Effective Design Space Exploration

Exploration of the UV curing process for uniform UV distribution in commercial printers

de Zeeuw, Storm

Award date:
2021

CANON PRODUCTION PRINTING

Electrical Engineering Department
Electronic Systems Group

Effective Design Space Exploration:

Exploration of the UV curing process for uniform UV distribution in commercial printers

Master Thesis

Storm de Zeeuw

TU/e Supervisor:
dr. ir. M.C.W. Geilen

Canon Supervisor:
dr. J.A. Luiken
dr. ir. B.D. Theelen

Assessment committee:
dr. ir. M.C.W. Geilen
dr. J.A. Luiken
dr. ir. P.J.L. Cuijpers
dr. ir. B.D. Theelen

Eindhoven, October 2021

Contents

Abstract

Automated Design Space Exploration (DSE) methods are, due to the ever growing increase in available computational power, a now viable way to perform design space exploration on complex engineering problems. Canon Production Printing (CPP) is looking to leverage the power of automated DSE to improve their development process and find new interesting designs faster.

For this thesis, we applied automated DSE methods to find optimal designs for curing ink in Canon printers. To do so, a model and evaluation method were created, a tool for performing automated DSE was selected, a design space was created using input generators and DSE was performed. To evaluate and improve the automated methods, various measurements for the process of automated DSE were created and applied, which gave insight into the workings of the methods and their achieved results. The measurements were used to evaluate the DSE methods which led to a significant improvement of the results for one of the methods and showed the flaws in another, these flaws were previously difficult to discover.

Chapter 1

Introduction

UV curing, the process of drying ink using ultra-violet light, is a complex physical chemical process. Due to its many interacting elements, which will be described in subsection 3.2.1, it is hard to predict the outcome of the UV curing process. Canon Production Printing (CPP) employs a model to simulate the process of UV curing, which allows for easy testing of designs. Using this model as a basis, it is possible to develop an evaluation method and perform automated Design Space Exploration (DSE) to optimize the curing process. The goal is to find the optimal lamps and their trajectories to cure the ink such that the occurrence of wrinkles on the final finish is minimized. An important facet in design space exploration is the reliability of the results: we must be sure that the design space is sufficiently explored.

1.1 Canon Production Printing

Canon Production Printing, formerly Océ, was founded in 1877 in Venlo, the Netherlands. The company has a long history of technical innovation and development. A key asset is inkjet, a widely applicable imaging technology. Their ambition is to build on their expertise in jetting for high-volume, high-speed printing and to position themselves as a leader in inkjet technology and applications.

1.2 Problem description and Research question

CPP is looking to improve the effectiveness of their design space exploration for the creation of printers by leveraging automated DSE methods. CPP is interested in utilizing these methods to solve complex engineering problems, as the methods can find good and interesting designs without requiring the intensive and time consuming process of manual design space exploration. However, these automated methods are often grey-box, i.e. it is known which process they follow, but due to many stochastic processes their outcome cannot be determined from input. These methods therefore are not able to explain the results. This means there is no guarantee for good results. Especially methods employing parallelization, which have become more viable due to the increase in available computational power, are hard to analyze properly. Due to their lack of concurrency it is hard to follow their exploration through the design space.

The core question then becomes: “How can we effectively measure the quality of a parallelized automatic design space exploration method to improve confidence in its results?”

1.3 Subproblems

For the thesis, and to answer the research question, multiple smaller problems need to be addressed.

1.3.1 Creating a setup for design space exploration

Performing automated DSE on any single problem involves an intricate setup phase, which makes it difficult to perform DSE on a new problem without extensive knowledge of specific tools or the algorithms and methods themselves. To address this, we create a problem-agnostic setup for automated DSE in which various problems can easily be inserted and explored using different methods without requiring domain-specific knowledge.

1.3.2 Analyzing and measuring the performance of automated DSE methods

A core problem in DSE is that there is no a priori knowledge on which combination of design parameters constitute the optimal design. This makes it hard to evaluate a DSE method based only on its final results. In this thesis, we explore various methods and quality measurements for objectively evaluating DSE methods.

1.3.3 Creating the design space for the UV curing process

The UV curing process is the main contributor to ink wrinkle in the final print. To minimize this, CPP is interested in exploring all design possibilities to minimize or even eliminate the wrinkling from their prints. To be able to do so, there needs to be a model which takes a design and simulates it, and an evaluation method to evaluate the results of the simulation to allow for an objective comparison between designs and their performance. CPP has provided a model, but it needs to be expanded and adjusted to fit the DSE process.

1.3.4 Exploring the design space of the UV curing process

An effective automated DSE method needs to be found or developed and adjusted to explore the design space. This will be achieved by applying various DSE methods to the problem and analyzing the results and process using quality measurements.

1.4 Other papers related to UV curing

The concept of wrinkling in UV cured ink itself has been discussed in various papers on polymer science. There is a growing interest in how to induce wrinkle, rather than minimizing the effect[8], for material design purposes. In the paper by Juan Rodriguez-Hernandez [23] the effect of wrinkling in materials such as UV cured polymer materials are discussed, such as the process of buckling. The cause for these effects is also discussed: there is much disagreement on what causes wrinkling. The conclusion is that wrinkling occurs due to a combination of factors, including UV curing, but also environmental gases, contamination products, and most of all a rigid skin layer on top of the material. This skin layer, in our case, would form due to the UV light only curing the top layer of the ink, resulting in a top layer that is cured while the rest of the material is still “fluid”. This skin then shrinks

which would cause the wrinkle. This skin can also be a gradient in curing grade, as can be seen in Figure 1.1.

There is one paper by Jonas Simon and Andreas Langenscheid which goes into depth on how UV light affects the curing process [26], and describes the optimal settings for UV intensity and dosage. They describe ways to change UV dosage, and how certain wavelengths affect curing. Unfortunately it focuses mostly on changing the ink structure itself and how different wavelengths effect the curing process. These properties could be interesting for further development of the model used in the DSE process, but are not used for this thesis.

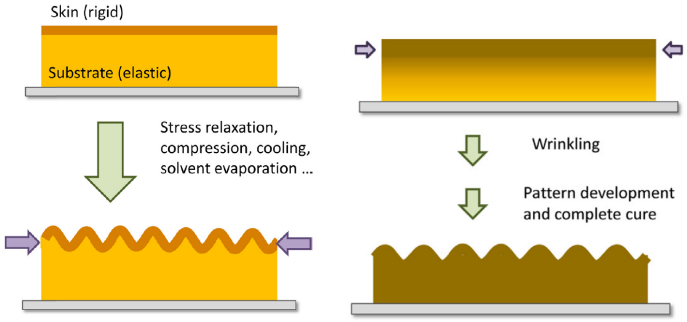


Figure 1.1: Schematic representation of wrinkle formation in a system formed by a rigid skin layer or a gradient in cure on top of an elastic substrate. [23]

1.5 Thesis structure

- In chapter 2, the required background information is supplied and the existing methods explained
- In chapter 3, the setup for the model and performing DSE are explained
- In chapter 4, the quality measurements are explained and discussed
- In chapter 5, the DSE methods are analyzed and compared based on the quality measurements
- In chapter 6, the best designs found by the DSE process are discussed
- chapter 7 contains the conclusion

1.6 Glossary

This section contains a glossary of domain-specific terms and their descriptions.

UV, Ultra Violet Part of the electronic wave spectrum, specifically wavelengths between 10 and 400 nm, which is just at the edge of the most energy-dense side of visible light, often used in curing processes.

UV ink Ink which needs to be cured using UV light.

UV dosage UV dosage is how much UV light has been received over time, Dosage = Intensity multiplied by time ($(J/cm^2) = (W/cm^2) \cdot s$).

Wrinkling Buckling of the UV ink which may happen during the UV curing stage.

Printing surface Also known as the canvas: the complete surface a printer can cover.

Jet/Jetting Depositing ink on the printing surface.

Area of Interest(AoI) The area on the printing surface where the ink is deposited and where minimizing wrinkle is of interest.

Samples/sampling The evaluation of a solution or design.

Fitness the quality measurement of a design

Design space The space containing all possible designs, mapped to the resulting fitness of that design.

Printing direction The direction the paper travels through the printer.

Swath direction Direction at perpendicular angle to the printing direction.

Optimum Design or solution with the highest fitness.

Local optimum An optimum which is only optimal locally, but not necessarily globally.

Global optimum An optimum for which no better optimum exists.

Local/in the neighborhood of Designs that are similar to each other are considered local to each other.

Exploration Broadly searching for new designs.

Exploitation Improving an already found design by exploring only locally.

Design A selection of parameters that form the basis of a design, in our case a movement pattern and lamps.

Gene Term used in evolutionary algorithms, an element of a design, equal to a parameter or collection of parameters of a design.

Chapter 2

Existing methods and measurements

In this chapter, we provide the required background information on the employed DSE methods. We first explain the basics of DSE, provide some more insight into the relevant elements of the automated DSE process, we then give some background on existing measurements for DSE methods and end with the details of the used DSE methods themselves.

2.1 Design Space Exploration

In its most simple form, Design Space Exploration (DSE) is the process of creating designs, evaluating them analytically or by simulating the design and adjusting the design based on the evaluation, repeating these steps until one has a satisfying design. Doing this process for large design spaces, however, is laborious and time-consuming: performing DSE manually is in many cases an unrealistic or very expensive prospect. Since computers with immense computation power are now more accessible, given a method to effectively simulate, evaluate and adjust parameters according to some evaluation, DSE can be performed on problems previously prohibitively large. Luckily, such methods already exist and are largely problem agnostic (see Section 2.4, Section 2.5). These are metaheuristic methods, which look for a solution to a problem without understanding the underlying problem itself. They do so by employing strategies that drive the algorithm to discover the design space and change the input parameters towards solutions they expect to return good results based on earlier found solutions.

2.1.1 The design space

The design space is the space spanned by the values of all design parameters, i.e., each parameter is reflected by a dimension of the design space. This design space, therefore, contains all possible designs and can be expressed as an n-dimensional space, where n is the number of input parameters. The different dimensions of the design space can be continuous, i.e. where there is always a point between any two distinct points, or discrete, where the parameters have a step size between each possible parameter they can represent. These two types of dimensions can both be, and often are, in a single design space at the same time. In this thesis, we focus on design spaces of considerable size, where it would be unreasonable to check all possibilities, or at least all possibilities given the allowed accuracy limit for the

continuous parameters. As for small design spaces, it will often be less time-consuming to just test all possibilities manually.

In absolute terms, a design space is unique to a problem, there is one design space containing all possible designs. In reality, there is a need to approximate and represent the design space using abstractions, to cull most infeasible designs and to make exploration using automated methods easier. One can define how to represent the possible designs to create this design space. This design space should cover all valid options and allow for effective explorations of those options, given the method of exploration. What this means is that design spaces themselves also need to be designed: a design problem needs to be created that takes input values for the design and returns its evaluation, as long as all designs of interest are included and similar inputs give similar results, performing automated DSE is viable.

2.1.2 Exploration and exploitation

DSE has as final goal to find feasible designs. There are two main, nearly contrary, aspects to balance when performing DSE: exploration and exploitation. Given a problem, all possible designs should be considered, however, this is often infeasible. Still, it stands to reason that the more varying designs you test, the more you can be confident that a found optimal design is not superseded by other, not yet discovered, designs. The more the design space is explored, the more confidence you can have in your final result. Exploration, however, takes time, which is exactly what we want to shorten. Instead, it would be better to start looking for better designs based on already found designs, for example by sampling further away from where bad designs were previously found, or closer to already discovered good designs. In this case, the design space is exploited: the focus is put on improving already discovered designs by testing similar designs and evaluating their results. A good DSE method uses both exploration and exploitation to find the best designs. Enough parameter combinations should be considered, but there should also be convergence to an optimal solution within a reasonable amount of time, where reasonable is dependent on the problem and the time the user is willing to spend.

2.1.3 Local and global optima

During DSE, one might encounter an optimum: a design for which, according to testing, no better solution exists. It, however, can not be claimed that the optimum is a global optimum, unless all other possibilities have been explored and found to be worse or as good as the found optimal solution. A visual example illustrating the difference between a local and global optimum can be seen in Figure 2.1. Without testing all possible designs, it is impossible to claim a certain point to be the global optimum since an untested design can not be claimed to be inferior.

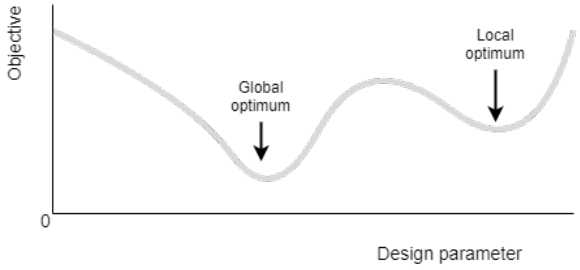


Figure 2.1: Local and global optima

2.1.4 Example: one dimensional DSE

To give a concrete idea of what a design space exploration looks like in a formal setting, we present an exemplary single-dimensional design space optimization problem, Equation 2.1. There is some function f with a single input x, which gives an output, y.

$$y = f(x) \tag{2.1}$$

The goal is to find a value for x which gives us the smallest value for y. Although most might think to first analytically find the optimal for the function, in the case of most engineering problems, the actual function is unknown, not a simple n-dimensional formula, but unpredictable, or at least very hard to analyze with common methods. This forces us to sample the function instead, by testing various values for x. Sampling itself however takes time, the function might take hours to simply return a result y given an input x. Given that we do not know the results of the function, at first, the design space is empty, as no samples have been taken and therefore nothing is known about the design space. For illustrative purposes, the actual function is made visible in grey in Figure 2.2.

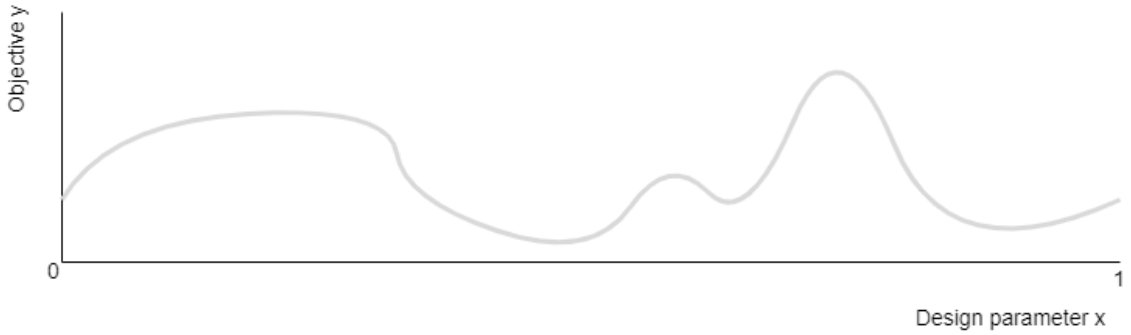


Figure 2.2: In grey: The design parameter x, mapped to objective y as $f(x)$

It is now up to the method, automatic or manual, to explore the design space by "sampling" the function $f(x)$. This results in the goal as shown in Equation 2.2. Where D is the range over which we explore the parameter, or parameters, x and where arg max returns the argument which returns the minimal value for the function.

$$\arg\min_{x \in D} f(x) \tag{2.2}$$

Depending on the problem, it may be required to maximize as opposed to minimize.

2.1.5 Multi-objective optimization

Many problems require multi-objective optimization. In a lot of cases these objectives are opposing, for example, energy usage and speed in processor design. The multi-objective optimization causes some issues for standard methods of DSE: multi-objective designs are not directly comparable, making it hard to elect one design as better than another.

Pareto

To make multi-objective designs more comparable, the concept of Pareto optimality was developed [3]. For Pareto, solutions are defined as being either Pareto optimal or Pareto dominated. A solution is Pareto optimal if no solution exists which has better results for all objectives. A solution is Pareto dominated if a different solution exists which has better results for all objectives.
By considering all Pareto optimal solutions as the best solutions, we have a simplistic initial ordering between Pareto optimal and Pareto dominated solutions. However, this does leave the fact that there can theoretically be infinite Pareto optimal solutions which all are incomparable to each other. To deal with this, many methods of comparing the Pareto optimal solutions have been developed [31], but none are perfect.

2.2 Automated DSE

Automated DSE is the process of performing DSE by utilizing computative power. Instead of performing DSE manually, the space is explored using a pre-described, automated method, called a metaheuristic.

2.2.1 Grey box

The metaheuristics do not employ intuition or domain knowledge, relying completely on its own sampling of the design space to explore it further. Although the steps a method takes are known, and often quite intuitive to understand, the actual exploration by the metaheuristic is not predetermined. All metaheuristic methods are reactive to the results of their own sampling of the design space and it is not possible to predict the path through the design space it will choose beforehand.

2.2.2 Regarding parallelism

In automated DSE there are two types of exploration, sequential search methods, which are memory efficient and meant to run sequentially and parallel search methods which utilize parallelization. Sequential methods consider only one point at a time but can store multiple solutions in memory to compare against. These sequential methods are mostly employed for single objective search. In recent times, however, multi-core hardware has become commonplace and multi-threaded applications have become more viable and in many cases faster than their single-threaded counterparts. With four cores, one can in theory sample four different places at the same time, resulting in a speedup of factor four. Linear methods, however, can not make use of it due to their sequential nature, each new sample is evaluated based on all previous samples and only one sample can be considered at one time so sampling four at the same time is not possible. This has resulted in the development of new parallel methods which do not only benefit from speedup from parallelization but can also

use new approaches which were previously either impossible or very time intensive. Many of such new methods are, for example, able to perform multi-objective optimization. Examples of parallel metaheuristics are Evolutionary algorithms, and multi-agent optimization algorithms such as Particle Swarm Optimization, these are described in-depth in Section 2.4 and Section 2.5 respectively.

2.2.3 Other advantages of automated design space exploration

There are more advantages than just speed to automated DSE. The automated DSE, in concept, is unbiased: The methods used have no inherent knowledge or notions of the domain of the problem, there are no assumptions made and no bias preferring certain solution types over others. This allows objectivity in the exploration which can find new solutions previously untested due to false assumptions and bias by those performing manual DSE. This does not mean that there is never bias, as the bias of designers of the model and evaluation method can still influence the outcome of the exploration. However, the exploration itself can be unbiased.

2.2.4 Designs and solutions

In this thesis, there are many mentions of the concept of "solutions" and "designs". Designs reference printer designs, however, in the field of optimization, the concept of solutions is used. A solution is an input for a function and the most generic way to describe this concept. In this thesis, you can interpret a design as a solution, where the problem, or function, is the UV curing process.

2.3 Quality measurements for DSE methods

There do not exist many standardized ways to evaluate design space exploration methods. In this section, the common method and some novel methods from literature for quality measurements of automated DSE are discussed.

2.3.1 Benchmarking

The main method of comparison is benchmarking, where multiple metaheuristic methods are run on the same test suite and the results are compared to each other. This comparison happens based on the quality of the found optimal solution.

These methods give a good idea of the exploitative measures of a metaheuristic, but do not reflect why a metaheuristic might perform well on a given problem. This method of evaluating methods is also heavily dependent on the problems in the test suite, making results depend as much on the test suite as on the metaheuristic itself. This makes the results mostly useful to see if a metaheuristic performs as intended, but not what qualities the metaheuristic has that might make it better or worse than other metaheuristics on problems outside of the benchmark.

To give more insight into relevant characteristics, work has been put towards creating test suites that cover a wide variety of problem types [28], such that one can compare methods by seeing how well they perform on various tests with different characteristics. This form of testing is, however, still very much black-box, only looking at the input and the result, but not at the process of the exploration.

2.3.2 Uniqueness

Uniqueness is a quality measurement based on novelty, a concept for measuring DSE methods put forward by Reehuis et al. in [22]. It focuses on the novelty, or newness of solutions that are explored. Given an archive of previously considered solutions: how different is the newest sample which is explored? This difference is defined by a measurement, for which Reehuis gives some relevant examples in his paper based on (Euclidean) distance, such as the sparseness measure by Lehman and Stanley [16]. For this measure, Equation 2.3, a candidate solution x is compared to a set of solutions Q. Q, Equation 2.4, is the combination of all earlier sampled solutions S and the set of solutions X which is currently being considered for testing and evaluation, excluding the candidate solution x to determine the k closest neighbors given distance measurement d which returns the euclidean distance between two solutions or points.

$$\arg\min_{\{q_1,\ldots,q_k\}\subseteq Q} \sum_{i=1}^{k} d(x, q_i) \tag{2.3}$$

$$Q = \{q|(q, f(q) \in S\} \cup X \backslash \{x\} \tag{2.4}$$

The sparseness of x then is the average distance to the closest k solutions, as described in Equation 2.5.

$$Sp(x) = \frac{1}{k} \sum_{i=1}^{k} d(x, q_i) \tag{2.5}$$

This measure was simplified to the 'Uniqueness of solution' measurement. Where sparseness measures an average distance to all k neighboring solutions, Uniqueness only measures the distance to the closest neighbor. Uniqueness also does not take into account X, the collection of points under consideration, in the solution set to measure distance against. The measurement, as can be seen in Equation 2.6, uses an archive S containing all sampled parameters, which was inspired by a similar method from [10].

$$Un(x) = \min_{q,(q,f(q))\in S} d(x, q) \tag{2.6}$$

The difference between the two measurements comes down to the archive of points compared to: Sparseness shows how new the point is compared to a limited set of previous points. This means that the measure is relative and not absolute, a new measurement with a high novelty is only new compared to the limited set. This gives more insight into behaviour in the moment. Uniqueness compares to all points and is therefore an absolute measurement, it is measured against everything sampled so far. Uniqueness gives more insight into the discovery of the complete design space while novelty can give more insight into behaviour of a method. Novelty, however, requires that you choose a set size beforehand, and different set sizes give different results, which makes it difficult to generate objective results. Uniqueness does not have this problem.

2.3.3 Coverage measurements

Another measure of interest is the coverage of the design space, which should give insight into the exploratory capabilities of the DSE methods under consideration. For this thesis,

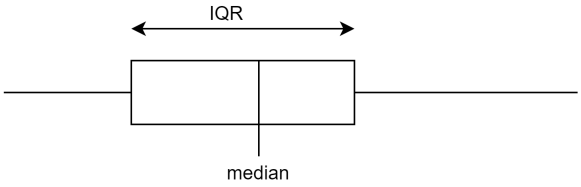


Figure 2.3: The IQR as shown in a boxplot

we will define the coverage as the dispersion of points, the higher the dispersion, the better the coverage. This measurement reflects the exploratory capabilities of the metaheuristic method; if there is not a lot of dispersion, then the samples are not covering the space well and there would be less confidence in our result: there exist a lot of unexplored regions where a better solution could have been. The state of the art for measuring of dispersion of points in multidimensional spaces is discussed in depth in [15]. There, unfortunately, does not exist a commonly accepted method for the measurement of the dispersion of multidimensional data. Instead, the individual parameters will be checked for their one-dimensional dispersion, and a pairwise correlation will be performed between all pairs to give at least an approximation of the coverage of the design space.
For the correlation measurement, Pearson[6] will be used, a method for measuring the correlation between points. A more detailed explanation is in Section 4.1.3.

For the measure of dispersion of points in the one dimensional space, so per input parameter, the inter-quartile range (IQR) can be used. The IQR represents the range of the 25th % to 75th % of all datapoints, sorted by value, as shown in Figure 2.3. This makes IQR more robust compared to measurements such as the standard deviation, as outliers do not have a large impact. The value to aim for is half the total size of the possible parameter range, as this would indicate 50% of the values are distributed over half the total range. A too small IQR would indicate most points are concentrated in one spot, while an IQR above the halfway mark would indicate that the points are more concentrated on two or more spots far removed from each other.

2.4 Evolutionary algorithms

The basic concept of evolutionary algorithms is inspired by the evolutionary process of nature. Imitating the "survival of the fittest", evolutionary algorithms work by having a population containing members which all have their own genetic makeup: each member is made up of genes. For DSE, these members are designs, while the genes are parameters that make up the design. The fittest members are selected for breeding and create offspring with similar genetic makeup. The basic concept is that the best genetic traits get passed on to every new population while the bad traits are filtered from the gene pool. As it is not known which genes are good or bad, an evaluation is needed which evaluates the performance of each individual in the population. Those who perform better are assumed to have better genes.

2.4.1 Population

In evolutionary algorithms, the population size stays the same throughout each generation. The members of the population all have the same gene structure, but with different values. The evolutionary algorithm looks for optimal values for the genes. The size of the population determines how much variety can be tested at any one time, as there is never a direct comparison between different generations. This means that having a small population size can and will result in less exploration as only a small variety of genes can be tested every generation. However having too large a population is also problematic: if the population becomes too large, then each generation takes a longer time to evaluate, thus increasing the total time needed to carry out the optimization.

2.4.2 Breeding

After a generation has had its members evaluated, a new generation needs to be formed. This creation of the next generation happens via the processes of breeding. The breeding process exists out of 3 distinctive steps: selection, crossover and mutation.

Selection: First, there needs to be a selection of members of the generation to breed with. The selection decides which members have a chance of creating offspring for the next generation, so the goal is to have good, but also diverse genes. One can for example choose only the fittest members of a generation, but this often leads to the homogeneity of the population and loss of diversity, which is equivalent to getting stuck in a local optimum. Another example would be choosing your selection randomly, but this would result in no real improvement as there is no preference for fitter individuals and is equal to random search. One widely used method for selection is binary tournament selection [2], which involves picking 2 members randomly and discarding the least fit of the two. In this way, there is improvement on average, but there is no bias against average performers and only the worst fit member is never chosen. This results in a good diversity of genes in the selection which are all at least better than one other random member.

Crossover: After the so-called parents are selected, crossover is performed. From the selection, each parent is matched with a random different parent from the selection to create one or more children. This process of creating a new child from the parents is called crossover because genes from both parents are crossed over which results in the new combination of genes that forms the offspring. There are many variations of crossover: most popular methods cross over complete parts of the gene, here either the first or second parent is selected for each gene and that exact gene value is copied over to the child, this is also known as discrete crossover and can be seen in Figure 2.4. This however comes with the problem that for each gene, the new value is always an exact copy of that of the parents, even though we might be more interested in a value that lies between the values of the parents. For this, we looked at another crossover method, namely extended line recombination. Extended line recombination is a crossover method first mentioned in [21], based on line recombination from [20]. Extended line recombination can, instead of copying genes from only one parent, get new values for the genes based on the genes of both parents. It does this by taking 2 parents, p_i and q_i as points in the design space and basing the new gene on the line between those two points as seen in Equation 2.7, here α is chosen uniform

randomly in [-0.25,1.25] and n is the amount of genes.

$$g_i = p_i + \alpha(q_i - p_i) \quad i = 1, \ldots, n \tag{2.7}$$

Another crossover method similar to extended line recombination is intermediate crossover, also from [20]. For intermediate, instead of the line between the two parents, the whole hypercube that forms between the parents are possible candidates for children. These versions of recombination can be found in Figure 2.4. As can be derived from the image, the [-0.25,1.25] causes both the line recombination and intermediate recombination to have offspring beyond the space or line between parents.

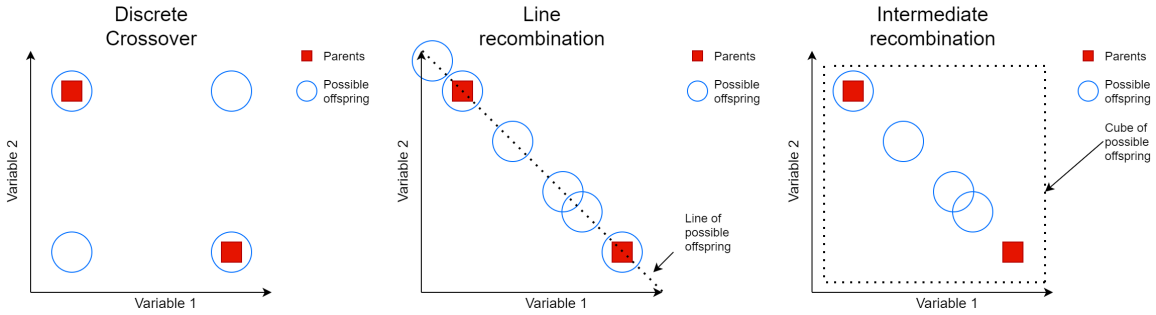


Figure 2.4: Three forms of crossover

Mutation: While crossover provides new combinations of genes, it does not introduce new variants of genes, all genes are inspired by one of the parent's genes. This is problematic because it would mean our complete DSE process is entirely dependent on the initial population where the first values for genes are generated. If no good selection of values is in the initial population then it is unlikely to ever find an optimum. To address this problem there is the concept of mutation: for each child, there is a chance that one or multiple of their genes "mutate" to a completely different value. Of course, if this happens too often, the exploration once again becomes random, but with some restraint is allows for the exploration of new possible gene parameters which could be more optimal.

2.4.3 Exploration and exploitation in evolutionary algorithms

As discussed in subsection 2.1.2, all DSE methods must balance both exploration and exploitation of the design space. Evolutionary algorithms are no exception. Exploration, when new designs are considered, happens mostly at the start and during mutation. There is also exploration during the crossover phase, as this generates new designs. However, for many crossover methods, these new designs do not contain new values for the genes and only cross over existing genes which does not add new information to the process. The intensity of mutation, i.e. how much is changed during a mutation, can be adjusted by changing the mutation parameters or the mutation itself, and this process will then happen every generation. The other method of exploration is the initial population generation. At the start of the evolutionary process, a generation is randomly generated as starting point. As this only happens once at the start, it makes the discovery process very dependent on initial conditions: if good individual genes are not included in the first generation, or are included in designs with bad other parameters, the discovery of good parameters is left solely to the mutation, while if the crossover method does not contain methods

for new parameter creation, which they often do not. This shows the biggest weakness of evolutionary algorithms which is their limited exploration of the design space. The exploitation however is very effective. Given a good solution, the mutation can easily perform what is similar to a "local search" where designs similar to a certain design are considered. Evolutionary algorithms employ elitism to keep the most optimal solution so far in the population and this combination of elitism and local search means that, locally, the best solution is found very quickly and efficiently, given good initial parameters.

2.4.4 SPEA2

SPEA2, the Strength Pareto Evolutionary Algorithm [5] is the most popular evolutionary algorithm in use today. It is an algorithm specifically made for multi-objective optimization, where it searches the design space for Pareto optimal solutions. SPEA2 adds to evolutionary algorithms an archive of all Pareto optimal solutions found so far. This archive is used instead of elitism, every new generation this archive is selected in addition to the best solutions of the previous generation to choose parents from. This results in Pareto optimal solutions never being lost after being found. If solutions are found which dominate the Pareto optimal solutions in the archive, the solution replaces those dominated solutions to form the new Pareto optimal selection.

2.5 Particle Swarm Optimization

Particle Swarm Optimization, also known as PSO, is a single-objective multi-agent optimization algorithm, initially proposed by J. Kennedy and R. Eberhart in [13]. It makes heavy use of parallel processing to search for an optimal solution.

2.5.1 Basic concept

PSO utilizes multiple agents, also known as particles, that are exploring the design space at the same time. All particles have their own position, velocity, acceleration and direction of travel. Each particle initially starts at a random position in the design space, and get assigned a random direction of travel and velocity. Each "timestep" all particles move in the direction of the travel at their given velocity and update their position. Then all particles are evaluated, as their position is just a place in the design space. After evaluation, each particle is updated with a new direction and velocity which are based on four different factors: the current velocity and direction, the location of the particle that found the overall best solution this timestep, and the location of the particle which found the best solution in its neighbourhood for the current timestep, where neighbourhood implies the ten closest particles. An example of a first generation is shown in Figure 2.5, here neighbourhoods are not considered due to the low amount of particles. At first we see the initialization, then the adjustment of the direction and velocity of travel based on the best solution found, and finally the new situation, after which the direction and velocity will be adjusted based on the new best solution and the cycle repeats.

2.5.2 Strengths and weaknesses

The strengths are the speed and expected coverage. With a large number of parallel agents, a good portion of the design space can be explored in a short amount of time, as the agents

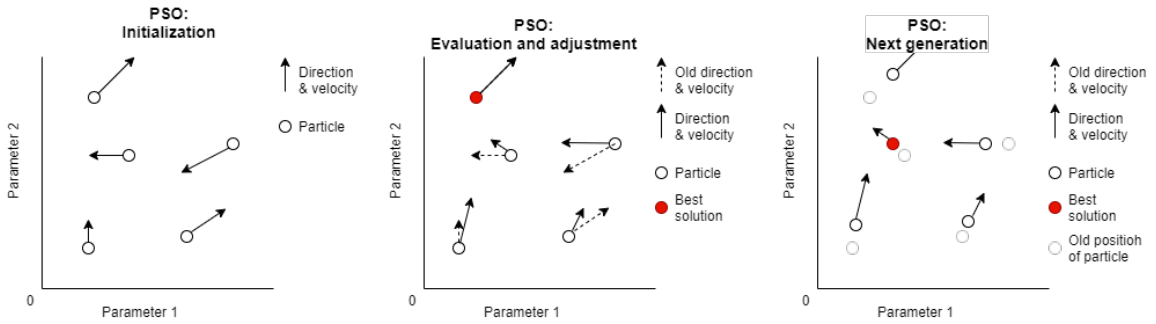


Figure 2.5: A single generation of basic PSO, including initialization

are all at different starting points and act partly individually. The downside of PSO is that choosing the right method parameters can be pretty hard, causing too slow or too fast convergence. PSO is most limited by the fact that it relies on travel of particles, which can not move too fast else their behaviour becomes irrational. This means that PSO has a big disadvantage when there are a limited amount of generations available: the particles can only travel so far during the given amount of generations.

2.6 Simulated Annealing

Simulated annealing, often shortened to SA, is a single objective linear design space exploration algorithm [14], inspired by the hill climbing method[24]. By digitally simulating the process of annealing metal, the method aims to start exploratory and become more exploitative over time, at an exponentially decreasing pace.

2.6.1 Basic concept

Annealing is the process of cooling down hot metal to avoid cracks forming in the crystal structure. Instead of cooling it down in a short time frame, the temperature of the metal is slowly lowered, so the crystals can move around freely to find an optimal formation, such that stress in the metal is reduced and cracks do not form. By slowly lowering the temperature the crystals slowly settle down in place and, if done correctly, when completely cooled down the crystal structure is very stable, in an optimal state. SA uses this same concept: it starts with a solution, then generates a new solution and considers the solution based on the current temperature: a value whose acceleration decreases negatively after every sample. When the temperature is high, then SA can "choose" a worse solution as its next best found solution. As the temperature decreases, the chance of SA "choosing" a worse solution is also lower. At the same time, if the new solution is better, it will always choose that solution over the current one no matter the temperature. The temperature is simulated to go down until the chance is 0 and it can only optimize. Due to the temperature changing, the method starts very exploratory but gets more exploitative over time. The probability equation of accepting new solution j is shown in 2.8 where t indicates the temperature, which decreases over time, f is the fitness function and P_t indicates the probability of j being accepted at temperature t.

$$P_t(accept(j)) = \begin{cases} f(j) \leq f(i), & 1 \\ f(j) > f(i) & e^{\frac{f(i)-f(j)}{t}} \end{cases} \tag{2.8}$$

The new solution j is chosen randomly using a normal distribution. If solution j is better than the current best solution i, j is accepted as the new best solution. If j is not better than the best solution i, there is a random chance based on the temperature and difference in fitness between solutions i and j which decides whether or not j will be taken as solution under consideration. This chance is $e^{\frac{f(i)-f(j)}{t}}$. As is apparent, the higher the temperature t is, the higher the chance P_t that solution j is accepted. Take note that $f(j) > f(i)$ which in turn implies that $f(i) - f(j)$ is negative.

2.6.2 Strengths and weaknesses

The strength of the Simulated Annealing algorithm is that it is simple and robust, it is very easy to reason about how the method will search the design space, and also easy to intuitively adjust parameters of the method according to the requirements. The downside is that the method is not flexible; its moment of convergence is set from the start. This makes it very reliant on initial conditions: if there is too much time the method is wasting resources, however, if the convergence happens too soon it will become stuck in a local optimum while the probability to escape decreases due to the continuing temperature decrease. Another weakness of the algorithm is that it only considers new points in its neighborhood, making it harder to explore large design spaces.

Chapter 3

Setup for DSE

For the project, a setup for automated design space exploration was made. This was done in multiple steps. In this chapter, the setup is explained.

3.1 The process of DSE

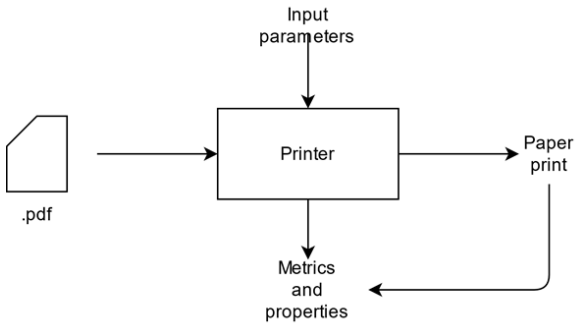


Figure 3.1: The traditional evaluation setup for printer design

The process of manual design space exploration for large complicated machines such as production printers is laborious and time-intensive. The setup can be seen in Figure 3.1: A design is proposed, specifying all input parameters, a prototype testing setup is built and a test case is run, resulting in various metrics and properties of the printer and the resulting print which need to be evaluated and analyzed. When this is completed, based on the evaluation and analysis, a new design is proposed, with new parameters. This process repeats until a satisfactory design is found at which point the next stage of development can begin.

3.2 The model

To help in this process, models can be developed which simulate various parts of the printing process. For our case, a model was developed that predicts the intensity of light at arbitrary positions in the print surface. The model provides an impression of how the metrics of the real printer will turn out.

3.2.1 Core concept

The core concept of the model is to simulate lamps moving over the printing surface while measuring the intensity of light reaching the top of the ink layer. The main elements which are to be simulated are therefore the lamps, which have a well defined light profile that describes the intensity of light received on various positions on a surface below the lamp at a well-defined height, the paths of the lamps, which describe the trajectory of the lamps and the printing surface. The printing surface moves horizontally while the lamp moves vertically (swath direction), independent from the movement of the printing surface. This setup can be seen in Figure 3.2.

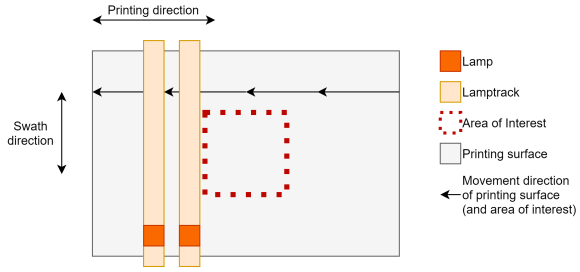


Figure 3.2: The real elements which are modelled

3.2.2 Simulation

The lamps are abstracted to only their light distribution as it reaches the printing surface, also referred to as the lamp profile, which is shown in Figure 3.3 where a sideways view of the real setup is given. This profile is two-dimensional and an example is given in Figure 3.4.

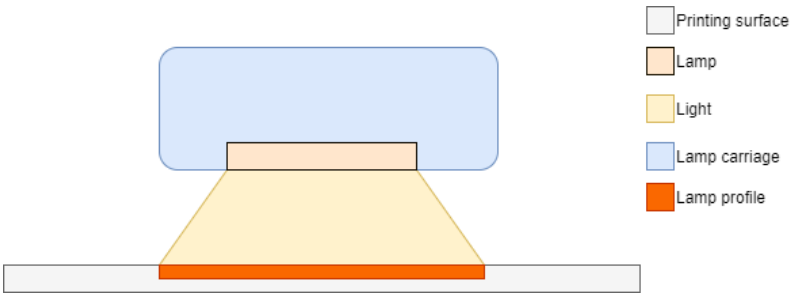


Figure 3.3: Sideways view of how the lamp

To simulate the printing surface, a 2D matrix is used. Each element, or pixel, in this 2D matrix represents a 2mm by 2mm surface. To simulate movement, we add a discrete-time dimension and move the lamp profile over the printing surface. During this movement, the local light intensity is added to the corresponding pixel of the printing surface, multiplied by the time of the irradiation and surface area of each pixel. This results in the accumulation of UV light on the printing surface. In this way we simulate the dose received by each pixel during the printing process as function of time.

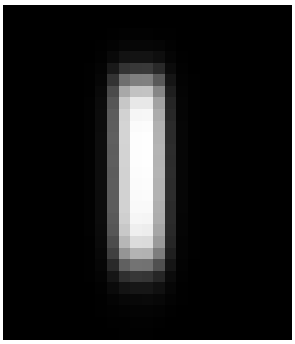


Figure 3.4: An example of the post curing lamp lamp profile

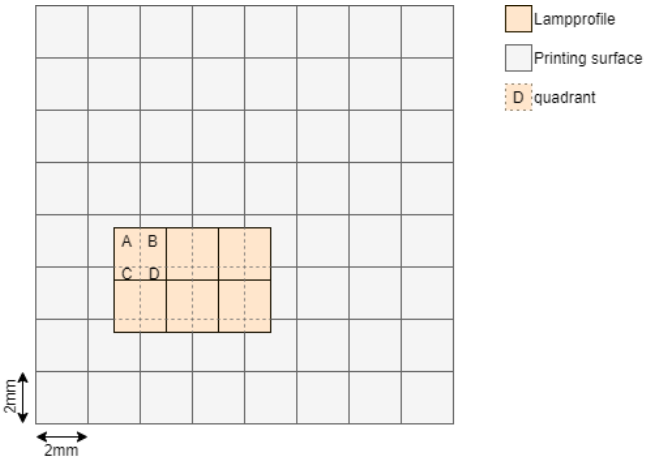


Figure 3.5: A single timestep in the simulation

Quadrants

As can be seen in Figure 3.5, the pixels of the lamp do not necessarily overlap one-on-one with the pixels of the printing surface. Because of this, every pixel of the lamp is divided into 4 quadrants for each timestep, and the value of the lamp pixel is multiplied by the area of the quadrant before being added to the printing surface pixel the quadrant is located on. The size of the quadrants is the same for each light pixel, so the area calculation needs to happen only once for each timestep and not for each pixel each timestep.

Details of the simulation

In the final model, there is a 1000 by 1000 pixel printing surface, on which an Area of Interest of 100 by 100 pixels is selected. Each pixel is 2mm by 2mm which makes the total surface of the Area of interest 20cm by 20cm. The time step was set at 0.0001s, which means, at a maximum speed of $1m/s = 1000mm/s = 1mm/timestep$, so the lamp moves over the surface at a maximum speed of half a pixel per timestep which ensures that a pixel on the light profile passes does not "skip" a pixel on the printing surface. Each lamp is also divided into pixels of 2mm by 2mm, and for each pixel the light intensity is given in units of J/m^2, which is then added to the pixels of the printing surface they are covering, as per quadrant size every timestep.

3.2.3 Lamp paths

In the real setup, the lamps move only in the swath direction, while the media moves in the printing direction. For our model, we use a relative path: the way the lamp moves relative to the media. To convert the relative paths back to individual lamp and media movement, one must take into account that the media stops moving on a set interval to allow the printing head to jet ink, which is not taken into account in the relative path. To convert from our relative paths, to the actual paths, in simple terms, the printer step needs to be introduced, by temporarily stopping the lamps during this moment.

The path of the lamps is characterized by a collection of points visited, which indicates the location and time when the lamp would be there. The actual path was interpolated using the timestep between points. This very unrestricted input format meant that there would be a very large amount of inputs that resulted in infeasible designs, including designs in which the lamps could move backward over the printing surface, teleport over the printing surface randomly, etc. Such designs are unrealistic and should be discarded. Instead, a more restricted parameterization of the lamp path needs to be created, which still covers all feasible designs, but will not allow most infeasible designs: designs which can never return valid results should be excluded from the to be explored design space as much as possible. The feasibility is based on engineering restrictions and/or the cost of the design. The goals for the design generation are as follows:

- **Repeatable** The total path is a sum of repetitive motions that can be extended indefinitely in time.
- **Only left to right movement** The lamps are only allowed to move in the print direction (forward only), while they are able to move freely in the swath direction.
- **Minimal redundancy** It is not useful to have 2 different designs which are functionally the same and therefore also result in identical fitness values. This would result in a larger design space than necessary and create multiple optima in different places which makes the search more complex.
- **Limited amount of inputs** The more inputs, the harder it is to explore as this increases the size of the design space exponentially.

Based on these goals, the idea of using a repeating pattern was born. This pattern will be generated based on a few inputs, and then repeat itself until the complete area of interest was covered by the lamps. This resulted in the following parameters, which form patterns visualized in Figure 3.6:

- D_l The Delta, or length of the repeating pattern. The repeating patterns are all of length D.
- C_l A parameter that influences the slope of the path during vertical movement. At 1 it is the lowest angle possible, while at 0 the angle is 90 degrees.
- R_l The Ratio between the up and down strokes. At 1 they are equal; at 0 the downstroke is of length 0.
- PR_l Short for Power Ratio, indicates the difference in lamp power between the up and down strokes.

- W_l The width of the path. At 0 the path is a straight line through the middle, at 1 the strokes go over the complete area of interest.

The generator generates 2-dimensional points, P, which form a repeating horizontal pattern using the parameters mentioned before as follows:

$$0 \leq R \leq 1$$
$$0 \leq C \leq 1$$
$$0 < D$$

$$P_{i,0} = [D \cdot i + \frac{R}{1+C} \cdot D, 0.5W] \tag{3.1}$$

$$P_{i,1} = [D \cdot i + \frac{D}{1+C}, 0.5W] \tag{3.2}$$

$$P_{i,2} = [D \cdot i + P_{i,1}.x + C \cdot \frac{R}{1+C} \cdot D, 1 - 0.5W] \tag{3.3}$$

$$P_{i,3} = [D \cdot i + D, 1 - 0.5W] \tag{3.4}$$

In simple terms, R decides whether or not there is space on the x-axis between Equation 3.1 and the previous point $P_{i-1,3}$ and between Equation 3.2 and Equation 3.3, when R is 0, both points are at the same x value, and the vertical movement is completely vertical as the points which are at the same x value have different y values, given that W is larger than 0. When R is larger than 0, the offset in the x-axis becomes bigger forming a slope between the points instead.

C is the parameter that governs the ratio between the up movement and the down movement, at $C = 0$, the lamp moves immediately up in a straight line once it reaches the bottom. For C higher than 0, the lamp movement on the bottom becomes more like how it behaves on the upper side, becoming completely identical and forming a symmetric pattern at $C = 1$ as can be seen in Figure 3.7. C is included in the fractions in Equation 3.1, Equation 3.2 and Equation 3.3 to keep their offsets relative, and used in Equation 3.3 to move the point between $P_{i,1}$ and $P_{i,3}$, creating different pattern types. The extremes of this pattern generation are shown in Figure 3.7, but any value between 1 and 0 is also possible, resulting in a wide array of possible patterns.

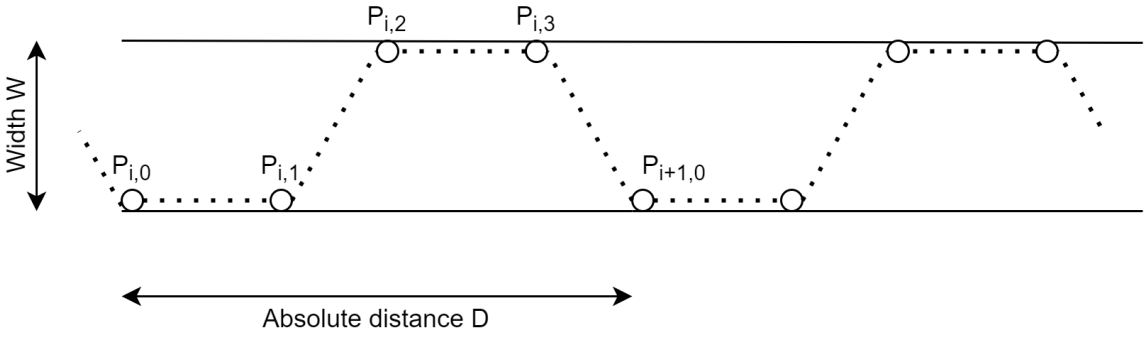


Figure 3.6: The 4 points of P_i, pattern number i, which form a pattern, visualized

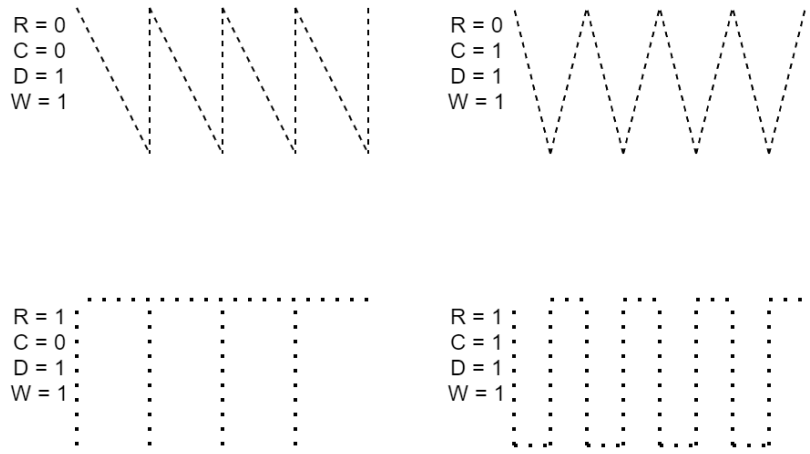


Figure 3.7: The 4 extremes of the pattern forming parameters, R and C.

3.2.4 The lamps

Custom lamp profiles can be generated based on input parameters. To do this, the following input parameters were created.

- X_l the x-dimension of the lamp.
- Y_l the y-dimension of the lamp.
- p_l the base power or intensity of the lamp.
- STD_l represents the length of the falloff of the lamp.

The lamp profiles are generated as follows using these parameters. First, the main lamp area is generated using the X_l and Y_l parameters as width and height, and this area is given intensity p_l. Around the lamp a falloff area is generated based on STD_l, which, according to half a Gaussian distribution approaches 0 at the ends facing away from the lamp area. This distribution is rounded down to 0 when under 0.01. This is visualized in Figure 3.8.

In the corners of the lamp profile, the falloff is the product of the horizontal and vertical falloff, which results in a more steep decline in the diagonal direction away from the lamp.

3.2.5 General parameters

These are the parameters that are not specific to a lamp or its path. O describes the offset between the lamps at the start, as a different offset allows for different interference patterns given the same paths.

- O the offset of lamp 2 from lamp 1
- S the speed of the printing surface

3.2.6 Model summary

The resulting model has a design space covering all feasible designs and excluding most infeasible designs, without redundancy and excellent performance results. The model allows for the usage of multiple lamps of any size and with variable speed and detail of simulation.

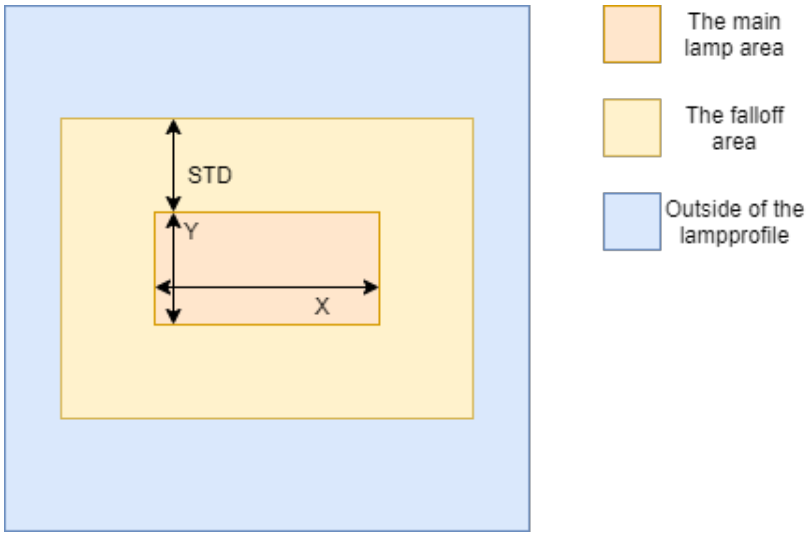


Figure 3.8: The parameters and generation of the lamp profile.

The path generator allows the creation of almost any pattern consisting out of 4 straight lines which move up and down, without requiring many parameters to generate it. At the same time, the path generator allows any input within the bounds, allowing the paths to smoothly morph into other patterns. This again allows for continuous exploration methods to be more effective such as hill climbing as it means similar-looking input parameters also generate similar-looking output. Now, our setup looks as shown in Figure 3.9.

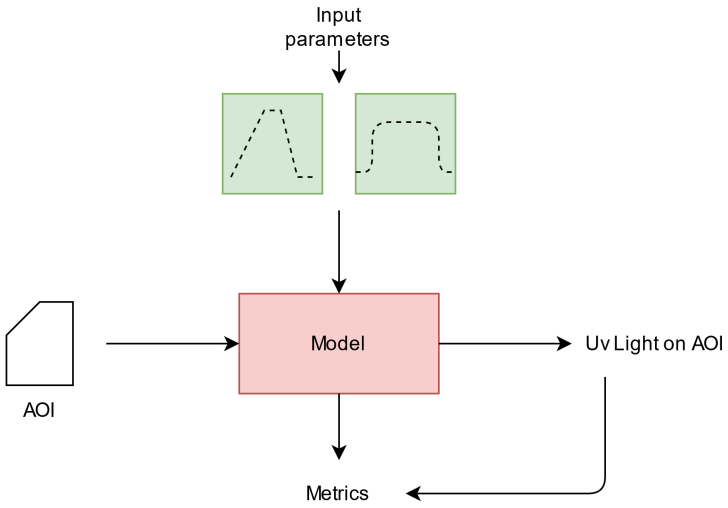


Figure 3.9: The setup, with the new generators

3.2.7 Deprecated parameters

During the process of development, various design spaces were tested, these used a different set of parameters compared to the final model, but these are still of interest as older results are also discussed in the analysis.

- LS short for LampSwitch, continuous parameter between 0 and 1. Indicates which lamp is used using the following relation: utilization of lamp1 = $min(1, 2 - 2 * LS)$ while utilization of lamp 2 = $min(1, 2 * LS)$. At 0 both lamps are fully utilized, while at the extremes 0 and 1, only one lamp is fully utilized while the other is unutilized.
- S_l Speed, the speed at which a path is traversed per lamp. Deprecated because it allowed lamps to move at different speeds horizontally as well.

3.3 Model bounding

The design space should cover all designs which could be of interest. Various elements are limiting our design space naturally, such as design constraints and limitations imposed by reality. We weigh two things during the introduction of model bounds and constraints: we want to include as many possible designs in our exploration to be sure that there is no good design outside of the design space, but at the same time, we want to limit the size of the design space as much as possible to make exploration of the design space faster.

3.3.1 Deciding on bounds

For our design space, we chose to only bound our input parameters individually, based on physical constraints. This means that a parameter itself is always within feasible bounds, however, a combination of parameters might not be. For example: the lamp might realistically be limited in its surface, but we range our length and height parameters to their respective upper bounds. This means that unrealistic cases can exist, for example where both the height and width are at their maximum, surpassing the feasible surface, but not the feasible width or height individually: a lamp with large height and small width or vice versa is perfectly possible. In these cases, the model detects these infeasible designs, described in subsection 3.3.3, and rejects them by returning a very bad fitness for the design, indicating to the method that these designs are unwanted. This allows for the broadest range of inputs while keeping the designs close to the bounds of feasibility and reality.

3.3.2 Design requirements

A few constraints were embedded in the model to ensure that the resulting designs would be feasible. If these constraints are not met, the model returns a bad fitness for the design.

- Minimum speed of the print: a minimum printing speed was required, measured in 40 square meters per hour.
- Maximum Speed of the lamp carriage: the lamp was not allowed to move at a speed above 1 meter per second.
- Minimum total dosage per pixel: a minimum of 8000 J/m^2 was set as the minimum requirement for each “pixel”.

These requirements are measured while the model is running, as they cannot be all be derived directly from individual input parameters. For example, the dosages for all pixels are not known beforehand given all inputs, only after simulation will the actual dosage for the area of interest become known. If these requirements are not met, the resulting evaluation will return the worst evaluation possible, signaling to the DSE method that these

input parameters are unwanted. This is visualized in Figure 3.10, where the white areas are normal samples, but samples in the red area have a very bad evaluation. This makes sure no infeasible design is eventually chosen, however, it does give rise to the challenge of getting a feasible sample; if a method does not find a feasible sample, it results in a failed run.

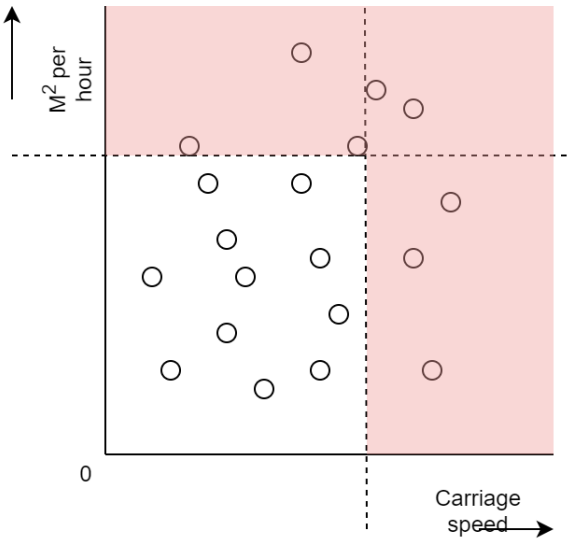


Figure 3.10: 2 of the model boundaries visualized

3.3.3 Input parameter boundaries

The final boundaries are as follows:

- $R_i = [0, 1]$: This is a relative parameter; The absolute bounds are dependent on other parameters.
- $C_i = [0, 1]$: This is also a relative parameter.
- $PR_i = [0, 1]$: The percentage of power, divided by 100.
- $W_i = [0, 1]$: The width of the path divided by 20cm, bounded to 20cm, or 1, as having a lamp significantly wider than the printing area has no additional benefit.
- $D_i = [10, 100]$ translates to 2 to 20cm. Below 2cm the pattern requires too much time to finish printing in time, while above 20cm is longer than the area of interest so the pattern would not be totally used.
- $O = [0, 100]$ translates from 0 to 20cm. At 0 cm the lamps have no gap between them in the printing direction. At 100 this gap becomes 20cm, which is as long as the maximum pattern path, so the interference pattern would be identical to having no gap.
- $S = [10, 200]$ in 2mm/s; less than $10 \cdot 2mm/s$ is too slow to be feasible, while $200 \cdot 2mm/s$ is the fastest the printing surface will move relative to the lamp.

- $x_i = [5, 110]$ in $2mm$; $5 \cdot 2mm$ is the smallest lamp size available, while $110 \cdot 2mm$ is larger than the area of interest.
- $y_i = [5, 110]$ in $2mm$; $5 \cdot 2mm$ is the smallest lamp size available, while $110 \cdot 2mm$ is larger than the area of interest.
- $std_i = [3, 12]$; 3 was inspired by the existing lamp profiles' minimum spread, and 12 as maximum as otherwise the lamp profiles will become infeasibly large.
- $p_i = [0, 200]$ in W/m^2; 0 W/m^2 as minimum because the lamp should also be able to be turned off and 200 W/m^2 as larger intensities are not feasible or result in safety hazards.

3.4 Evaluating the designs

Now that we can generate designs and simulate them, the next step is analyzing the results using an evaluation method. If we can automate the evaluation, it becomes a lot easier to judge a design without human intervention during the process of automated DSE. The goal is to minimize the local gradients in the x and y directions, and in the time direction. To do so, we record the difference in dosage every timestep t on the printing surface between pixels in both the y direction and x direction using Equation 3.5 and Equation 3.6 respectively. Here, $\delta_{[t,x,y]}$ is the accumulated dosage for the pixel at location (x, y) in the area of interest at moment t. Equation 3.7.

$$dy_{[t,x,y]} = |\delta_{[t,x,y]} - \delta_{[t,x,y+1]}| \tag{3.5}$$

$$dx_{[t,x,y]} = |\delta_{[t,x,y]} - \delta_{[t,x+1,y]}| \tag{3.6}$$

To evaluate the designs, a simple single objective evaluation was used based on the difference in local curing dosage by taking the largest difference found at any point in time in

$$obj = max_{t,x,y}(dx_{[t,x,y]}, dy_{[t,x,y]}) \tag{3.7}$$

When we minimize this objective, the end result will be the smallest maximum difference in accumulated dosage in the area of interest at any time. This objective was chosen as it is expected that large differences in the curing dosage between pixels at any time will cause the unwanted wrinkling effect. Having a small objective then means having minimized wrinkling. Our process now looks like Figure 3.11.

3.5 Automating the DSE process

We used an existing tool for automated exploration to save development time and also prevent any errors or bias compared to making our own tool for exploration. In this section we describe various candidates for DSE tools and our rationale for choosing a particular tool.

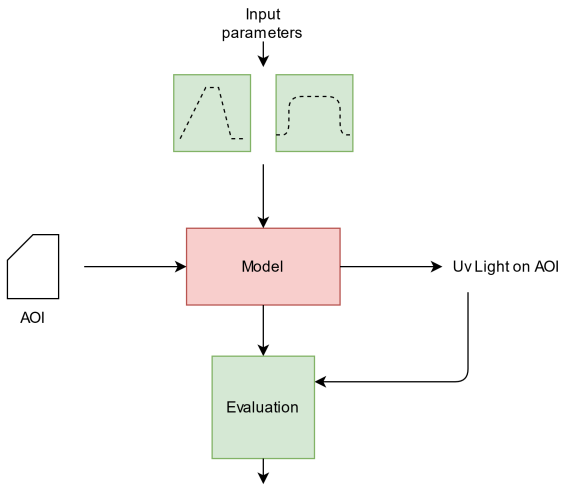


Figure 3.11: The setup, with evaluation

3.5.1 Deciding on the tool

As our original goal for the project was multi-objectivity, the main considered tools had as a requirement that they would support evolutionary algorithms, which are the most popular multi-objective DSE algorithms. However, during the project it was decided to not use multi-objectivity for our example case of UV-curing, but nonetheless we stuck to this requirement as it may become useful in the future.

JGAP

JGAP, also known as the Java Genetic Algorithm Package is a Java Package, is a Java Package for running evolutionary algorithms [18]. This package however is very limited in its flexibility and also was not written with parallel processing in mind. Next to that it also lacks good multi-objective DSE options which made it less attractive for our initial expected use case. Another problem found was that it was abandonware, as the software has not been updated in multiple years and issues with the program raised by users were not being addressed.

Matlab GO toolbox

The Matlab GO toolbox, where GO stands for Global Optimization, is a strong contender [27]. It contains many different algorithms and options. However, Matlab itself is somewhat inflexible, being closed source which makes analyzing the methods harder. Next to that, the GO toolbox is not free.

Shark

Shark is a machine learning library with options for single and multi-objective optimization. It benefits from many features but has not been updated recently. Next to that, evolutionary algorithms are only a small element of Shark. It lacks any alternative optimization methods and focuses mainly on machine learning.

EvA2

EvA2 is the Evolutionary Algorithms workbench developed by The University of Tübingen [30]. It is an expansive tool covering many algorithms and has been developed with parallel processing in mind, making it very fast. Next to that, it is open source and very flexible, having many options to do almost anything. The main downside is that it has not been actively developed in the past 5 years.

ECJ

ECJ, or Evolutionary Computation Java, is a toolbox for performing evolutionary algorithms and many other methods[25]. Like EvA2 it is very flexible and also benefits from speedup from parallelism. It has received grants from various institutions for further development and is still being actively developed to this day.

Final choice and adjustments

Due to the many upsides, both EvA2 and ECJ were tested. In the end, ECJ was chosen due to its up-to-date development and extensive documentation. After the requirement of multi-objectivity was dropped for the use case, JGAP could have made the shortlist, however the lack of support and fact that it is abandonware would have still prevented it from being chosen as the final tool.

Some small additions to ECJ were necessary expands it capabilities and usefulness greatly. The main addition was the ability to use python scripts to define problems or models. Although this change was simple to implement, merely forwarding a call with parameters to a python script and reading the output of the python script, it allowed for development of the model in python instead of in ECJ. This means that there is no recompilation needed for each update or change to the model and that the model can be easily interchanged with another model. There is a small increase in runtime, less than 1 second for each forward of the call, but this addition is insignificant compared to the running time of our model.

3.5.2 DSE Methods

After choosing the tool, there was not only one meta-heuristic method available, but multiple. This allowed us to choose methods based on the ones which were available in ECJ itself. Firstly an evolutionary algorithm was chosen, and then very different types of metaheuristic methods: SPEA2 employs an evolutionary algorithm, PSO employs a moving particle optimization methods, and SA employs a hill-climbing inspired method.

SPEA2

The Strength Pareto Evolutionary Algorithm 2, SPEA2, was chosen as an example of an evolutionary algorithm as described in Section 2.4. SPEA2 has attracted much attention due to its good performance in multi-objective design space exploration. In our case, the problem is single-objective, which is expected to make SPEA2 perform a lot more exploitative due to having a singular best solution instead of a set of Pareto optimal solutions as is the case in multi-objective optimization. An alternative that was considered was NSGA-II (Non-dominating Sorting Genetic Algorithm), which is very similar in its workings to

SPEA2, however, it has various downsides such as worse performance for designs with binary represented values for their parameters[7].

SPEA2 Extended

During the project, it was found that SPEA2 itself was not performing optimally based on the results in Chapter 5, so a focus was put on improving SPEA2 to be more exploratory and have less chance of falling into a local optimum. This resulted in the development of SPEA2 extended. By increasing the chance of mutation to 90%, and changing the crossover method to intermediate recombination as described in subsection 2.4.2, SPEA2 becomes a lot more exploratory.

PSO

Particle Swarm Optimization was chosen as a single objective design space exploration method that made heavy use of parallelism. This method differs greatly from genetic algorithms and is based on agents moving through the design space with a motion vector. This is another popular class of optimization algorithms and PSO is a very good representative for methods that exploit parallelism for a significant speedup. The expectation is that PSO will perform very well on the problem, given the right input parameters.

SA

Simulated Annealing, SA, is a single objective design space exploration method that moves through the design space mostly considering designs in the neighborhood of the current design under consideration. It is a very simple method that has proven to be popular in literature and by extension many engineering problems[17][11][19][9]. Although newer methods have now mostly surpassed SA, SA remains an interesting method to consider due to its simple properties which make it ideal to analyze the design space and adapt SA itself.

Random explorer

The random explorer is a single objective design space exploration method written for this project as a reference. It is exclusively exploratory and has no exploitative capabilities. It is of interest to compare other methods against the random explorer. The random explorer will not only show ideal values for the exploration measurements but also act as a control: if a method performs worse than the random explorer, it implies that the logic is working against it and not helping.

3.5.3 Environment

Now that a tool and its methods have been chosen, our setup looks like Figure 3.12. We used a 20 core, 40 thread system which was advantageous for our parallel methods and caused great speedup. The server ran Linux, and the python version used was python3.

3.6 Comparing and evaluating DSE methods

Although automated DSE is now possible, we must still be able to determine whether the resulting solution is optimal, how long a method should continue searching for better

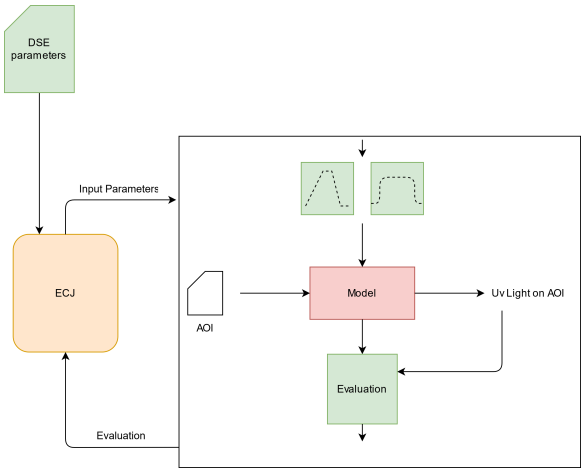


Figure 3.12: Our setup, including the new methods and ECJ

solutions, whether we are sure if the method performs well repeatably and not just randomly, etc. This is where testing and analysis of the methods come into play.

To correctly compare and measure DSE methods, a robust testing setup is needed. In our case, ECJ will be used which contains the basic tools for performing various DSE methods. As most methods are not deterministic, a single run is not necessarily representative of the general performance of the method itself. It is therefore necessary to run each method multiple times and combine the results so we can improve our confidence in the expected behavior and results of a method.

3.6.1 Goals

The goals that we want to achieve are as follows:

- Perform a sufficient number of runs to have statistical confidence in the performance results of our methods
- Compare the DSE methods with each other in an unbiased manner
- Analyze the behavior of the DSE methods over time

To achieve these goals, a selection of tests is performed. In our case, it was decided to perform 30 independent runs per method. The main motivator was time. A single run takes multiple hours to complete, limiting the number of tests possible as we also need the option to adjust methods or re-do tests. The data per method and the amount of tests are noted in Table 3.1 and Table 3.2 respectively. Unfortunately, SA performed very slowly due to not benefiting from speedups from parallelism, which resulted in only limited testing.

3.6.2 Tests

Algorithm	Evaluations per generation	Generations per run	Evaluations total
SPEA2	80	40	3200
SPEA2 Ex	80	40	3200
PSO	80	60	4800
SA	2	800	1600
Random	1	3200	3200

Table 3.1: Evaluation and generation data per run

Algorithm	Total number of runs	Total number of evaluations
SPEA2	30	96000
SPEA2 Ex	30	96000
PSO	30	144000
SA	30	48000
Random	30	96000

Table 3.2: total number of tests performed

3.6.3 Implementation

To perform all these tests, an additional script was made to both perform multiple runs of the same method and to capture all the output data, resulting in our final testing setup as seen in Figure 3.13; a detailed explanation of what was done with the data captured is given in Chapter 4.

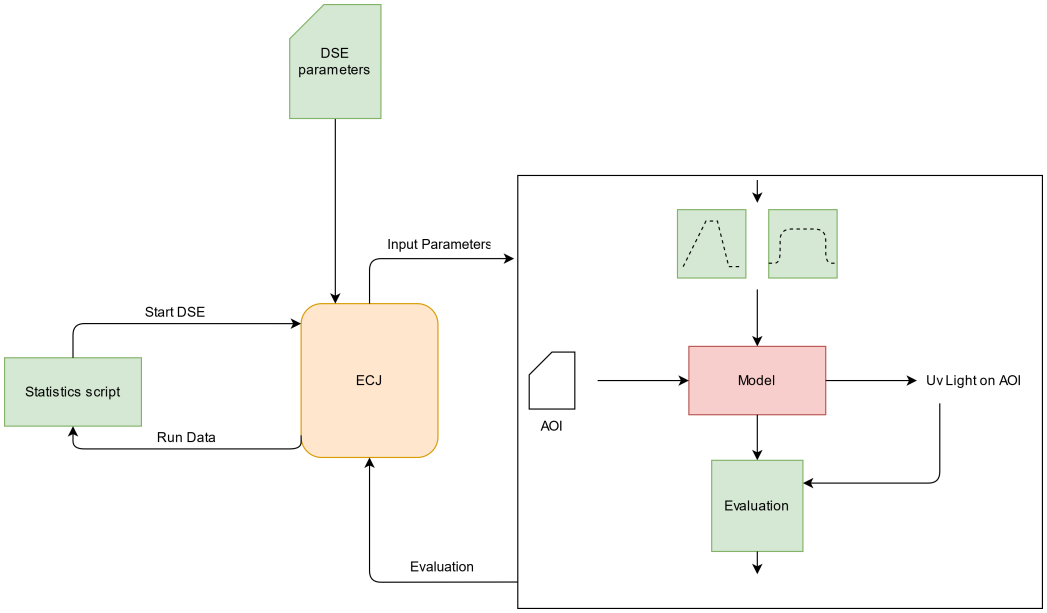


Figure 3.13: The final testing and DSE setup

3.7 Final DSE setup

The final setup now contains a module taking inputs for a design and generating output in the form of an evaluation of the design. This module is used by ECJ which generates designs and uses the module to evaluate the designs. ECJ uses the DSE methods described by the DSE parameter file. ECJ itself is called by a statistics script that starts the metaheuristic methods and captures the generated data. This setup allows for the automated search of our problem, but also other problems if needed, the module can simply be replaced by one containing a different problem. It also allows for easy switching of the metaheuristic method using the DSE parameter file, as long as it is available in the ECJ program.

Chapter 4

DSE measurements and analytics

Testing of DSE methods and processes is something yet to be explored in-depth, and no standardized methods exist. There are some problem sets that aim to give a neutral view of how a DSE method performs, but these problem sets mostly consist out of artificial problems of a comparable class, such as optimization of mathematical equations[12] [29] [32], which do not reflect the variety of problems one might encounter in practice. There has also been research done for designing these test problems [4]. These testing methods often do not take into account the parameterization of methods, or optimize the parameters of some methods but leave some as-is due to unfamiliarity with the problem or method itself. All in all these test sets do not give a clear view of how the DSE is performed, as only the final result is considered, not the process or features of the methods. In this chapter, we describe the various methods used for testing the effectiveness of the DSE methods.

4.1 Measurements for analysis

When comparing different methods one needs compatible measures that are properly defined for each of the methods in comparison. Here we describe the quality measurements we used for comparison.

4.1.1 Pitfalls and caveats

Automated design space exploration methods are reactive, and so all of them change behavior over time. For example: for the coverage measure of the design space, an aim could be to have as large a coverage as possible, however, the expectation is that a good DSE method will eventually converge to a solution. The expectation is therefore that eventually, the samples will concentrate around locations where good solutions are expected. A simplistic visualization of this behavior is shown in Figure 4.1: at different stages in the design exploration process, the importance and relevance of quality measures vary. To account for this, not only is there testing of all points all at once but the performance over time is also considered, from early stages of discovery and exploration to the moment(s) of convergence and exploitation.

Combining of data

As multiple runs will be performed, it is important to take into account that all data should be directly comparable. Given two runs of the same algorithm, they might both find

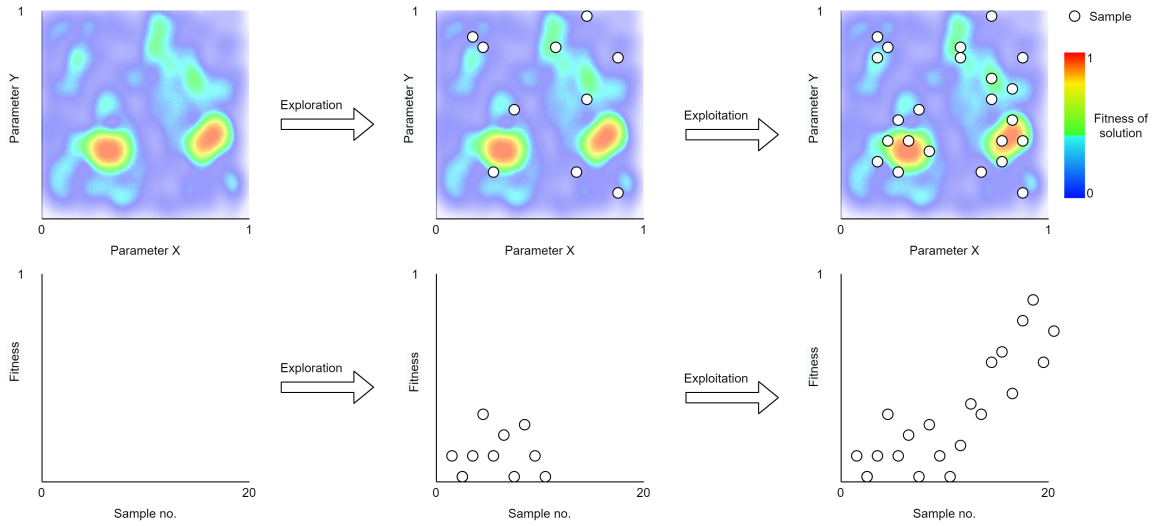


Figure 4.1: A simplistic example of the expected sampling behaviour during DSE with two parameters.

different optima in different places or find the same optima via different routes through the design space. The runs might both have explored only one half of the design space, which is unfavorable, but if combined it might seem as if though the exploration covers the complete design space evenly. If the data of multiple runs is combined naively, such important details of individual runs might be lost. It is therefore important to combine data without losing important data on individual runs. In the given example one might average the percentage of design space explored, which would give a more relevant measurement.

Outliers in individual runs

Outliers are runs that do not match the average; they might by chance find a good solution very fast or miss the optimal solution by chance. In either case, these are important: they show faults in the DSE method or show that the other runs are all missing the best designs. The question is how to consider them, as they can skew the results. In the case of failing runs, it is important to take into account that the exploration of the design space might be insufficient, being unable to find a valid solution. We consider one failing run to be acceptable, leaving it out of the dataset, but two or more failing runs to be a pattern indicating that something is wrong with the method itself. In the case of quick convergence, there is less skewing of the data and these results are averaged out, as good results are desirable.

Objective data

We have two datasets per method we are using. The first dataset contains, for each generation of a method, the fitness of each run. This will give us insight into exploitative qualities and the expected progression of the methods. The second dataset which is used, is all considered designs, that is to say, samples from the design space. This will give insight into the explorative qualities of the methods. So if there are 30 runs of 40 generations of 80 members, that means for the first dataset, there are 30 arrays of 40 data points that show fitness progression over time. For the second dataset, that means 30 arrays of $80 \cdot 40$

designs, which themselves exist out of 20 parameters, giving us a total of 96000 designs or samples. These designs are stored in order of discovery per run.

4.1.2 Exploitation

In the case of well-functioning exploitation, it should be hard to improve upon a resulting solution. This process of exploitation happens by local search, or what effectively amounts to a local search, such as mutation in evolutionary algorithms. If it is possible to easily find a better result close to the found optimum e.g. by a manual search of neighboring designs, then the exploitation did not exploit the design space as effectively as possible. It should be noted, however, that optimization takes considerable time, and if the optimization method does not have enough time it is unreasonable to expect it to have found a good solution already.

4.1.3 Exploration

Exploration of the design space is important because it results in confidence in the found optimum: if many different designs were considered, then we can be more confident in our final design being optimal. Measuring exploration, however, has no readily available method. To address this we have come up with various measurements, which together should give a good impression of the exploratory capabilities of a method. Measurements are explained below.

Mean variance of all parameters per generation

The measure of the mean-variance shows us how dispersed the sampled points are per generation. This only considers the spread of the samples and no relation between samples. As the average is taken over all parameter types it gives a general idea about the dispersion of a generation. If this number approaches 0, all points are nearly identical, which means the meta-heuristic has converged to a single solution.

Uniqueness of solutions

Using the uniqueness measurement, explained in subsection 2.3.2, it is possible to track how unique the newly sampled designs are in comparison to the already sampled solutions. The running mean of all samples of all runs allows us to compare various methods. The uniqueness is expected to decrease, given that the more unique points that are sampled, the less unexplored combinations of input parameter values remain in the design space, as can be seen in case 3 in Figure 4.2 where the maximum uniqueness possible for any new sample is small compared to the beginning stages as shown in case 1. The random explorer shows this expected decrease due to its near-perfect exploratory capabilities. Although the results themselves do not indicate a good or bad method, they *do* indicate how fast the results converge and how different methods explore the design space and their effectiveness in doing so compared to each other.

For this measurement, it is possible to combine multiple runs and take the mean for all runs, as these results are not dependent on initial conditions. It should be noted, however, that convergence to an optimum can happen at different moments for each run. This is not of interest as we are looking for a method that generally performs well and not one which only does better in a single run.

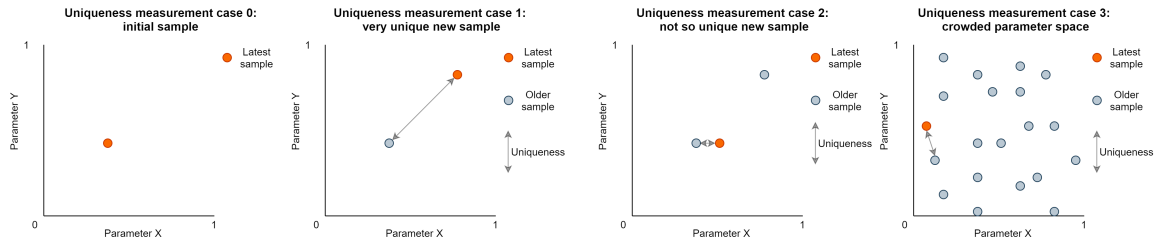


Figure 4.2: These 4 graphs show different dispersions of samples and numbers of samples in a two dimensional design space and how the uniqueness measurement measures new points in the design space

Gaps in the individual parameter dimensions

When sampling the design space, it is of interest to explore as broadly as possible. Whereas it is hard to detect how much percent of the region has been explored, it is relatively easy to detect which part is unexplored. The spaces between neighboring samples can be measured per parameter, per dimension. Knowing the size of the largest gap in the parameter dimension per run of a method, we can average them and compare the average to other methods. Then the comparison is in between how large the average gap is of each method and how this progresses over time. Large gaps in a parameter dimension are undesirable, though they do not necessarily need to be minimized in size. This results in the following equation for *gapsize*: Equation 4.2, where *sp* is defined in Equation 4.1. Here *points* is the set of all data points, *sort*() sorts a list from low to high, p indicates a parameter, and n is the number of samples taken. The $+\!\!+$ operator is the list concatenation operator and $[x]$, where x is a number, indicates a list with element x.

$$sp_{p,n} = [0] +\!\!+ \, sort(points_{p,n}) +\!\!+ \, [1] \tag{4.1}$$

$$gapsize_{p,n} = max_j(sp_{p,n}[j+1] - sp_{p,n}[j]) \tag{4.2}$$

The gap size is dependent on the number of samples taken, if one takes only a few samples, the gaps are expected to be large such as in case 0 in Figure 4.3. As more samples are taken the gaps should become smaller; a more ideal example can be seen in case 1, points are spread out evenly and there are no big gaps for the number of samples taken. However, as exploitation happens the method will focus on searching more locally instead of exploring more and filling its own gaps. If a large gap occurs, such as in case 2 of Figure 4.3, it is a clear indicator of something bad. The lack of large gaps does not automatically mean the points are well dispersed, as can be seen in case 3 which shows a false negative. These false negatives can be detected in combination with other methods, however, such as Pearson's correlation coefficient[6].

Pairwise correlation of parameters

Finally, there is a pairwise correlation between parameters. By using the Pearson method [6], it is possible to detect the degree to which pairs of parameters are correlated. A high absolute correlation indicates that those parameters are not uniformly distributed in 2D space, but are more related to each other. Examples of how various distributions of samples and their Pearson's coefficient can be seen in Figure 4.4. As can be seen, there is

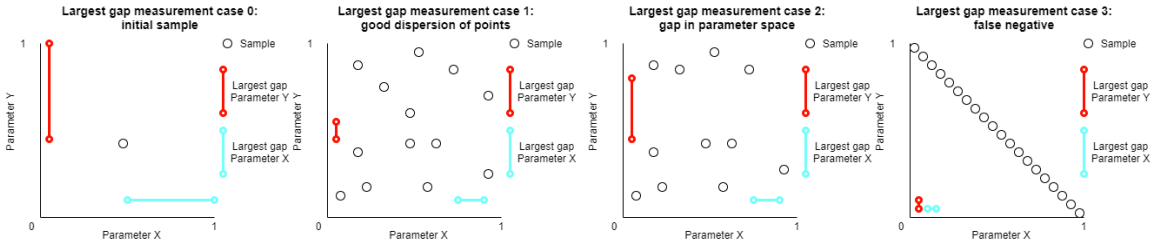


Figure 4.3: These 4 graphs show different dispersions of samples and numbers of samples in a two dimensional design space and how the gap measurement measures these dispersions.

the possibility of false negatives: if the Pearson coefficient is near zero that does not imply the samples are equally randomly distributed, however at the same time there are no false positives: if the Pearson coefficient nears 0, we can be sure that the points are not well distributed. This measurement is the only measurement that takes into account more than one dimension and gives a basic grasp of how the samples are distributed in two-dimensional space. The test is performed on all pairs, meaning in our case a total of 200 unique pairs. After exploitation we expect some pairs to become more related, as the DSE methods can find actual relations between parameters.

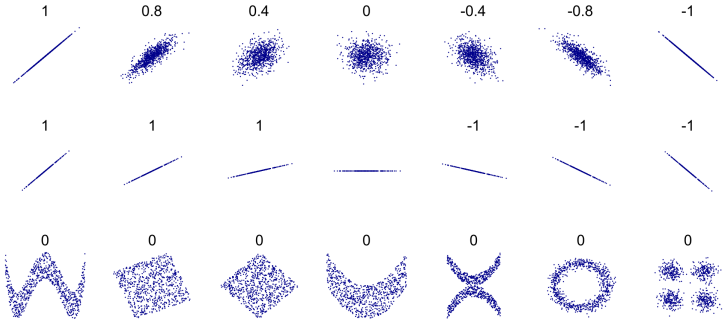


Figure 4.4: Various examples of distributed samples and their Pearson correlation coefficient [1]

4.2 Reference method

Next to quality measurement, it is also useful to have one or more control methods, as a reference. To ensure our methods are performing effectively.

4.2.1 Random Explorer

The random explorer is a DSE method that simply tests random designs. Its exploratory capabilities are near perfect: due to its uniform random sampling, the samples are very uniformly distributed, but due to the fact that it is random, it can never be completely uniform. This being said, it has no exploitation features: it simply picks random points and evaluates them. If this method can outperform any of our methods, given the same computational budget, it shows that our methods are performing worse than random exploration which means their exploitative capabilities are working against them, or that their

exploratory methods are not effective.

4.2.2 Additional meta-heuristics

To test the effectiveness of a new or already existing automated DSE method, it is very useful to have additional methods to test against. In our case, the setup allows for quick usage of alternative methods on the same problem which makes this process very easy. When another method is run on the same problem and using the same computational budget, we expect the same or worse results. If the results of the other method are on average better given the same computational budget, it implies that the used method gets stuck in a local optimum or might not be suited for the problem at hand.

Testing with additional meta-heuristics always results in either a better method or additional confidence in our findings. It is, however, recommended to try fundamentally different methods. For example, an alternative evolutionary algorithm to compare to a method which is also evolutionary would not expose any faults that the evolutionary part of the algorithm would cause.

Chapter 5

Analyzing and comparing DSE methods

In this chapter, the DSE methods are analyzed utilizing the results of our measurements described in Chapter 4.

5.1 Final results of all DSE methods

The final results of all tests in subsection 3.6.2 are as described in Table 5.1. The results for SA were very dissapointing, which was against expectations, combined with the fact that the SA method in the tool was performing very irregularly, it was decided to leave the results of SA out of the picture. The expectation is that the SA method provided by ECJ is not configured correctly and due to its long simulation time, it is hard to figure out what exactly is going wrong.

	SPEA2	PSO	Random	SPEA2 Extended	SA
Mean	133.68	140.93	312.51	105.33	1959.98
Median	128.28	134.00	265.01	102.24	639.26
Std	44.21	51.30	239.41	24.26	3582.16
Best	60.17	65.05	135.14	67.97	333.176
Failed runs	0	0	0	0	

Table 5.1: Statistics of the final results, the largest difference found between two pixels on the printing surface in J/m^2, of each run for each method; lower is better

As can be seen from the results, the SPEA2 Extended method has the best results on average and also the lowest standard deviation, indicating a consistently good performing method. From the other methods, SPEA2 and PSO have also achieved good results, with SPEA2 even having found the best design overall. Both these methods report higher averages and, more importantly, higher standard deviations, meaning they are less consistent in achieving their results.

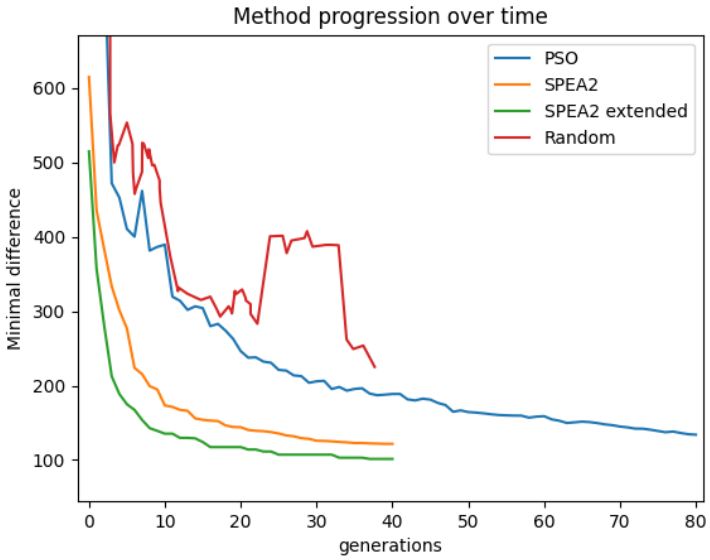


Figure 5.1: The median progression of the methods, recorded per generation

In Figure 5.1 we can see the progression of the methods over time, the results here show that both SPEA2 and SPEA2 Extended are quick to exploit and optimize designs, while PSO requires more generations to find designs with scores comparable to those found by SPEA2 (Extended), actually following the trend of the Random search. The Random search, in this case, was more difficult to portray accurately in terms of generations, as it does not use the concept of generations. Instead, the running average of all improving results of the 30 Random runs, sorted over time, has been portrayed, where a generation is equal to 80 samples. So between generation 25 (sample $25 \cdot 80 = 2000$) and 38 (sample $38 \cdot 80 = 3040$) the 10 Random methods that found an improvement, found on average a solution with a difference in dosage of 400 J/m^2.

5.2 Results sorted by time

To best portray the results of the DSE methods, the measurements were taken at moments of importance in time. The first measurement was taken after 5 generations, which are the generations in which the most improvement takes place according to Figure 5.1. The second measurement was taken at 15 generations, after which the improvements shift to a less steep linear progression. The third measurement is after 30 generations, which is when the improvements found by all methods have significantly slowed down.

5.2.1 First 5 generations

The first five generations is where the first exploration takes place, the methods search the solution space in search of good solutions and parameter combinations. This results in very fast improvement, seeing as the initial solutions were generated randomly and are not likely to be optimal. This fast improvement can be seen in Figure 5.1. For these generations, it is best to have as wide a coverage as possible. Our measurements are recorded in Figure 5.2,

which covers the maximum gap measurement, and Figure 5.4, which covers the Pearson's Correlation Coefficient between all parameter pairs.

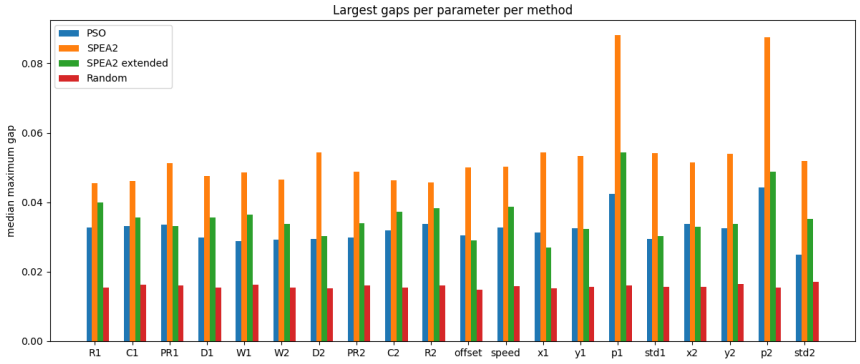


Figure 5.2: The median largest gap of 30 runs after 5 generations, per parameter

The gap measurement, from Section 4.1.3, shows the size of the largest gaps left in exploring the design space, per parameter. At this moment of the DSE process, uniform distribution of samples taken is very important, so the goal value for gapsize is as low as possible, which we expect the Random DSE method to achieve. In Figure 5.2 we see that SPEA2 leaves the largest gaps, followed by SPEA2 Extended. For many parameters, PSO closely follows behind SPEA2 Extended. The Random search, in this case, leaves the smallest gaps, which is as expected, but the Random method does not achieve the best results as shown in Figure 5.1. This implies that, for this problem, even in the first generations, there is already benefit to be had from exploitation. This most likely explains why PSO is behaving poorly in comparison to both SPEA2-based methods, given that the method needs some startup time due to time being an important factor in its exploration process.

An interesting observation here is that the median for both Power parameters p_1 and p_2, the parameters governing the intensity of the lamps (subsection 3.2.4), all non-random methods leave significantly larger gaps. When investigated further, the large gaps seem to have no specific preference for their location: no value between 0 and 1 for p_l has significantly more occurring number gaps than any other value. This can be seen from Figure 5.3 where only p_2 has a slight preference to have the largest gap at a higher value of p_2. There was also an analysis done where the size of the gaps was taken into account, but this also showed no clear pattern. This indicates that power is a very important factor during discovery; a lot of power values were not considered and gaps were left in the design space probably because the methods expect no good results there, or have found significantly better results in a specific place which it focuses all samples on.

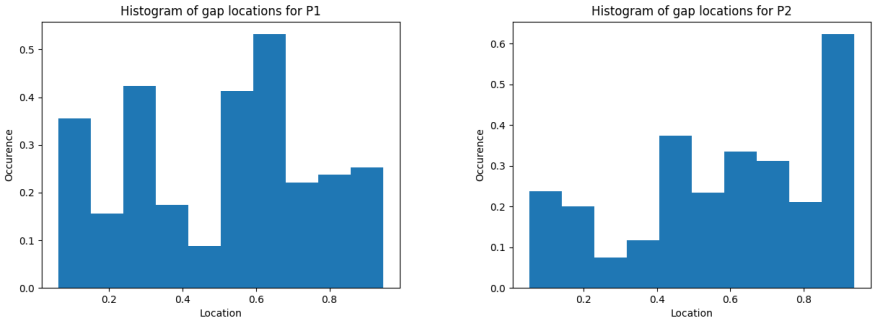
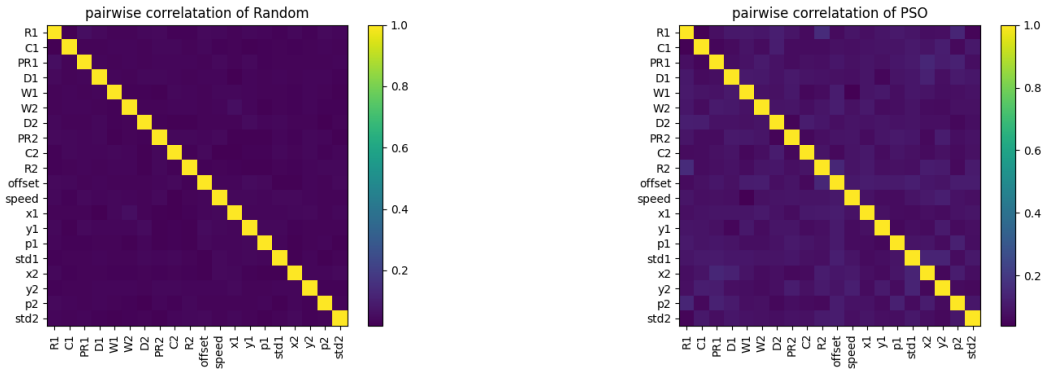
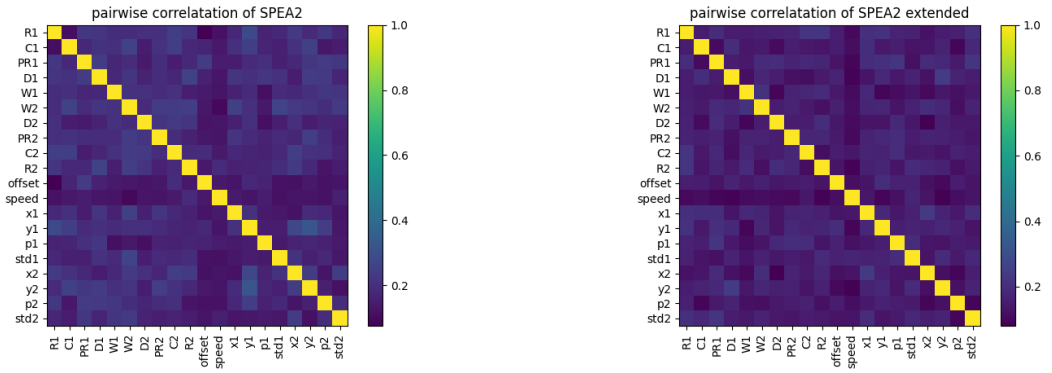


Figure 5.3: Occurrence histograms for location of gaps after 5 generations for SPEA2



(a) Random

(b) PSO



(c) SPEA2

(d) SPEA2 Extended

Figure 5.4: The Pearson Correlation Coefficient of all parameter pairs, for each method after 5 generations

The Pearson's Correlation Coefficients for Random show what is expected; almost no correlation. For PSO, the maximum correlation is less than 0.15, which is not statistically significant. SPEA2 has the most correlation on average, with the highest correlation values between Y_1 and Y_2 at 0.3. These parameters represent the width of both lamps, and this implies that they are correlated, or inversely correlated. This implies that the relation between the width of the lamps plays an important role for designs of interest to SPEA2.

SPEA2 Extended has less correlation overall with an interesting overall lower correlation for the speed parameter. This is very interesting as one would expect the speed parameter would be the most correlated due to its large influence on the curing process.

5.2.2 First 15 generations

After the first 5 generations, the improvements slow down to a steady pace, this is when the methods are exploiting the found results. This is the most unstable period, where all methods do not have steady progression but a more rough progression instead. Here most methods are locally exploring and finding a lot of different solutions with varying levels of success.

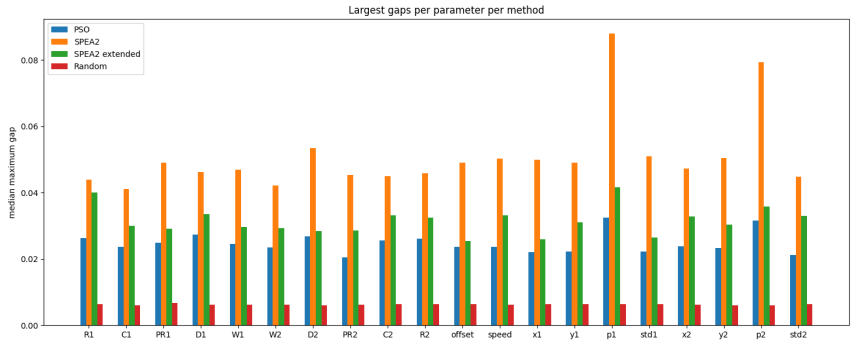
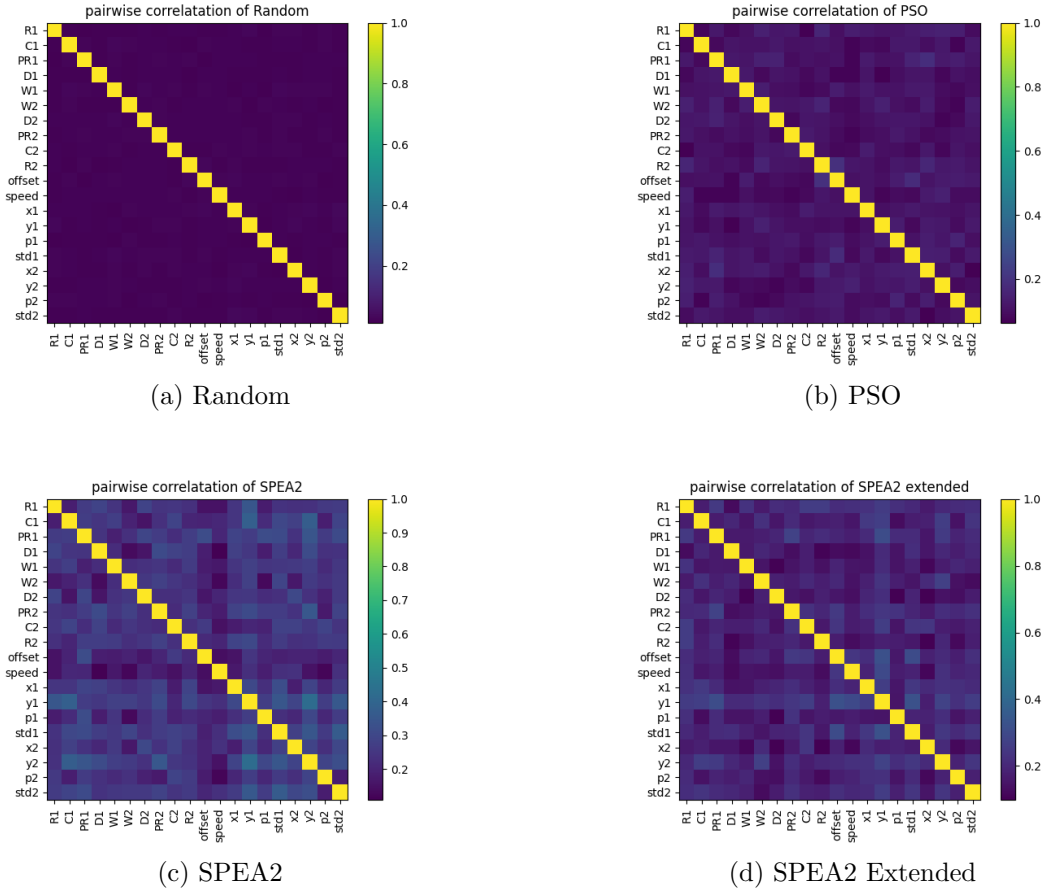


Figure 5.5: The median largest gap of 30 runs after 15 generations, per parameter

From the gap measure in Figure 5.5 we see that, apart from Random, the other methods have not significantly improved their coverage, meaning that the gaps which were left after 5 generations are not filled in the 10 generations afterward, which implies that all samples made for new designs must have happened in already existing smaller gaps, indicating exploitation and local search. For Random we know this is not the case, and as expected Random also shows significantly smaller gaps in the results compared to the previous measurement.



(a) Random

(b) PSO

(c) SPEA2

(d) SPEA2 Extended

Figure 5.6: The Pearson Correlation Coefficient of all parameter pairs, for each method after 15 generations

The Pearson Correlation Coefficients in Figure 5.6 show some interesting results. Random is as expected completely uncorrelated. PSO has a very low correlation in comparison with both SPEA2 methods, showing that it is searching more broadly: no 2 parameters show significant correlation which indicates that the sampling is still random, or at least not tied to parameter results. This effect could be due to PSO utilizing multiple agents at different places in the design space, which are initially randomly distributed and move somewhat slowly through the design space making their results still uncorrelated between all agents by default. SPEA2 again shows the highest correlation on average, with the highest values for correlation being between the lamp generation parameters, and the least correlation values for the offset and speed parameters. SPEA2 Extended has a lower correlation than SPEA2, without any significant spikes or absences of correlation. This implies SPEA2's samples are already more concentrated, indicating that some form of local search is taking place, which is as expected when looking at Figure 5.1.

5.2.3 First 30 generations

In Figure 5.1 after 15 generations, we see SPEA2 and SPEA2 Extended require increasingly more generations to significantly improve the design, while PSO gets a more steady but slower improvement than its previous generations. The uniqueness of solutions also stays

steady from generation 15 as can be seen in Figure 5.10.

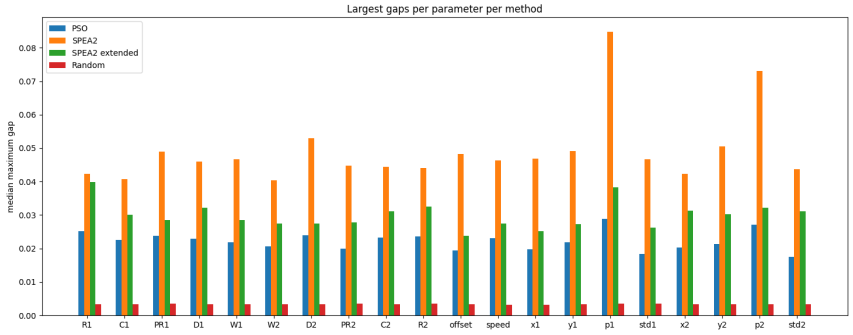
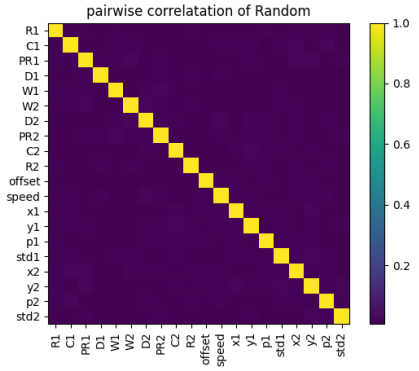
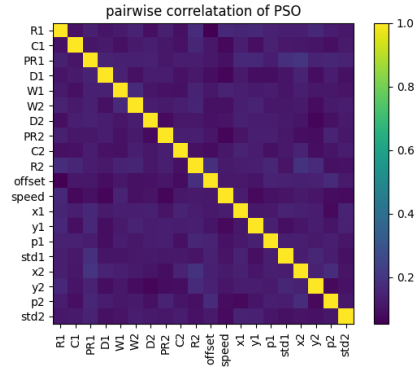


Figure 5.7: The median largest gap of 30 runs after 30 generations, per parameter

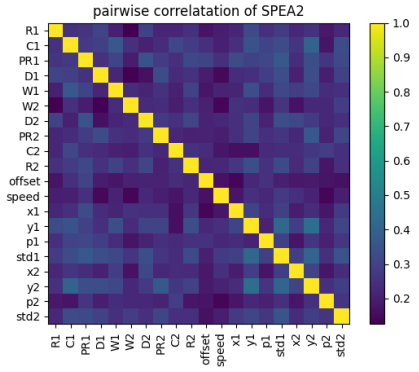
The gap measurement in Figure 5.7 shows no significant change anymore compared to the previous measurements, except for Random which is still filling gaps and effectively, but slowly, exploring the complete design space.



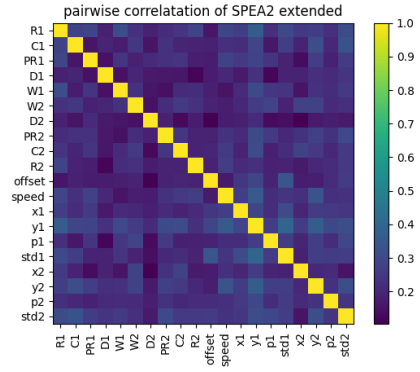
(a) Random



(b) PSO



(c) SPEA2



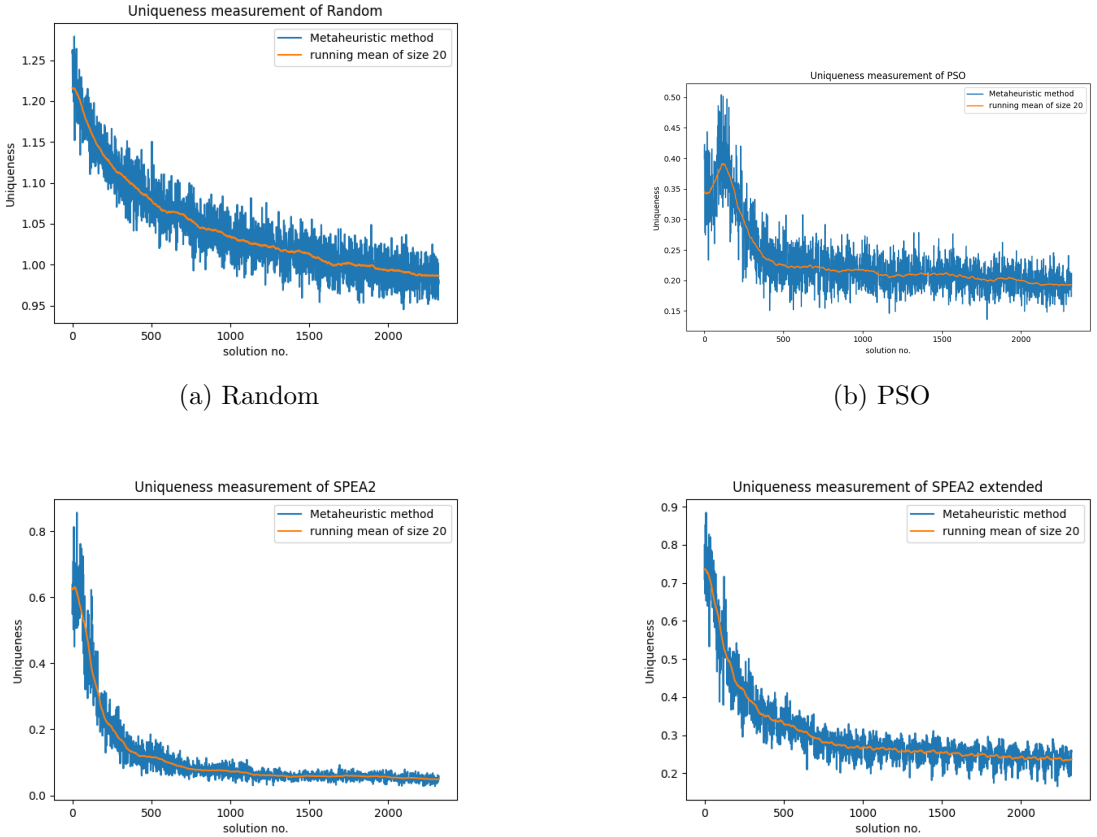
(d) SPEA2 Extended

Figure 5.8: The Pearson Correlation Coefficient of all parameter pairs, for each method after 30 generations

The Pearson Correlation Coefficient in Figure 5.8 now shows patterning for SPEA2: there seem to be certain trends in the data which results in a pattern showing up in the graph. For SPEA2 Extended there is both more patterning and higher correlation on average compared to previous results. The prevalent correlations are for SPEA2: the lower correlations in general with speed and offset, and increased correlation between the lamp parameters themselves, such as a correlation between Y_1 and Y_2, which indicates that the width of the lamps are linked. For SPEA2 Extended we see only a higher correlation between Y_1 and most other parameters, which indicates that Y_1 is the most important feature on which other parameters are adjusted to fit. PSO and Random remain unchanged when it comes to correlation.

5.2.4 Measurements for all generations

Next to "snapshots" at certain moments in time during the DSE process, the Uniqueness measurement and Inter Quartile range measurements are more fit to be analyzed over time as they can be condensed to or consist of a single value, instead of a collection of values for every timestep.



(a) Random

(b) PSO

(c) SPEA2

(d) SPEA2 Extended

Figure 5.9: The uniqueness measure over time, containing the first 30 generations

In Figure 5.9 we see the progression of the uniqueness measurement, from subsection 2.3.2, the measure of how "new" a point is compared to those already found, over

time for all methods individually, together with the trend line which represents the running mean of the 10 samples before and after the current sample, as uniqueness measurements themselves can vary wildly per sample. Random here indicates the scenario where, in principle, the uniqueness measurement is the average expectation, being completely random and therefore having the highest uniqueness. PSO starts with an increase in uniqueness at first, which is typical for the method where individual particles start randomly, and for the first few generations are moving randomly throughout the design space, after which significantly better solutions of interest are found, after which the particles start moving more predictably, resulting in the reduction in uniqueness. For SPEA2, the uniqueness very quickly drops off, while SPEA2 Extended has a less sharp decline.

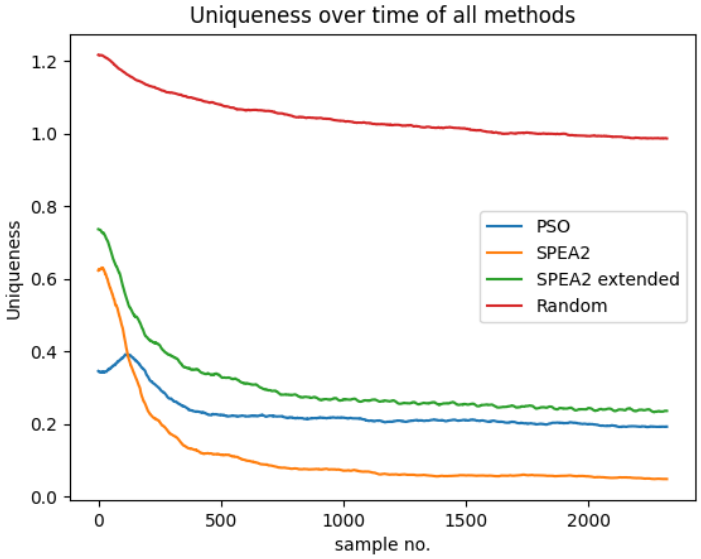


Figure 5.10: The average uniqueness measure over time for the first 30 generations, combined into 1 graph

When comparing all uniqueness progressions to each other in Figure 5.10, we observe some very interesting behavior. As expected the Random method has the most uniqueness, but we can see that PSO starts with low uniqueness and even after increasing slightly is still low, the expectation is that this is due to the speed of the particles of PSO being limited. What this means is that any new solution found by PSO is always in the close neighborhood of a previously found solution by that same particle. This makes it so that the average uniqueness is very consistent, especially after the particles exit their initial random distribution phase around sample 500, or generation 7. Here we also see the biggest difference between SPEA2 and SPEA2 Extended: the uniqueness of SPEA2 very quickly decreases to 0.1, while SPEA2 Extended, although also decreasing, flattens out at a higher uniqueness, meaning it keeps exploring more new and unique solutions.

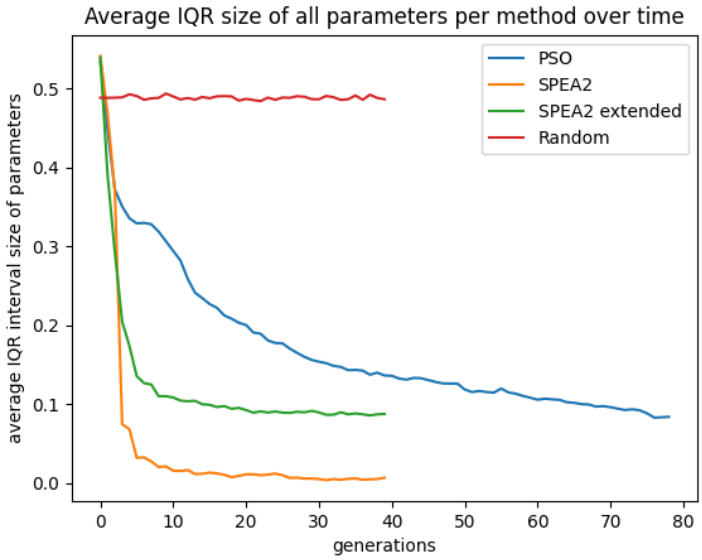


Figure 5.11: The width of the Inter Quartile range of all parameters, per generation of 80 samples

The Inter Quartile Range as seen in Figure 5.11 shows us the average interquartile range, from subsection 2.3.3, per generation of all parameters, a form of standard deviation. If the IQR range approaches 0.5, it means the points are more evenly spread out, and as it decreases to 0 it implies the samples are all closer together as the parameters are all found within a small range. At the start all points are random and as we can see, the resulting methods all start with an IQR of around 0.5. Over time we see similar results as found in Figure 5.10, however, the PSO measurements are decreasing steadily instead of flattening. Random is staying completely flat, as expected, as the IQR measure is taken per generation while the uniqueness measurement is based on all previous samples, meaning the uniqueness will slowly decrease over time as unique places in the space become filled by samples over time. The steady decline of PSO implies that even though its uniqueness stays fairly similar, on average the sampled parameters are being taken from a smaller space.

5.3 Analysis of the individual methods

5.3.1 Random

The Random method performs completely as expected; it samples the space randomly and has the overall worst performance. When it comes to exploration of the design space, it performs the best: the random sampling has lead to no gaps, no correlation, and the most uniqueness for each new sample. This also shows that the other methods are having an effect, because the 96000 random samples do not include better results than exploitative methods, implying that those methods are indeed effective for design space exploration.

5.3.2 SPEA2

The SPEA2 method achieves good results and converges to these results faster than other methods. After 6 generations, or 480 samples, new samples taken have low uniqueness and

the IQR is also very small. This means that all new samples are similar to one of the 480 samples already found. For a method such as SPEA2, this can lead to getting stuck in a local optimum, especially if in those first 480 samples no good set of parameter values is found. This is because new sample parameters are copied from older samples, with only a small chance of mutating into a new parameter. Still, for our problem, SPEA2 does find the best solution due to this fast convergence, and overall the results are not bad. For this design space, the SPEA2 algorithm approach works well.

5.3.3 PSO

PSO is a lot more exploratory, scoring high in all exploratory measurements except uniqueness, where it lags behind due to how PSO functions. Although this would work in very restrictive design spaces, where it is hard to find good solutions quickly, for our design space it implies that the method works very "slowly". It takes twice as long to reach the results as SPEA2 does. This methodology could be improved by changing the parameters based upon these results, as described in Section 7.2.2.

5.3.4 SPEA2 Extended

As mentioned in Section 3.5.2, SPEA2 Extended was created to be more exploratory. Instead of copy-pasting old parameters from two old solutions to new solutions, it now takes a solution that lies in between the two solutions, or in an extension of the line through the solutions. This results in a lot higher Uniqueness of samples, less correlation over time, and smaller gaps in the parameter space. As a result, we also see an increase in performance of the DSE method when it comes to final results, finding the best solutions on average. Not only are they the best on average, but it also reaches the same good results consistently as standard deviation is low.

In spite of those advantages, SPEA2 found a better solution than SPEA2 Extended. To further improve SPEA2 Extended, we could for example focus on the uniqueness of solutions decreasing more over time, as it now stays mostly stable at a rather high value indicating a very wide search space, even though we want to search more locally over time and exploit our good solutions instead of keeping exploring.

5.3.5 SA

Simulated Annealing severely under-performed compared to even the Random exploratory method. This is because only limited testing was possible as the implemented version of SA in ECJ is very limited feature wise and does not allow fine grain control such as using allowing multiple agents to explore the design space faster. This resulted in poor performance and data which would not contribute to any meaningful result, design wise or DSE method wise. For improvement of the method, extensions of the method are available as described in Section 7.2.2.

5.4 Concluding remarks on the measurements

The measurements taken have helped visualize how the methods sample the design space and give insight into how methods are behaving and can be improved. With only the final results, or progression of the methods over time, the exact problems or advantages of a

method are not clear, but given our measurements, we get insights into what is happening during the process of automated DSE, which can help the development of good methods. For example it can show what causes early convergence. Every DSE problem needs a different approach, but using these methods we can see what aspects of methods makes them more effective. For the problem of UV curing, fast convergence has a lot of benefits, as is shown by the results of PSO compared to both SPEA2-based methods. From the results of both SPEA2 methods we can see that SPEA2 in its initial form is too aggressive in exploiting the design space as the uniqueness of solutions is very low. By adding to the exploratory components of SPEA2, thus resulting in the SPEA2 Extended method, we can increase the uniqueness of solutions which results in better results on average.

Chapter 6

Results of the UV curing case

The DSE methods en evaluation methodology of this thesis have been applied to the case of optimizing the UV curing process. The detailed results of this case are outside the scope of this thesis.

Chapter 7

Conclusions

In this chapter, the main conclusions are presented and further research is discussed.

7.1 Main conclusions

7.1.1 DSE Setup

For the thesis, a setup was created for performing DSE to explore the UV curing process. The setup includes the ECJ tool, which contained various DSE methods and which was adapted to perform on problems and models written in python. A python model was written, to emulate the UV curing process which allows for different lamps and lamp path designs to be simulated using simple parameters.

7.1.2 Quality measurements

The quality measurements used and developed for this thesis have given insight into how DSE methods explore the design space. They allow for a comprehensive study on the effectiveness of the DSE methods and also give us the tools to improve the used methods and see the effect of our improvements, not only in the final result but also during the overall process. Using these measurements, we improved the SPEA2 method, by increasing its exploratory capabilities, resulting in the SPEA2 Extended method. This method improved the average results significantly compared to the original method and decreased the standard deviation of final results, indicating a more reliable method that depends less on random chance to get good results.

7.2 Further research

7.2.1 DSE quality measurements

The DSE quality measurements already give good insight into what occurs during the DSE process, however, some methods are currently more insightful than others. The measurements that cover a lot of parameters individually, such as Pearson and the gap measurement, are particularly hard to interpret properly. They contain a lot of data but are hard to analyze. In future work, it would be of interest to simplify these measures so their progression can be shown over time in a single graph, instead of requiring various snapshots as was done in Section 5.2.

7.2.2 Improvement of the methods

Below we list several options for improvements of the DSE methodology for our specific use case.

Adaptive parameters for SPEA2

Instead of being reliant on changing the DSE parameters by hand, it is possible to have reactive or adaptive parameter changes during the DSE process which can be linked to the quality measurements. For example if uniqueness becomes very low, while no better solutions are found for multiple generations, one could make it such that the method changes its own parameters governing exploration, such that more exploration takes place. In this way the exploration can be rejuvenated and the method would be able to escape a local optimum more easily. This type of method can include an automated stopping mechanism and run indefinitely until interesting solutions are found. Especially evolutionary algorithms would benefit from this adjustment due to their many parameters whose effect is easily understood.

Improvement of PSO

PSO as used for the thesis is under-performing both SPEA2 methods, but it could benefit from parameter adjustments. For example, the method is very slow in improving and exploiting the designs. To change this behavior, the particles could be given a faster speed, and the best-found local solution could be given higher importance in the changing of the direction of each particle. Another tactic would be lowering the number of particles which could allow for more exploitation given that the number of particles is still a multiple of the number of available threads. This would allow for more generations to be performed in quick succession which gives more advantage to PSO due to its reliance on number of generations.

Improvements of SA

Our version of SA was under-performing even the Random method. To improve SA, the multi-agent version of SA could be considered, which performs multiple SA searches at the same time, allowing the method to enjoy speedup from parallelization. Other adjustments which are more advanced relate to the use of different temperature curves. Currently a falloff of 95% each generation was used, but this is very simplistic and can often cause the method to become stuck on local optima as it has no means of escape after a certain temperature. Custom temperature curves can be developed and include actual raising of the temperature at intervals to allow the method to escape local optima.

7.2.3 Multi-objective optimization

Due to lack of time and domain knowledge, it was not possible to take all aspects of the design into account. Multi-objective optimization can take multiple objectives into account; SPEA2 Extended already includes the required functionality for multi-objective optimization. It should therefore be straightforward to extend the design space to include multiple objectives. The performance is not guaranteed however, as it was mainly tested and improved based on single-objective evaluation. The expectation is that for multi-objective optimization, there will be more a more varied population as different objectives

will most likely require different designs. This will inherently result in more varied offspring between parents in evolutionary algorithms which again should result in higher uniqueness of solutions. As a result, there is less need for increased exploratory capabilities as implemented in SPEA2 Extended.

Bibliography

[1] Denis Boigelot. *An example of the correlation of x and y for various distributions of (x,y) pairs*. Sept. 5, 2011. URL: https://commons.wikimedia.org/wiki/File:Correlation_examples2.svg (visited on 07/01/2021).

[2] Anne Brindle. "Genetic algorithms for function optimization". In: (1980).

[3] James M. Buchanan. "The relevance of Pareto optimality". In: *Journal of Conflict Resolution* 6.4 (1962), pp. 341–354. DOI: 10.1177/002200276200600405. eprint: https://doi.org/10.1177/002200276200600405. URL: https://doi.org/10.1177/002200276200600405.

[4] K. Deb et al. "Scalable multi-objective optimization test problems". In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*. Vol. 1. 2002, 825–830 vol.1. DOI: 10.1109/CEC.2002.1007032.

[5] Marco Laumanns Eckart Zitzler and Lothar Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. 2001.

[6] David Freedman, Robert Pisani, and Roger Purves. "Statistics (international student edition)". In: *Pisani, R. Purves, 4th edn. WW Norton & Company, New York* (2007).

[7] Mariano Frutos et al. "Comparison of Multiobjective Evolutionary Algorithms for Operations Scheduling under Machine Availability Constraints". In: *The Scientific World Journal* 2013 (Dec. 2013), pp. 1–9. DOI: 10.1155/2013/418396.

[8] Carmen M. González-Henríquez et al. "Innovative procedure for precise deposition of wrinkled hydrogel films using direct inkjet printing". In: *Materials & Design* 194 (2020), p. 108959. ISSN: 0264-1275. DOI: https://doi.org/10.1016/j.matdes.2020.108959. URL: https://www.sciencedirect.com/science/article/pii/S0264127520304937.

[9] Darrall Henderson, Sheldon Jacobson, and Alan Johnson. "The Theory and Practice of Simulated Annealing". In: Apr. 2006, pp. 287–319. DOI: 10.1007/0-306-48056-5_10.

[10] Todd Hester and Peter Stone. "Intrinsically motivated model learning for a developing curious agent". In: *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*. 2012, pp. 1–6. DOI: 10.1109/DevLrn.2012.6400802.

[11] F. Javidrad and M. Nazari. "A new hybrid particle swarm and simulated annealing stochastic optimization method". In: *Applied Soft Computing* 60 (2017), pp. 634–654. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2017.07.023. URL: https://www.sciencedirect.com/science/article/pii/S1568494617304349.

[12] Don Jones. "Handbook of Test Problems in Local and Global Optimization by C.A. Floudas, P.M. Pardalos et al. In Noncovex Optimization and Its Applications, volume 33, Kluwer Academic Publishers, 1999". In: *Journal of Global Optimization* 16 (Mar. 2000), pp. 299–300. DOI: 10.1023/A:1008328212973.

[13] J. Kennedy and R. Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by simulated annealing". In: *SCIENCE* 220.4598 (1983), pp. 671–680.

[15] Adam Kołacz and Przemysław Grzegorzewski. "Measures of dispersion for multidimensional data". In: *European Journal of Operational Research* 251.3 (2016), pp. 930–937. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2016.01.011. URL: https://www.sciencedirect.com/science/article/pii/S0377221716000187.

[16] Joel Lehman and Kenneth O. Stanley. "Abandoning Objectives: Evolution through the Search for Novelty Alone". In: *Evol. Comput.* 19.2 (June 2011), pp. 189–223. ISSN: 1063-6560. DOI: 10.1162/EVCO_a_00025. URL: https://doi.org/10.1162/EVCO_a_00025.

[17] Majdi M. Mafarja and Seyedali Mirjalili. "Hybrid Whale Optimization Algorithm with simulated annealing for feature selection". In: *Neurocomputing* 260 (2017), pp. 302–312. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.04.053. URL: https://www.sciencedirect.com/science/article/pii/S092523121730807X.

[18] Klaus Meffert. *Java Genetic Algorithms Package*. URL: https://sourceforge.net/projects/jgap/ (visited on 07/01/2021).

[19] Bernardo Morales-Castañeda et al. "An improved Simulated Annealing algorithm based on ancient metallurgy techniques". In: *Applied Soft Computing* 84 (2019), p. 105761. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2019.105761. URL: https://www.sciencedirect.com/science/article/pii/S1568494619305423.

[20] H. Mühlenbein, M. Schomisch, and J. Born. "The parallel genetic algorithm as function optimizer". In: *Parallel Comput.* 17 (1991), pp. 619–632.

[21] Heinz Mühlenbein and Dirk Schlierkamp-Voosen. "Predictive Models for the Breeder Genetic Algorithm i. Continuous Parameter Optimization". In: *Evol. Comput.* 1.1 (Mar. 1993), pp. 25–49. ISSN: 1063-6560. DOI: 10.1162/evco.1993.1.1.25. URL: https://doi.org/10.1162/evco.1993.1.1.25.

[22] Edgar Reehuis et al. "Novelty and Interestingness Measures for Design-Space Exploration". In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. GECCO '13. Amsterdam, The Netherlands: Association for Computing Machinery, 2013, pp. 1541–1548. ISBN: 9781450319638. DOI: 10.1145/2463372.2463557. URL: https://doi.org/10.1145/2463372.2463557.

[23] Juan Rodríguez-Hernández. "Wrinkled interfaces: Taking advantage of surface instabilities to pattern polymer surfaces". In: *Progress in Polymer Science* 42 (2015). Topical Issue on Polymer Physics, pp. 1–41. ISSN: 0079-6700. DOI: https://doi.org/10.1016/j.progpolymsci.2014.07.008. URL: https://www.sciencedirect.com/science/article/pii/S0079670014000835.

[24] Alejandro Rosete-Suárez, Alberto Ochoa-rodrguez, and Michèle Sebag. "Automatic Graph Drawing and Stochastic Hill Climbing". In: 2 (Aug. 1999).

[25] et all. Sean Luke Eric O. Scott. *Evolutionary Computation Java*. URL: https://cs.gmu.edu/~eclab/projects/ecj/ (visited on 07/01/2021).

[26] Jonas Simon and Andreas Langenscheidt. "Curing behavior of a UV-curable inkjet ink: Distinction between surface-cure and deep-cure performance". In: *Journal of Applied Polymer Science* 137.40 (2020), p. 49218.

[27] inc. The Mathworks. *Matlab Global Optimization Toolbox*. URL: https://www.mathworks.com/products/global-optimization.html (visited on 07/01/2021).

[28] Darrell Whitley et al. "Evaluating evolutionary algorithms". In: *Artificial Intelligence* 85.1 (1996), pp. 245–276. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(95)00124-7. URL: https://www.sciencedirect.com/science/article/pii/0004370295001247.

[29] Xin-She Yang. "Test Problems in Optimization". In: (Aug. 2010).

[30] Prof. Dr. Andreas Zell. *Evolutionary Algorithms 2*. URL: http://www.ra.cs.uni-tuebingen.de/software/EvA2/index.html (visited on 07/01/2021).

[31] E. Zitzler and L. Thiele. "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach". In: *IEEE Transactions on Evolutionary Computation* 3.4 (1999), pp. 257–271. DOI: 10.1109/4235.797969.

[32] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results". In: *Evolutionary Computation* 8.2 (2000), pp. 173–195. DOI: 10.1162/106365600568202.