converge to the solution over time and the fitness slowly decreases.

5.3.2. SrTiOx Si

The second experiment will assess the performance of our DE algorithm on a thin film of strontium-titanium-oxide on a silicon substrate. This material is visualised in Figure 3.1. The objective is to find the areal densities of strontium, titanium, and oxygen in the top layer. We have 7 vector parameters being the 4 experiment parameters, the areal density of the stronium-titanium-oxide layer, and the ratio of strontium and titanium. We performed 20 experiment runs for this material which took an average of 4 minutes each. The DE parameters used can be found in Table A.1.

Figure 5.13: Thin film of strontium-titanium-oxide on a silicon substrate

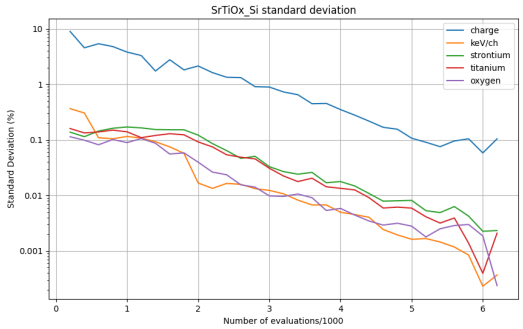


Figure 5.14: Thin film of strontium-titanium-oxide on a silicon substrate standard deviation

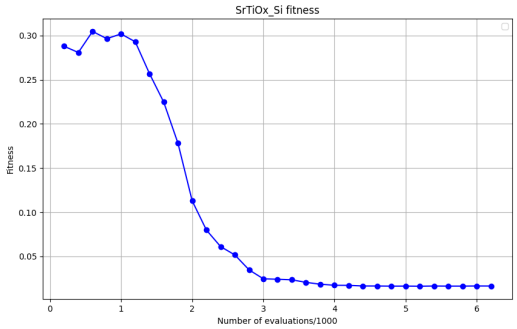


Figure 5.15: Thin film of strontium-titanium-oxide on a silicon substrate fitness

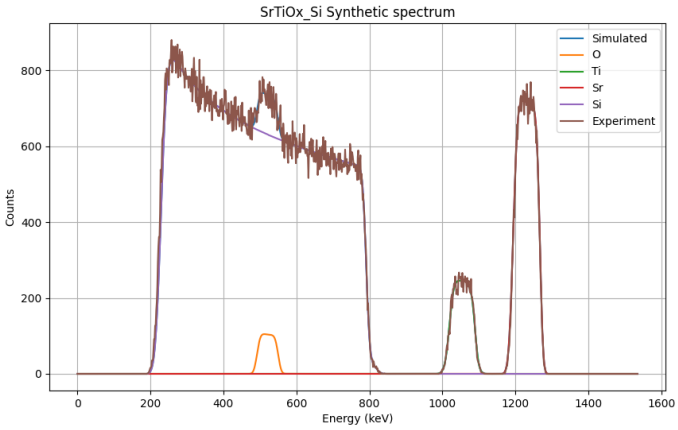


Figure 5.16: Thin film of strontium-titanium-oxide on a silicon substrate synthetic spectrum

The standard devitation, in Figure 5.14, shows a downward trend for all parameters. All parameters except of the charge converge very well to the same value. In Figure 5.15, the average fitness over all runs is shown in function of the amount of function evaluations. We can see that the algorithm efficiently converges to a small fitness value. It can be seen that a good solution is found at 4000 function evaluations and the algorithm does not improve much after. The simulated spectrum is plotted in Figure 5.16. The experiment and simulated spectra follows the same trends, even large spikes are followed.

Based on this observations we can conclude that the proposed DE algorithm works very well on this material instance. The algorithm is able to find a good solution in an acceptable time.

5.3.3. SrTiO3 LaFeO3 SiO2 Si

The final single spectra DE optimization experiment is on a multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate. This multilayer material will test the performance of the proposed DE optimization algorithm in a complex setting. The material is visualised in Figure 5.17. The aim is to find the areal densities of strontium, titanium, oxygen layer 1, lanthanum, iron, oxygen layer 2, silicium, and oxygen layer 3. We have 12 DE parameters being the 4 experiment parameters, the areal density of the strontium-titanium-oxide layer, the areal density of the lanthanum-iron-oxide layer, the areal density of the silicon-oxide layer, and the ratios of strontium, titanium, lanthanum, iron, and silicon. For the experiment we performed 20 runs which took an average of 35 minutes each. The algorithm stops if the maximum amount of funtion iterations of 12000 is reached. The DE parameters can be found in Table A.1.

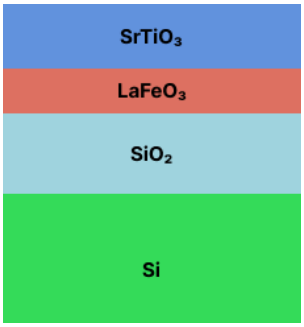


Figure 5.17: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate

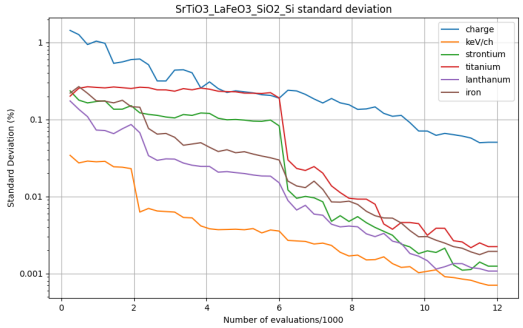


Figure 5.18: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate standard deviation

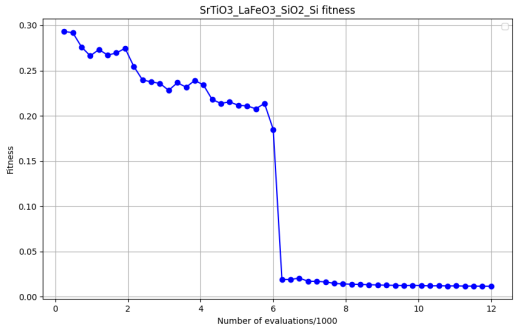


Figure 5.19: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate fitness

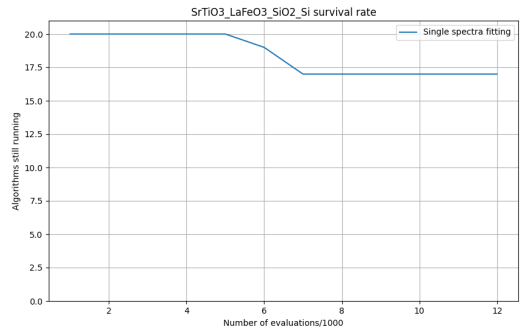


Figure 5.20: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate survival rate

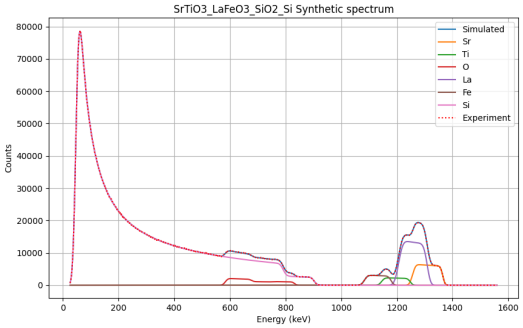


Figure 5.21: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate synthetic spectrum

In Figure 5.18, the standard deviation of the the desired parameters of this problem are plotted. We can see that there is a downward trend for all parameters but the charge lacks a bit behind. An interesting thing to note is the drop at the 6000 function evaluations mark. The fitness, in Figure 5.19, gradually converges which is desirable. The same drop at the 6000 function evaluation mark can also be noticed. Because of the strange behaviour at the 6000 function evaluation mark, I chose to also investigate the survival rate. In Figure 5.20, we can see that around the 6000 function evaluation mark, 3 runs found a solution and stopped the run. These runs were probably stuck in a local minima different from the absolute minimum. This difference in convergence point causes the standard deviation and fitness to be relatively high. When the 3 runs that got caught in the local minima stop their run, the standard deviation and fitness drop significantly. In Figure 5.21, the simulated and experiment spectra are plotted

on top of each other. We can see that the simulated spectra is fitted well and follows the experiment spectra trends very well.

From these observation we can conclude that our algorithm works well for this problem instance. The standard deviation and fitness converge quite well. It is good to mention that only 3 of the 20 runs converged to a sufficient solution themselves. The other runs had to be stopped by the maximum amount of function evaluations stopping condition. If we look at the simulated spectra plot in Figure 5.21, we can see that the proposed approach is able to find a good solution.

5.4. Multi-spectra optimization

For the multi spectra optimization experiment we chose to use a complex material, being a multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate. The material is visualised in Figure 5.17. We have a duo spectra setup where the difference between both spectra is in the incidence angle of the beam. The first spectra has an angle of 170° (alpha), 110° (theta), and 10° (beta). The second spectra has an angle of 20° (alpha), 170° (theta), and 10° (beta). The difference in incident angle of the beam creates a totally different spectra measurement.

The objective is to find the areal densities of strontium, titanium, oxygen layer 1, lanthanum, iron, oxygen layer 2, silicium, and oxygen layer 3. We have 16 DE parameters being 4 experiment parameters for the first spectra, 4 experiment parameters for the second spectra, the areal density of the strontium-titanium-oxide layer, the areal density of the lanthanum- iron-oxide layer, the areal density of the silicon-oxide layer, and the ratios of strontium, titanium, lanthanum, iron, and silicon. For this experiment we performed 20 runs which took an average of 72 minutes. The algorithm stops if the maximum amount of function iterations of 20000 is reached. The DE parameters can be found in Table A.1.

Combining single and multi spectra fitting directly with each other is an unfair comparison. In multi spectra fitting there are more parameters and each generation takes twice or more as much function evolutions as single spectra fitting. As such we decided to also perform 20 runs of a multi spectra setup but with 2 identical spectra measurements. The identical spectra used is the multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate from Figure 5.17 with an incident angle of the beam of 170° (alpha), 110° (theta), and 10° (beta). We hypothesise that multi spectra fitting with different spectra gives a comparable or better result than fitting with identical spectra. This is due to the added information the algorithm receives by including an extra spectra.

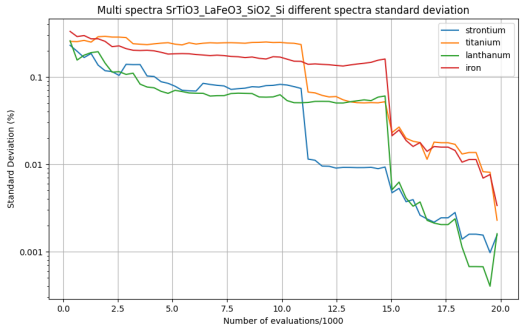


Figure 5.22: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate multi spectra fitting standard deviation

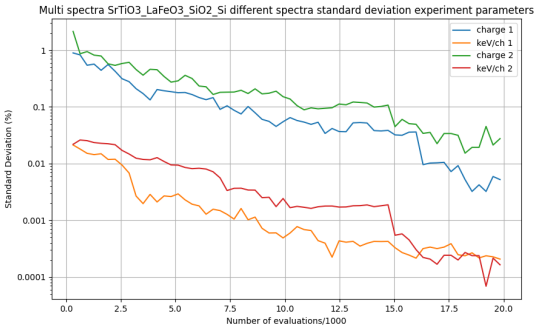


Figure 5.23: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate multi spectra fitting standard deviation experiment parameters

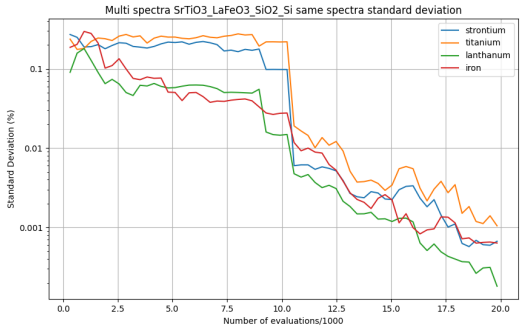


Figure 5.24: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate multi identical spectra fitting standard deviation

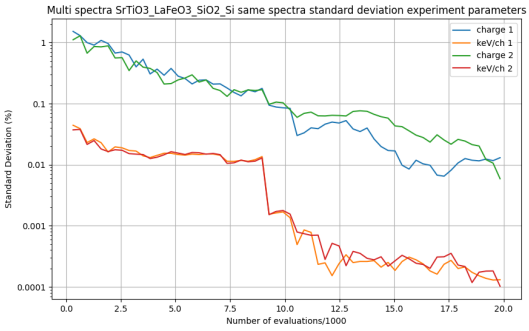


Figure 5.25: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate multi identical spectra fitting standard deviation experiment parameters

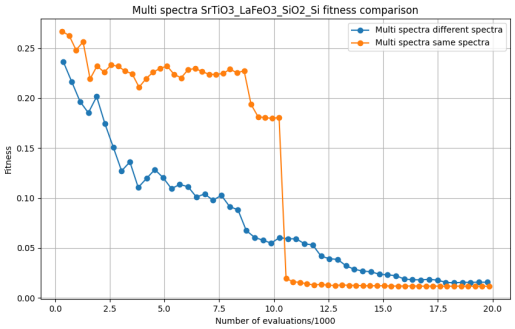


Figure 5.26: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate multi identical and different spectra fitting standard deviation

Figure 5.27: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate multi different spectra fitting standard deviation experiment parameters

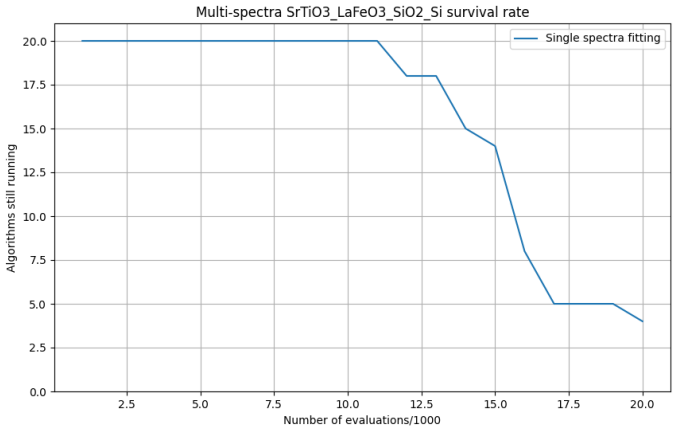


Figure 5.28: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate multi different spectra fitting survival rate

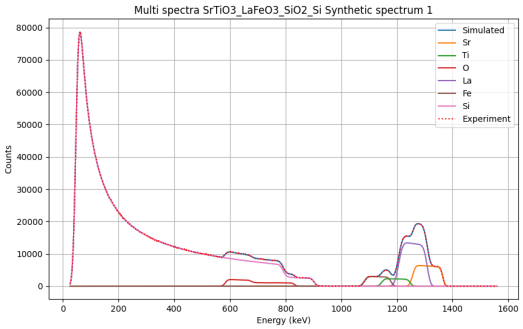


Figure 5.29: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate multi different spectra synthetic spectrum 1

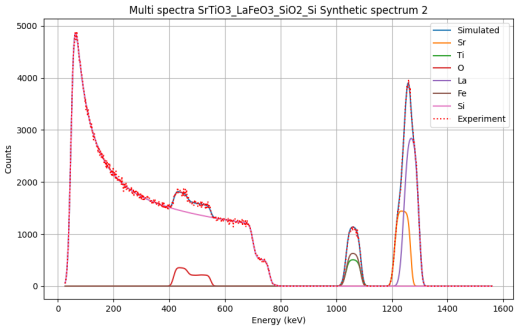


Figure 5.30: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate multi different spectra synthetic spectrum 2

In Figure 5.22 the standard deviation of the elements found using multi spectra fitting with different spectra is plotted over time. At first, the runs seem to have a hard time converging to the same solution. After 10000 function evaluations, the algorithm starts converging and over time find a good common solution. Next, in Figure 5.23 the experiment parameters are plotted for multi spectra fitting with different spectra. Here we can see that the algorithm converges well for both spectra which means that the algorithm is able to use both spectra for finding an optimal solution.

Next, in Figure 5.24 we can see the standard deviation of the elements found using multi spectra fitting with identical spectra. We can see, as with the different spectra optimization that the algorithm has a hard time converging before the 10000 function evaluations mark. After that point, there is a drop and the standard deviation gradually converges to a common solution. In Figure 5.25, the standard deviation of the experiment parameters is plotted. Just as with multi spectra fitting with different spectra, the experiment parameters converge good. Both spectra of the identical spectra fitting converge equally good and help to find an optimal solution.

In Figure 5.26, the fitness of multi spectra fitting of both different and identical spectra is plotted over time. We can see that the fitness of different spectra fitting gradually converges to a very good fitness. The fitness for identical spectra fitting also converges to a very good fitness but does it with bigger drops. After the 10000 function evaluations point, the fitness drops a large amount. This drop is at the same time as the drop in standard deviation as shown in Figure 5.24. This is probably caused by the termination of some runs that were stuck in local minima. In Figure 5.27 the fitness of spectra 1 and 2 of multi spectra fitting with different spectra is fitted over time. Both the fitness of spectra 1 and 2 gradually converge over time and end at a very good fitness. This observation confirms the observation made based on Figure 5.23 that both spectra are used to find an optimal solution.

The survival rate for multi spectra fitting using different spectra, which can be seen in figure 5.28, shows us that the algorithm needs at least 10000 function evaluations to find an optimal solution. Another observation is that only 4 out of 20 runs were stopped based on the maximum amount of function evaluations stopping condition. This means that 20000 is a good stopping value for this experiment.

In Figure 5.29 and 5.30, the simulated and experiment spectra for both spectra of multi spectra fitting with different spectra are plotted on top of each other. The experiment and simulated spectra follow the same trends in both figures. Even large spikes are followed. The artificial noise in spectra 2 does not cause a problem and we can still see a good fit.

Based on these observations, we can conclude that our multi spectra fitting algorithm works as expected. The algorithm is able to find good fits for both spectra and the solutions converge over time. We observed that both spectra are equally used to find the material parameters as both standard deviaton and fitness improves over time. We even see a more gradually decline in fitness of using different spectra than using identical spectra. The addition of a different spectra possibly avoids the algorithm to go in local minima as often as with identical spectra.The simulated spectra follows the experiment spectra very well, even at spikes as can be seen in Figure 5.29 and 5.30. In comparison to multi spectra

fitting with identical spectra, our algorithm has at least a comparable performance in terms of standard deviation and fitness.

5.5. Simulated annealing

An important part of this research is to assess if differential evolution is a good technique for IBA. Therefore, in the following experiment we will test simulated annealing as an alternative. We use a multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide as material so a comparison can be made to the results gathered in section 5.3.3. The material is visualised in Figure 5.17. The objective is to find the areal densities of strontium, titanium, oxygen layer 1, lanthanum, iron, oxygen layer 2, silicium, and oxygen layer 3. We have 12 parameters being the 4 experiment parameters, the areal density of the strontium-titanium-oxide layer, the areal density of the lanthanum-iron-oxide layer, the areal density of the silicon-oxide layer, and the ratios of strontium, titanium, lanthanum, iron, and silicon. The experiment is conducted 20 times which took an average of 58 minutes each. The algorithm stops if the maximum amount of iterations of 20000 is reached. We used the SciPy default initial temperature of 5230. Relevant DE parameters can be found in Table A.1.

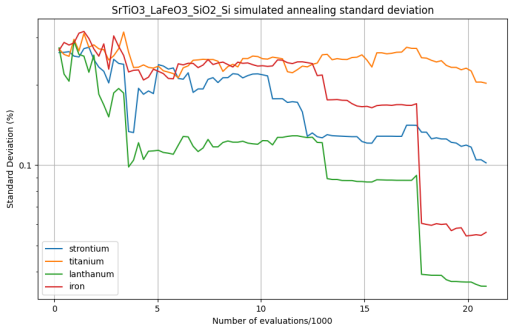


Figure 5.31: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate simulated annealing standard deviation

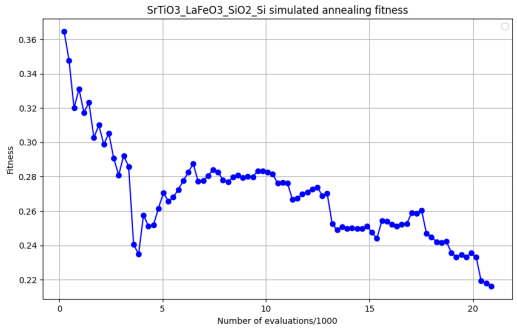


Figure 5.32: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate simulated annealing fitness

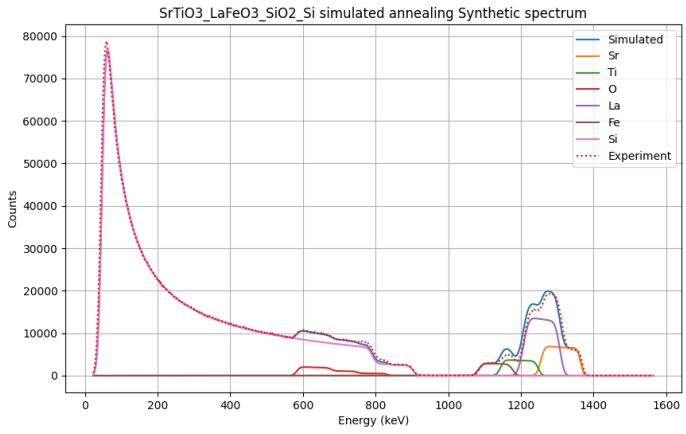


Figure 5.33: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate simulated annealing synthetic spectrum

In Figure 5.31 the standard deviation of the areal densities of strontium, titanium, lanthanum, and iron is shown. The starting standard deviation of all parameters is quite good but there is barely a downward trend over time. Lanthanum and iron converge only at the end to a good standard deviation but strontium and titanium stay rather static over time. In terms of fitness, in Figure 5.32, we can clearly

see a downward trend over generations but it spikes back up and we never reach a fitness below 0.2. This indicates that the algorithm has a hard time converging to the final optimal solution. Finally, in Figure 5.33, the simulated and experiment spectra are plotted on top of each other. We can see that the simulated spectra follows the trends but it sometimes is a bit of such as at energy level 1200.

Following these insights we can conclude that the SciPy implementation of the simulated annealing algorithm performs worse than the DE algorithm. Such as stated in the work by Jeynes et al. [6], simulated annealing works well in the exploration phase at the start of the run. The implementation of SciPy also includes a local search algorithm on accepted locations but it still underperforms in the exploitation phase. The algorithm finds average solutions and keeps improving them. From Figure 5.31 and 5.32 we can confirm this behaviour. The algorithm finds average solutions but the algorithm has a hard time exploiting these solutions later in the run to find the optimal solution. We conclude that the SciPy implementation of simulated annealing is a suboptimal technique to find IBA solutions.

5.6. Ruthelde comparison

This research work uses the work by Heller et al. [5] as a basis. In this section, we see how our approach compares to Ruthelde. It is important to note that the data from Ruthelde is not validated in this research work due to time constraints. As this project is in very close collaboration with the original researchers we saw no need for data validation. Data validation is the process in which we ensure results from another research work are correct by rerunning the experiments and confirm that these results are in line with the described results.

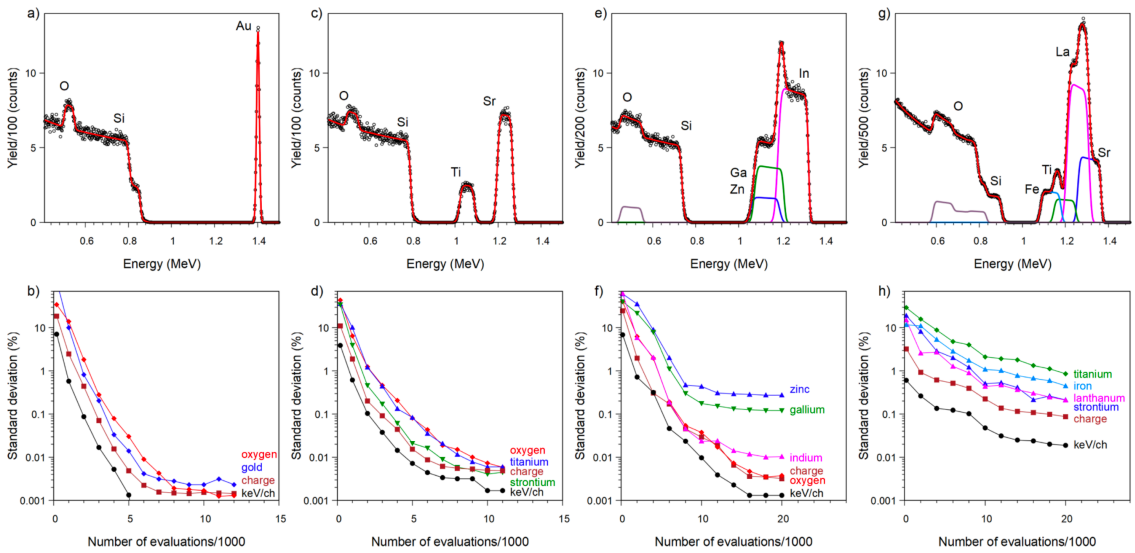


Figure 5.34: (a) Synthetic spectrum obtained for a Au film on a silicon-oxide layer on a silicon substrate, (c) for a SrTiO layer on a silicon substrate, (e) for a InGaZnO layer on a silicon substrate, and (g) for a SrTiO LaFeO SiO multilayer stack on Si. Black data points are the synthetic spectrum obtained with the Monte Carlo method. The red line corresponds to the best fit obtained after fitting the synthetic spectrum with differential evolution using Ruthelde. (b), (d), (f), and (h) Evolution of the convergence of the various parameters during the fitting as a function of the number of evaluations using Ruthelde. Figures made by Heller et al. [5].

In Figure 5.34, the sythetic spectrum and standard deviation graphs for 4 experiments using Ruthelde are shown. The first experiment is the same material as we tested in section 5.3.1. Ruthelde seems to converge better over generations in terms of the standard deviation. A large difference is the amount of function evaluations reached. While all the runs with our approach stop after 5000 evaluations, Ruthelde goes on untill 15000 but with minor improvement. Ruthelde has a better standard deviation on average but if we look at the synthetic spectrum and fitness graph we can see that both approaches are able to converge to a good solution. Experiment 2 in Figure 5.34 is the same material we tested in section 5.3.2. As with experiment 1, our approach converges a lot faster with only 6000 function evaluations. The standard deviation of most parameters of both our approach and Ruthelde goes

below the 0.01 threshold which is good. In terms of the synthetic spectrum plot, both find a good solution regardless of the artificial noise. Finally, experiment 4 in Figure 5.34 shows the results for the same material used in section 5.3.3. Our approach has a better standard deviation for most of the parameters going below the 0.01 threshold. Ruthelde, however, stays above the 0.1 threshold. The synthetic spectra plots are good for both approaches but for our approach, the simulated spectra follows the experiment data very close. Overall, we can observe that our algorithm is more likely to be stuck in local minima than Ruthelde. This can be observed by the drop in standard deviaition and fitness when certain runs terminate. This is probably because in Ruthelde, an individual can be mutated with random values based on a small probability at each generation. This allows the algorithm to retain variation of the population and escape local minima.

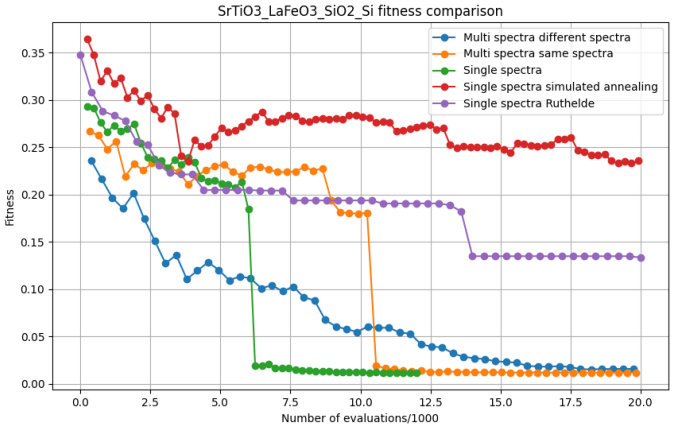


Figure 5.35: Multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate simulated annealing synthetic spectrum

In Figure 5.35, we aggregated all fitness data from this research project on the multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, silicon-oxide on a silicon substrate material. The purple line shows the plot of a single run of Ruthelde on this material using the same parameters as the simulated annealing experiment in section 3.6. In this plot we can see that our approach is able to find a good solution using both single and multi fitting for this material. Simulated annealing performs the worst because his lack of exploitation capability of found solutions at the end of the run. Ruthelde converges quite well. At the end of this research work we identified that the fitness calculation in the used version of Ruthelde is different than the fitness calculation we used throughout the thesis. This explains the difference with the results of our algorithm. An important note is that Ruthelde only has as single run because of time constraints and functionality limitations. Ruthelde does not have a method to run x amount of experiments and aggregate the data easily.

Based on these observations, we can conclude that our approach is able to find good solutions in reasonable time. Compared to Ruthelde, our algorithm is more likely to get stuck in local minima because of the lack of a good implemented technique to escape local minima. The proposed multi spectra fitting algorithm uses both spectra equally to find a good common solution. Finally, we can conclude that differential evolution outperforms simulated annealing.

5.7. Test coverage

For an optimization model it is quite straightforward to see how it performs. For a web application this is harder. Unit and integration tests are an often used method to measure the correctness of a codebase. Therefore, we chose to test all logic functionality of the web application and the optimization models.

Listing 5.1: Coverga report generated with 104 tests

```
Name                                       Stmts   Miss  Cover
------------------------------------------------------------
src/GA/ga_input_output.py                    111      2    98%
src/GA/multi_spectra_optimization.py         182     17    91%
src/GA/server.py                              52     19    63%
src/GA/single_spectrum_optimization.py       106     18    83%
src/__init__.py                                0      0   100%
src/logic/config.py                           14      0   100%
src/logic/convert.py                           9      0   100%
src/logic/file.py                            268      6    98%
src/logic/helper.py                           38      0   100%
src/logic/input.py                            32      0   100%
src/logic/spectra_reader.py                   90     18    80%
------------------------------------------------------------
TOTAL                                        902     74    92%
```

As discussed in section 4.7, pytest and unittest are used for unit testing and the mock libary is used for integration tests. The high dependency on the filesystem and Ruthelde container required a lot of mock objects which made testing hard and complex. In the end, 104 tests are written which tested most of the necessary functionality. The statements that are missed are either not testable or not important. In Listing 5.1, the coverage report is shown in which we achieved a coverage of 92%.

5.8. Summary

This section summarizes the findings and outcomes of the experiments conducted on the optimization models for IBA proposed in Chapter 3.

Our experimental journey began with the examination of different mutation strategies, namely rand1bin, best1bin, and randtobest1bin, as introduced in Chapter 3. Based on the results we found that overall best1bin is the best strategy. The strategy constantly finds superior solutions and converges to a good fitness.

Next, our single spectra optimization approach was tested on various material. Across different material configurations we found optimal solutions which proves the robustness of the chosen DE strategy best1bin. Even in a complex scenario such as with a multilayer stack of strontium-titanium-oxide, lanthanum-iron-oxide, and silicon-oxide on a silicon substrate the algorithm was able to give good results.

The performance of our algorithm on multi spectra optimization experiments was impressive. The algorithm utilized the additional data from both spectra the search for optimal solutions. This approach even matched the performance of the single spectra optimization while finding a good solution for both spectra.

A comparative analysis between DE and simulated annealing showed an interesting observation. While simulated annealing has its strengths in exploration, it fell short in exploitation phases. This shortcoming substantiates the same finding by Jeynes et al. [6] and shows that DE is a better choice for IBA optimization.

Finally, the testing of our codebase, which achieved a good 92% test coverage, ensured the relibility of the optimization models and web application. These tests, together with the experimental results, show the robustness and accuracy of the proposed optimization models.

6

Conclusion

In this thesis, we have explored different techniques to determine the composition of a material using IBA optimization. Based on the work done by Heller et al. [5], we decided to further explore the possibilities of using differential evolution. The proposed algorithms are controlled by a web application. In this thesis we investigated the following research questions:

1. How can Differential Evolution optimization be effectively applied to multi-spectra Ion Beam Analysis to improve material characterization accuracy and efficiency?
2. What are the key software engineering challenges in developing an advanced Ion Beam Analysis platform that incorporates Differential Evolution optimization, and how can these challenges be addressed?
3. In what ways do the newly developed optimization algorithms improve upon Ruthelde in terms of accuracy and efficiency?

Our investigation revealed several insights about them:

- **Mutation strategy**: The comparative analysis of mutation strategies -rand1bin, best1bin, and randtobest1bin- underscored the superiority of the best1bin strategy. Besides being the default strategy of SciPy and favoured by Mezura-Montes et al. [8], best1bin achieves high-quality solutions in reasonable time.
- **Single vs. Multi spectra optimization**: This thesis demonstrated the effectiveness in both single and multi spectra optimization problems. Multi spectra optimization presented an innovative approach to leverage additional data for finding search parameters. Experiment results underscored the potential of multi spectra DE optimization in handling complex material compositions within IBA.
- **Comparison with Simulated Annealing**: A critical evaluation of simulated annealing revealed its limitations. While simulated annealing excels in the exploration phase, the exploitation phase underperforms. This finding aligns with existing literature and establishes DE as a preferable choice for IBA optimization.
- **Comparison with Ruthelde**: Our proposed single spectra optimization algorithm achieved comparable results to Ruthelde. However, for the complex material in Section 5.3.3, our approach outperformed Ruthelde in terms of standard deviation and fitness. This observation should be taken with a grain of salt due to the limited data validation of Ruthelde. An interesting insight is that our approach is more likely to be stuck in local minima due to the lack of a technique to easily escape local minima such as in Ruthelde.
- **Web application**: This thesis presented a web application to control the proposed DE algorithms. By manipulating Docker containers, we offered a robust, user-friendly and scalable software package.

In conclusion, this thesis contributes significantly to the field of IBA by providing a comprehensive

evaluation of DE and simulated annealing. We show the potential of multi spectra IBA optimization. The findings do not only enhance our understanding of optimization algorithms for IBA but also show future research directions. These open avenues for future research could elevate the precision and efficiency the evolving semiconductor industry needs.

6.1. Future work

While this study has provided interesting insights, the field of IBA optimization methodologies requires further exploration. we conclude with the identification of a series of interesting research directions, namely:

1. Testing the multi spectra optimization algorithm on more experiments. It would be especially interesting to test a material where the single spectra optimization algorithm underperforms on and then see how the multi spectra optimization algorithm performs.
2. Explore techniques to reduce the probability that our DE algorithm gets stuck in a local minima. In Ruthelde, an individual can be mutated with random values based on a small probability at each generation. This technique allows the algorithm to retain variation in the population. In our current DE algorithms, no such techniques are implemented.
3. Explore different optimization techniques offered by SciPy. SciPy offers a wide variety of optimization algorithms[1] and an easy way to substitute. Interesting algorithms to explore are Basin-Hopping, Conjugate-Gradient, or Broyden-Fletcher-Goldfarb-Shanno.

[1] https://docs.scipy.org/doc/scipy/reference/optimize.html

References

[1] N Pessoa Barradas and A Vieira. “Artificial neural network algorithm for analysis of Rutherford backscattering data”. In: *Physical Review E* 62.4 (2000), p. 5818.

[2] Devansh. *Why you should be using Differential Evolution for your optimization problems.* Sept. 2020. URL: https://medium.datadriveninvestor.com/why-you-should-be-using-differential-evolution-for-your-optimization-problems-b3b2ed622c4a.

[3] Gartner. *Forecast: Public Cloud Services, Worldwide, 2021-2027, 2Q23 Update*. https://www.gartner.com/en/documents/4509999. Accessed: 2024-02-24. 2023.

[4] John L. Gustafson. “Moore’s Law”. In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 1177–1184. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_81. URL: https://doi.org/10.1007/978-0-387-09766-4_81.

[5] René Heller et al. “Differential evolution optimization of Rutherford backscattering spectra”. In: *Journal of Applied Physics* 132.16 (Oct. 2022), p. 165302. ISSN: 0021-8979. DOI: 10.1063/5.0096497. eprint: https://pubs.aip.org/aip/jap/article-pdf/doi/10.1063/5.0096497/16518372/165302_1_online.pdf. URL: https://doi.org/10.1063/5.0096497.

[6] C Jeynes et al. “Elemental thin film depth profiles by ion beam analysis using simulated annealing-a new tool”. In: *Journal of physics D: applied physics* 36.7 (2003), R97.

[7] Simon Marien. *Multi-spectra-IBA-optimization*. https://github.com/simonmarien/Multi-spectra-IBA-optimization. 2024.

[8] Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos Coello. “A comparative study of differential evolution variants for global optimization”. In: vol. 1. July 2006, pp. 485–492. DOI: 10.1145/1143997.1144086.

[9] Ji Qiang and Chad Mitchell. “A Unified Differential Evolution Algorithm for Global Optimization”. In: *IEEE Transactions on Evolutionary Computation.* (June 2014). URL: https://www.osti.gov/biblio/1163659.

[10] Heller. René. *Ruthelde*. https://github.com/DrReneHeller/Ruthelde. 2023.

[11] Ernest Rutherford. “LXXIX. The scattering of α and β particles by matter and the structure of the atom”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 21.125 (1911), pp. 669–688.

[12] Abraham Savitzky and Marcel JE Golay. “Smoothing and differentiation of data by simplified least squares procedures.” In: *Analytical chemistry* 36.8 (1964), pp. 1627–1639.

[13] Rainer Storn and Kenneth Price. “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of global optimization* 11 (1997), pp. 341–359.

[14] vbuel. *python-javaobj*. https://github.com/vbuell/python-javaobj. 2015.

[15] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[16] Florian Wittkämper et al. “Energy-Dependent RBS Channelling Analysis of Epitaxial ZnO Layers Grown on ZnO by RF-Magnetron Sputtering”. In: *Crystals* 9.6 (2019). ISSN: 2073-4352. DOI: 10.3390/cryst9060290. URL: https://www.mdpi.com/2073-4352/9/6/290.

[17] Y. Xiang and X. G. Gong. “Efficiency of generalized simulated annealing”. In: *Phys. Rev. E* 62 (3 Sept. 2000), pp. 4473–4476. DOI: 10.1103/PhysRevE.62.4473. URL: https://link.aps.org/doi/10.1103/PhysRevE.62.4473.

[18] Yang Xiang et al. “Generalized simulated annealing for global optimization: the GenSA package.” In: *R J.* 5.1 (2013), p. 13.

A

Experiment parameters

	# runs	N	F	CR	Max fun eval	# params	Time/run	Time SD
Rand1bin	20	20	0.6	0.85	/	7	25 min	8.2 min
Best1bin	20	20	0.6	0.85	/	7	19 min	3.3 min
Randtobest1bin	20	20	0.6	0.85	/	7	21 min	4.2 min
Au SiO2 Si	20	20	0.6	0.85	12000	7	6 min	1.6 min
SrTiO3	20	20	0.6	0.85	12000	7	4 min	0.5 min
SrTiO3 LaFe03 Si02 Si	20	20	0.5	0.7	120000	12	35 min	5.4 min
Multi spectra fitting	20	20	0.5	0.7	20000	16	72 min	12.5 min
Simulated annealing	20	/	/	/	20000	12	58 min	4.3 min

Table A.1: Experiment parameters for optimization model experiments

B

Ruthelde screenshots

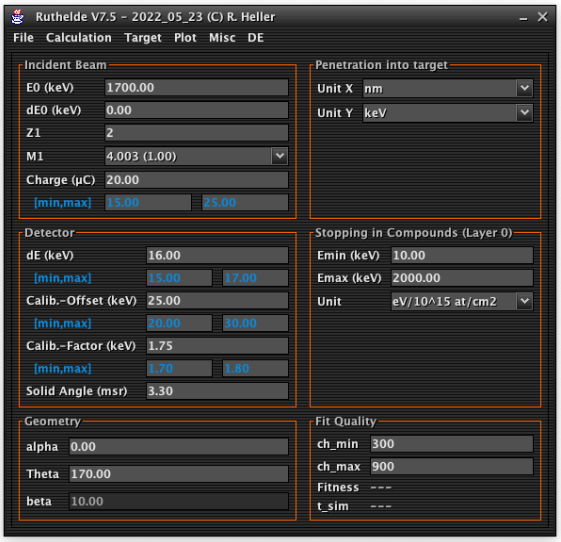


Figure B.1: Screenshot of Ruthelde main screen.

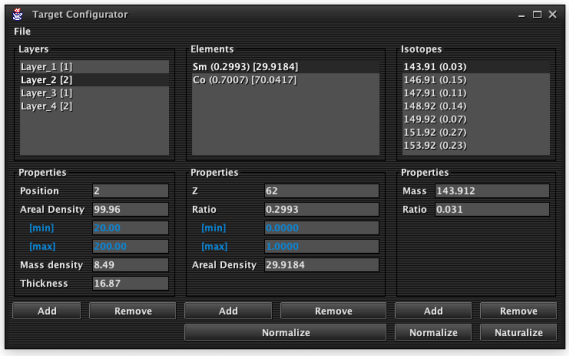


Figure B.2: Screenshot of Ruthelde target screen.

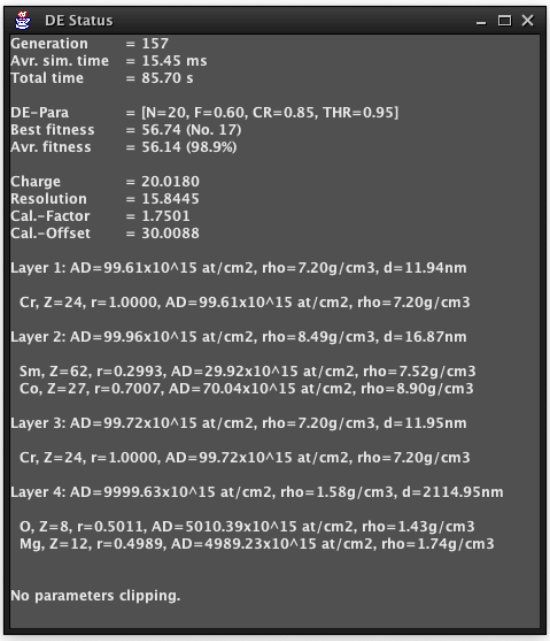


Figure B.3: Screenshot of Ruthelde DE status screen.

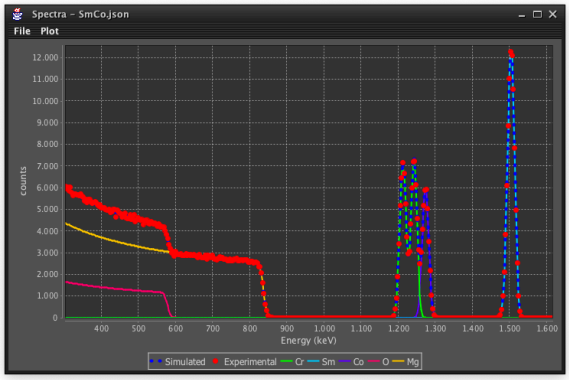


Figure B.4: Screenshot of Ruthelde spectrum screen.

C

Web application screenshots

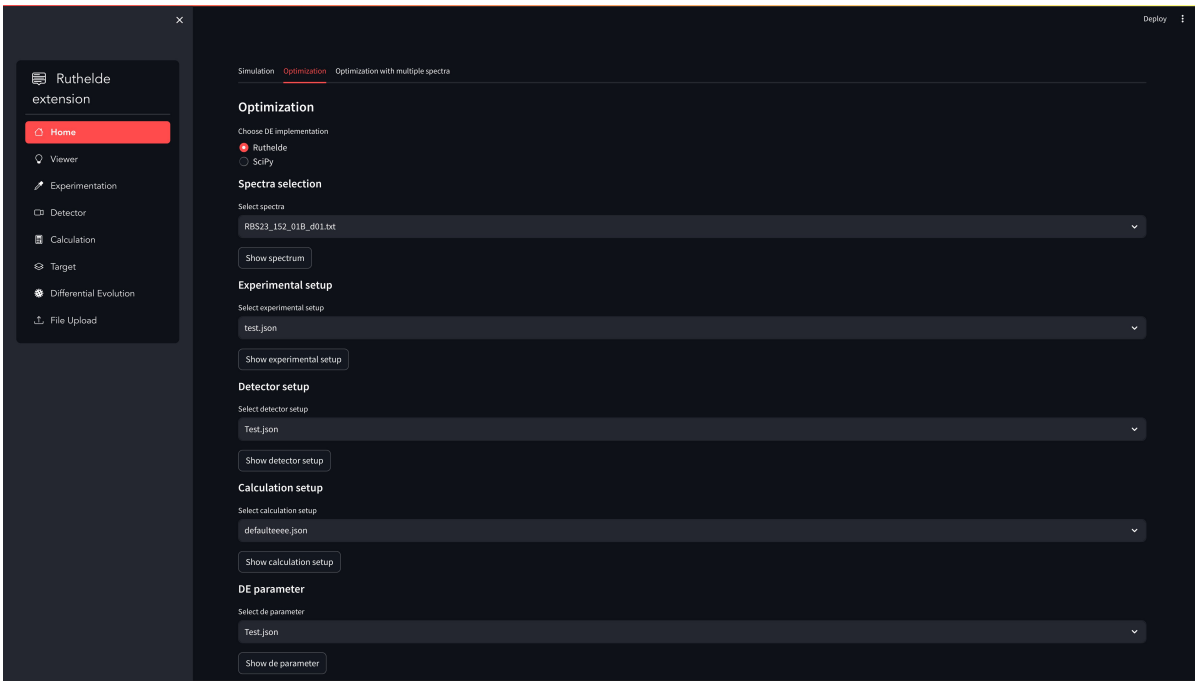


Figure C.1: Screenshot of web application optimization main screen.

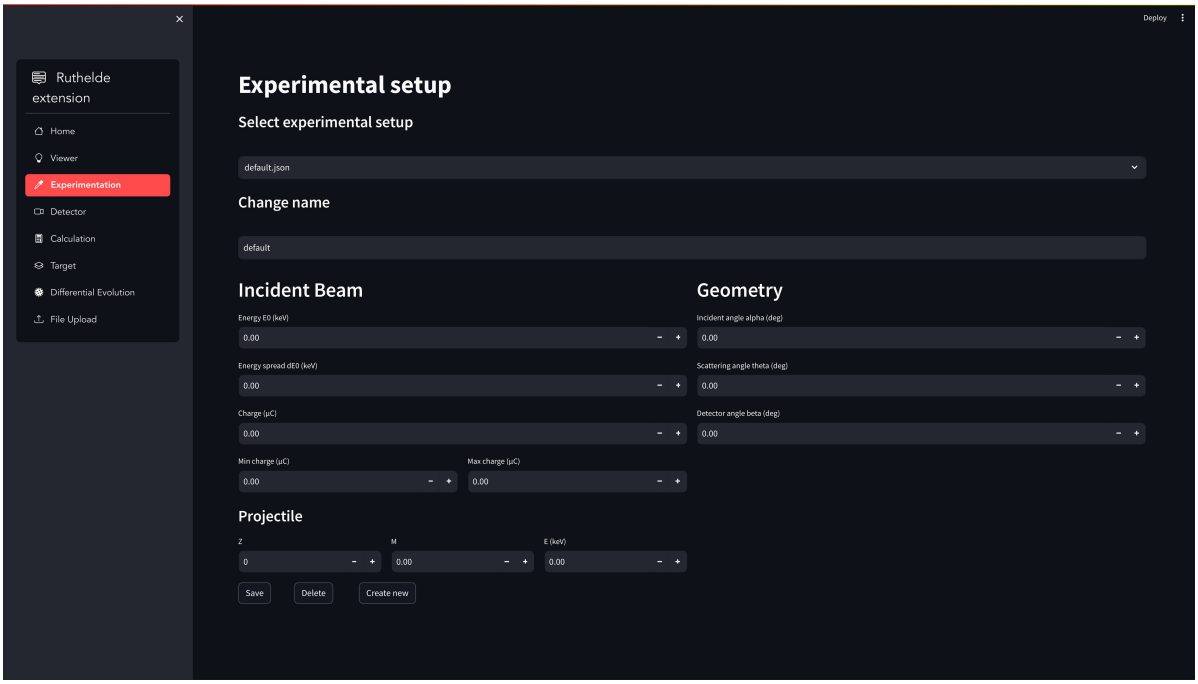


Figure C.2: Screenshot of web application experimental setup screen.

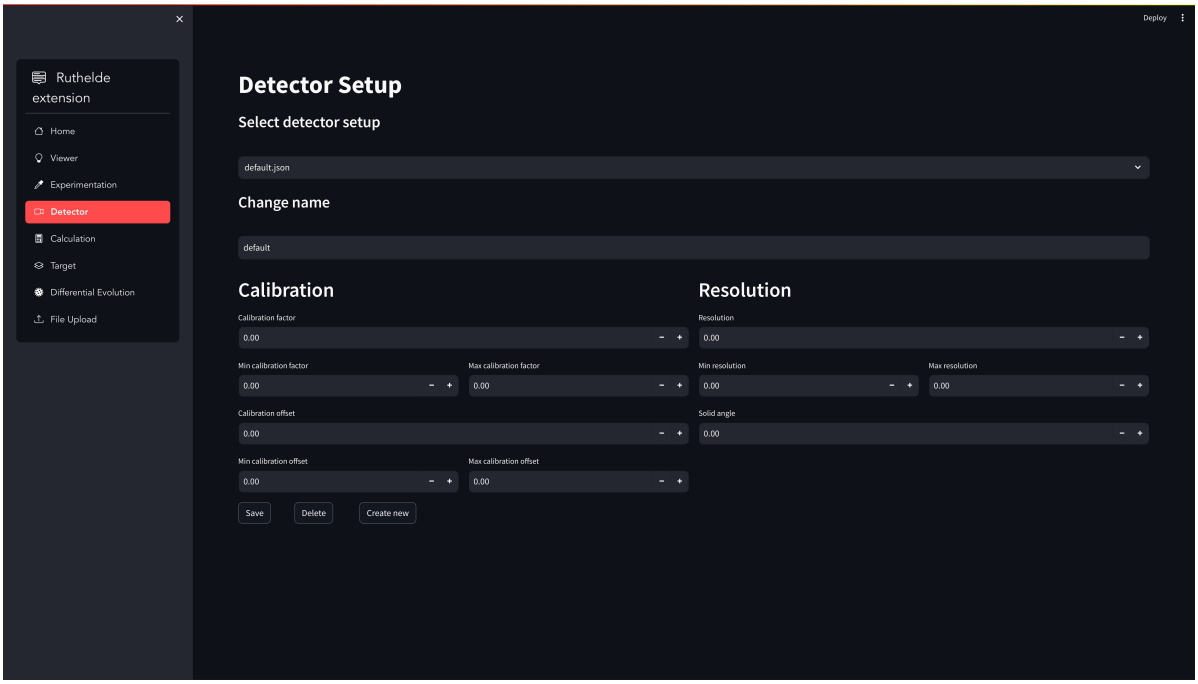


Figure C.3: Screenshot of web application detector setup screen.

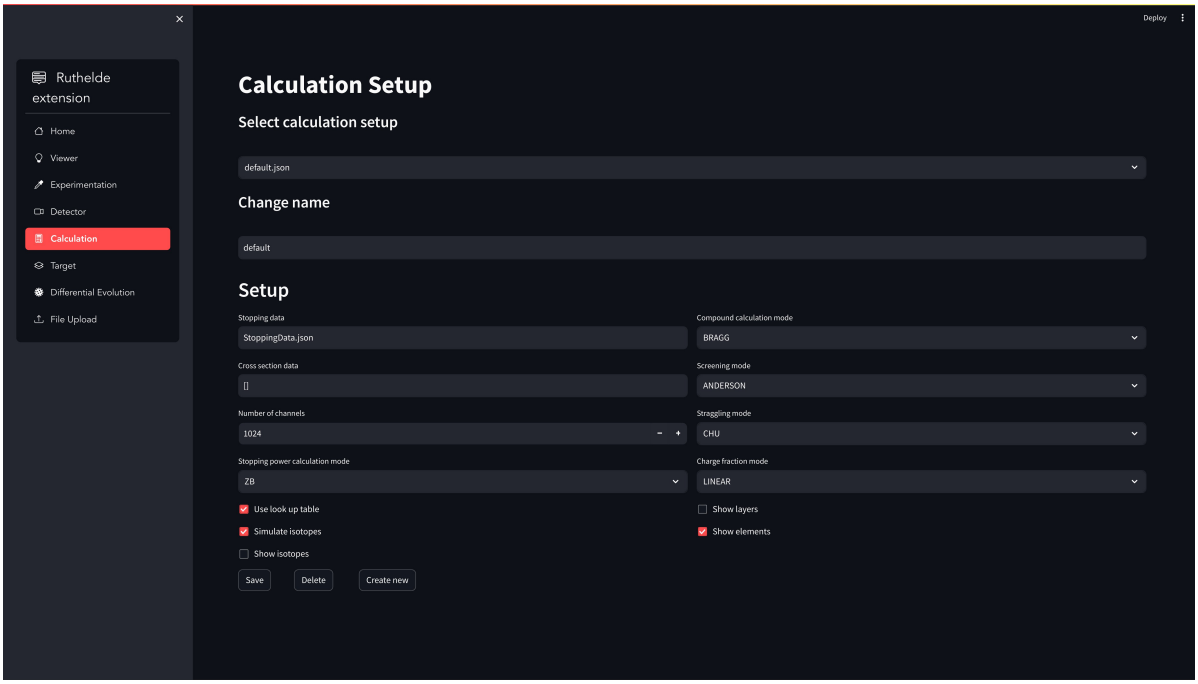


Figure C.4: Screenshot of web application calculation setup screen.

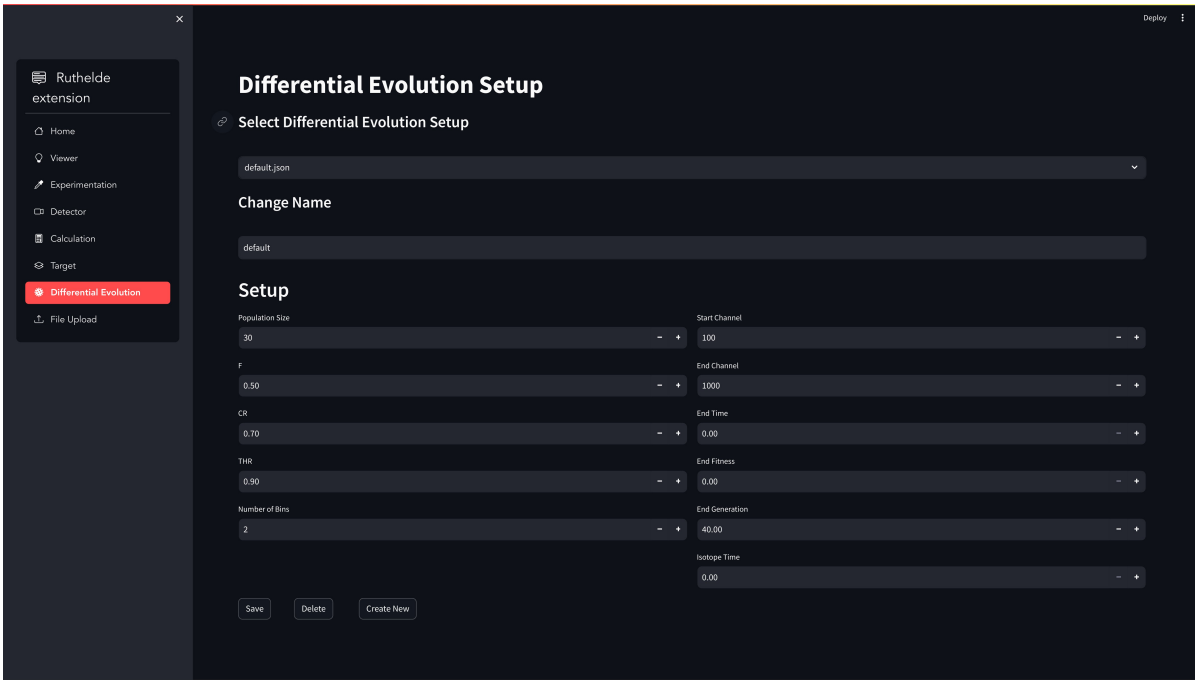


Figure C.5: Screenshot of web application differential evolution setup screen.

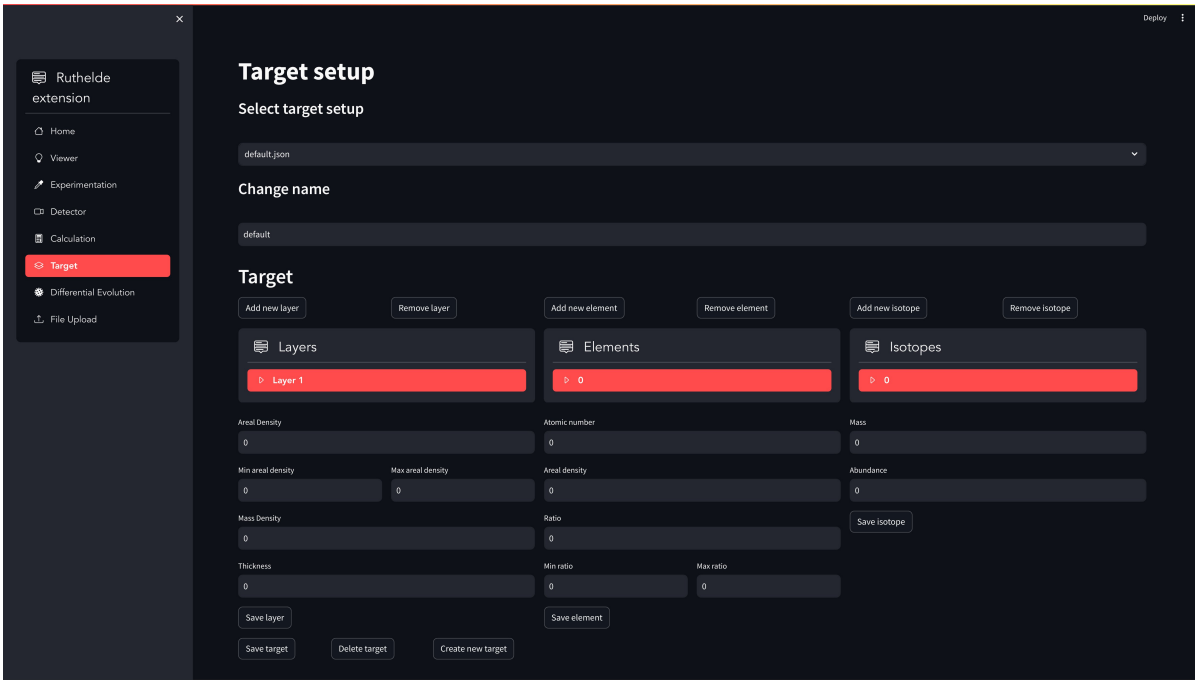


Figure C.6: Screenshot of web application target screen.

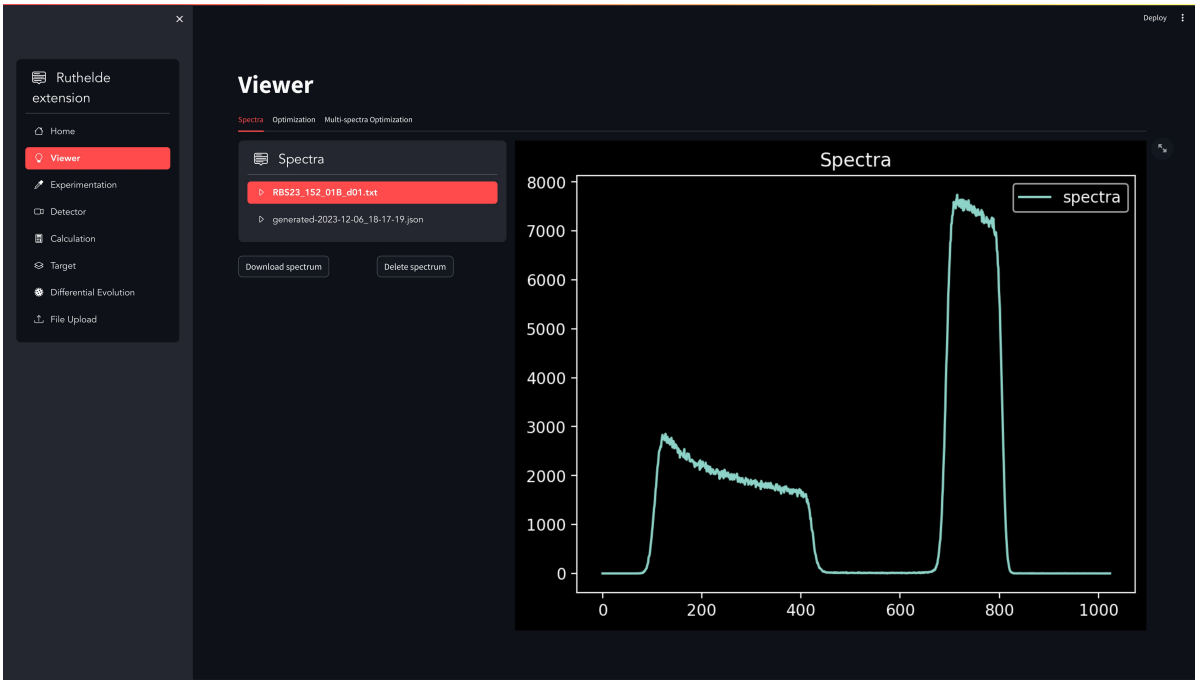


Figure C.7: Screenshot of web application spectra viewer screen.