Development of an Open-source Multi-objective Optimization Toolbox

Multidisciplinary design optimization of spiral staircases

Soheila AEENI

Academic supervisor: Camilla WEHLIN
Examiner: Johan PERSSON
Abstract

The industrial trend is currently to increase product customization, and at the same time decrease cost, manufacturing errors, and time to delivery. These are the main goal of the e-FACTORY project which is initiated at Linköping university to develop a digital framework that integrates digitization technologies to stay ahead of the competitors. e-FACTORY will help companies to obtain a more efficient and integrated product configuration and production planning process.

This thesis is a part of the e-FACTORY project with Weland AB which main mission is the optimization of spiral staircase towards multiple disciplines such as cost and comfortability. Today this is done manually and the iteration times are usually long. Automating this process could save a lot of time and money.

The thesis has two main goals, the first part is related to develop a generic multi-objective optimization toolbox which contains NSGA-II and it is able to solve different kinds of optimization problems and should be easy to use as much as possible. The MOO-toolbox is evaluated with different kinds of optimization problems and the results were compared with other toolboxes. The results seem confident and reliable for a generic toolbox. The second goal is to implement the optimization problem of the spiral staircase in the MOO-toolbox. The optimization results achieved in this thesis shows the benefits of optimization for this case and it can be extended by more variables to obtain impressive results.
Acknowledgements

This thesis was written by Soheila Aeeni at Linköping university and Weland AB during spring of 2019. The work has been challenging but at the same time educational and has given me a lot of valuable knowledge.

Firstly I would like to thank the director of this thesis, Mehdi Tarkian, for his feedback, guidance and support during the work in the e-Factory team. Your contribution during this thesis has been very helpful!

I would also like to thank my examiner, Johan Persson, and my supervisor Camilla Wehlin for their continuous feedback and helpful discussions. I must say that I am thankful for the pleasant supports from all involved persons and my teammates at e-Factory team in university.

Finally, I must express my very profound gratitude to my dear husband, the best friend and partner, Mahdi Morsali, for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without him. Thank you Mahdi!

Also, I would like to thank my dear parents and my sweet sister and brother where they were supporting and encouraging me during my years of study from long distance.
# Nomenclature

## Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LiU</td>
<td>Linköping University</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>MOGA</td>
<td>Multi-objective Genetic Algorithm</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>Fast Nondominated Sorting Genetic Algorithm</td>
</tr>
<tr>
<td>OpenMDAO</td>
<td>Open-source framework for efficient multidisciplinary optimization.</td>
</tr>
<tr>
<td>MOO</td>
<td>Multi-objective optimization</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>MDO</td>
<td>Multidisciplinary design optimization</td>
</tr>
<tr>
<td>ACO</td>
<td>Ant Colony Optimization</td>
</tr>
<tr>
<td>ABC</td>
<td>Artificial Bee Colony</td>
</tr>
<tr>
<td>HS</td>
<td>Harmony Search</td>
</tr>
<tr>
<td>GEM</td>
<td>the Grenade Explosion Method</td>
</tr>
<tr>
<td>VEGA</td>
<td>vector evaluated GA</td>
</tr>
<tr>
<td>NPGA</td>
<td>Niched Pareto Genetic Algorithm</td>
</tr>
<tr>
<td>WBGA</td>
<td>Weight-based Genetic Algorithm</td>
</tr>
<tr>
<td>SPEA2</td>
<td>Strength Pareto Evolutionary Algorithm</td>
</tr>
<tr>
<td>RWGA</td>
<td>Random Weighted Genetic Algorithm</td>
</tr>
<tr>
<td>PAES</td>
<td>Pareto-Archived Evolution Strategy</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>Fast Nondominated Sorting Genetic Algorithm</td>
</tr>
<tr>
<td>RDGA</td>
<td>Rank-Density Based Genetic Algorithm</td>
</tr>
<tr>
<td>DMOEA</td>
<td>Dynamic Multi-objective Evolutionary Algorithm</td>
</tr>
<tr>
<td>MEA</td>
<td>Multi-objective Evolutionary Algorithm</td>
</tr>
<tr>
<td>EC</td>
<td>Evolutionary Computation</td>
</tr>
<tr>
<td>DEAP</td>
<td>Distributed Evolutionary Algorithms in Python</td>
</tr>
<tr>
<td>EA</td>
<td>Evolutionary algorithm</td>
</tr>
<tr>
<td>Pyevolution</td>
<td>Pure Python Genetic Algorithms Framework</td>
</tr>
<tr>
<td>pySTEP</td>
<td>Python Strongly Typed gEnetic Programming</td>
</tr>
<tr>
<td>Inspyred</td>
<td>Bio-inspired Algorithms in Python</td>
</tr>
<tr>
<td>open BEAGLE</td>
<td>A Generic Evolutionary Computation Framework in C++</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>TOPSIS</td>
<td>Technique for Order of Preference by Similarity to Ideal Solution</td>
</tr>
<tr>
<td>EOQ</td>
<td>Economic order quantity</td>
</tr>
<tr>
<td>BOM</td>
<td>Bills of materials</td>
</tr>
</tbody>
</table>
# Contents

1 INTRODUCTION .................................................. 2
  1.1 Context ......................................................... 2
  1.2 Motivation ......................................................... 2
  1.3 The Company ..................................................... 3
  1.4 Project goals ..................................................... 4
  1.5 Research Questions .............................................. 4
  1.6 Deliverables ....................................................... 4
  1.7 Delimitations ....................................................... 5
  1.8 Thesis Outline ..................................................... 5

2 THEORY: MOO-Toolbox ............................................. 7
  2.1 Design Optimization ............................................. 7
  2.2 Genetic Algorithm ............................................... 8
  2.3 Multi-objective optimization(MOO) ............................. 9
  2.4 NSGA-II and MOGA ............................................... 11
    2.4.1 MOGA ......................................................... 11
    2.4.2 NSGA-II ....................................................... 13
  2.5 DEAP .......................................................... 15
  2.6 Multidisciplinary design optimization .......................... 17
  2.7 OpenMDAO ......................................................... 18

3 Method: MOO-Toolbox ............................................ 19
  3.1 Pre-study ......................................................... 20
    3.1.1 Literature study .............................................. 20
    3.1.2 Prior work .................................................... 20
    3.1.3 Different packages in Python ................................ 21
  3.2 Implement NSGA-II using DEAP .................................. 21
  3.3 Case studies ..................................................... 22
    3.3.1 Homogeneous Individuals ................................... 22
    3.3.2 Heterogeneous Individuals ................................ 23
    3.3.3 Constraint Handling ......................................... 25
    3.3.4 Bridging openMDAO and DEAP ............................... 26

4 RESULTS: MOO-Toolbox .......................................... 28
  4.1 The designed MOO-toolbox ....................................... 28
    4.1.1 How to use MOO-toolbox in more details .................... 29
  4.2 Evaluation of MOO-toolbox ...................................... 30
  4.3 Bridging MOO-toolbox and OpenMDAO ........................... 35

5 DISCUSSION: MOO-Toolbox ....................................... 37
  5.1 MOO-Toolbox ..................................................... 37
  5.2 Capabilities and Advantages ................................... 37
  5.3 Future works ..................................................... 38
6 THEORY:

Weland Optimization

6.1 Spiral Staircases
   6.1.1 Staircase Radius
   6.1.2 Step Rise and Depth
   6.1.3 Turning of Staircase
   6.1.4 Headroom
   6.1.5 Child Safety

6.2 Cost

6.3 SQL Database And Python

6.4 XML File

6.5 Ranking Pareto Front

7 METHODS:

Weland Optimization

7.1 Define Optimization Problem

7.2 Ergonomic Model
   7.2.1 Implementation in MOO-toolbox

7.3 Cost Model
   7.3.1 Reading SQLite in Python
   7.3.2 SQL Database
   7.3.3 How different tables are related
   7.3.4 Cost formulation
   7.3.5 Summary of Cost formulation

7.4 Reading XML file in Python

8 RESULTS:

Weland Optimization

8.1 Case1 (3 floor)
   8.1.1 Defining Optimization Problem
   8.1.2 Pareto front and solution selection

8.2 Case2 (7 floor)
   8.2.1 Defining Optimization Problem
   8.2.2 Pareto front and solution selection

9 DISCUSSION:

Weland Optimization

9.1 Problem Formulation

9.2 Optimal Results

9.3 Future Works

10 CONCLUSION

10.1 Research Question 1

10.2 Research Question 2

10.3 Research Question 3

10.4 Research Question 4

Appendices
A First appendix
PART I
INTRODUCTION
CHAPTER 1

1 INTRODUCTION

1.1 Context

This project constitutes a master thesis in the department of Machine Design at Linköping University, carried out in cooperation with the company Weland AB. Weland is a company located in Smålandsteden, Sweden. In fact, LIU conducts the research project e-FACTORY whose purpose is to investigate efficient automation of customized products and this thesis goal is developing efficient automation way from the user-customized product all the way down to production.

1.2 Motivation

The industrial trend is currently to increase product customization, and at the same time decrease cost, manufacturing errors, and time to delivery. These are however all conflicting goals. LiU has initiated the project e-FACTORY to develop a digital framework that integrates digitization technologies which can help companies within this project to stay ahead of the competitors. One main issue with today’s processes are the estimates the sales staff needs to make when it comes to both product and production development. The cost is too high to involve product and production engineers in each quote.

![Figure 1]
The scope of the project is to increase product and production development effectiveness for manufacturing companies by employing state-of-the-art automation technologies. e-FACTORY will help companies to obtain a more efficient and integrated product configuration and production planning process, starting already at the sales stage.

As shown in figure 2 in e-FACTORY project, four theses are related to each other which are Monitor API, sale configurator and optimization framework. Monitor API group extract SQL database from monitor API which includes different configurations of spiral staircases in Weland. So, the formulation of the cost model is based on this database. On the other hand, sale configurator group at first receive some data from the customer and deliver them as XML file to optimization framework. At last, the results of optimization will be converted to XML file again and this XML file can be used by sale configurator and CAD configurator group to visualize the staircase with optimized parameters. The following picture is some parts of dataflow in e-factory projects, the complete information is available in Appendix.1

![Figure 2: Dataflow between different theses (Appendix.1)](image)

### 1.3 The Company

The company, in this case, is Weland which is basically a family business founded in 1947, more than 300 employees, approx. 80,000 $m^2$ area and a business center in Småländsstenar. Weland operates in the manufacture and sale of iron and metal articles and is one of Sweden’s largest players in sheet metal processing. Weland is a leading manufacturer and supplier of spiral stairs, straight stairs, railings, handicap ramps, walkways, grating and entrances. Weland’s business concept is to quickly deliver products of high quality. Before guaranteeing a short delivery time, Weland has a large stock, both of raw material but also of standard components. Customizing, for example, a spiral staircase is often a time-consuming design process. The first draft of the construction work for a staircase often needs to be adjusted after a quotation has been sent out. This is because the customer may have changed some-
thing or specified non-specific information to the quotation. In this case, design automation would be able to save a lot of time when the lead time is to produce a quote to the customer would be shortened considerably and that it would be easier to make changes afterward. This project aims to help Weland to some extent automate the process by developing a configurator for their customer-specific spiral staircases.

1.4 Project goals

Before this thesis, a project is done to optimize spiral staircases. The objectives were ergonomy and cost. This thesis continues the previous work and it has two important goals. First, developing a multi-objective optimization toolbox which includes NSGAII or MOGA to solve the weland case. Besides, further developing of the optimization problem for spiral staircases. In fact, the main focus is cost optimization which should be formulated based on data from SQL database. Weland requirements:

- The optimization framework should be able to receive and send data from the sale configurator
- Should be possible (and pretty easy) for Weland to change optimization parameters, weighting, etc in the future.
- Minimize the time of the optimization run so that it can be used quickly in a sale process.

1.5 Research Questions

By developing, implementing and validating a multi-objective optimizer, the following research questions will be evaluated and answered:

1. How can the multi-objective optimization toolbox be generic and easy to use?
2. How can the optimization algorithm communicate with the sales configurator?
3. How can the SQL database be used together with optimization and a sales configurator to customize staircases?
4. How can the database be used for solving the optimization problem?

1.6 Deliverables

The project is completed when the following have been delivered:

1. Multi-objective optimization toolbox.
2. Concretization of the optimization problem (spiral staircases in Weland).
3. A solution to the optimization problem.
4. Optimization results for spiral staircases which are implemented in Python.
5. A full report documenting the methodology, work process, results, and future studies.

1.7 Delimitations

1. The optimization does not have to include all parts of the product such as railing of the staircase because the problem will be more complex and for this step, we concentrate on the steps of staircases.

2. Optimization algorithm should be implemented into Python and it would not be supported by any other software (e.g. MATLAB) to perform its operation.

1.8 Thesis Outline

This section gives an overview of how the thesis is structured. The thesis is divided into four parts, an introduction to the thesis, a description of the work done in the creating multi-objective optimization algorithm (MOO-toolbox), a description of the work done in the field of design optimization and finally a discussion of the research questions as well as what conclusions were made. In the figure, the outline is presented graphically. It should be noted that even though creating MOO-toolbox and optimization are parallel in the picture, the work was done serially. However, when reading the thesis either section can be read independently depending on the readers’ field of interest.

Figure 3: The structure of the thesis
PART II
GENERIC MOO TOOLBOX
CHAPTER 2

2 THEORY: MOO-Toolbox

In this chapter, the theory used to create the MOO-toolbox is presented. First, some basic theory about design optimization and what problems can be solved using different optimization algorithms will be given. After that, the theory of genetic algorithm will be presented. Moreover, the theory of multi-objective optimization problems and different methods which are used to solve these kinds of problems will be discussed. Then, MOGA and NSAGA-II will be presented in more details. At last, DEAP package will be presented which is used to implement GA in Python and also, some theory about MDO and OpenMDAO toolbox are presented.

2.1 Design Optimization

Design optimization is important in engineering design through decision making process which can help to make a product that can meet the human requirements. Design optimization includes certain goals (objective functions), a search space (feasible solutions) and a search process (optimization methods).[1]

The feasible solutions have all designs by all possible values of the design parameters (design variables). The optimization method navigates for the optimal design within all feasible solutions. Mechanical design in optimization process has specific objectives like strength, deflection, weight and cost regarding the requirements which can cause a stronger, cheaper or more environmentally friendly end product and improve the quality of the product even with traditional deterministic design optimization model.[2] However, design optimization can be a complicated objective function with a large number of design variables or there are some complex constraints which can have a conflict with each other. So the case gets harder to conclude about an optimum design.[3]

Analytic or numerical methods have good performance in many practical cases, they may fail in more complicated design situations because the objective function may have many local optima, but the desired result is the global optimum. There are different types of optimization algorithms which can be divided into two categories: local optimization algorithms and global optimization algorithms. Local optimization algorithms can solve design problems with a large number of design variables. These type of algorithms are the gradient-based method to find an optimum. But they are effective to find a local optimum and they are not able to find global ones and maybe different start points can help the algorithm to find better results.[4]
The global optimization algorithms are more efficient and effective optimization methods to handle mechanical design problems because they are able to find global optima in design space. There are two kinds of global optimization algorithms which are nature-inspired heuristic and deterministic methods but the first optimization methods are more effective than deterministic methods. Therefore, they are widely used. There are several nature-inspired optimization algorithms, such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), Harmony Search (HS), the Grenade Explosion Method (GEM), etc.[1][4]

Creating the model is the most important part of the optimization process. To define our goal in the optimization problem, one or several objective functions can be defined. For example, one objective function can be a weight equation that the design should be minimized in it. The design problem can also have constraints that limit the design space to fulfill it, for example, if the stress cannot be more than 300 Mpa. To optimize the problem some design variables should be declared. These variables will be evaluated during the optimization to obtain the best values. The optimization problem can be summarized into the equation 1

\[
\begin{align*}
\text{min} & \quad F(x) = f(DC_1(x), DC_2(x), \ldots, DC_m(x)) \\
\text{subject to} & \quad x_i^l \leq x_i \leq x_i^u \quad i = 1, 2, 3, \ldots \\
& \quad x = (x_1, x_2, \ldots, x_n)^T \\
& \quad g_j(x) \leq 0 \quad j = 1, 2, 3, \ldots 
\end{align*}
\]

Where DC stands for design characteristics.[5]

2.2 Genetic Algorithm

The Genetic Algorithm (GA) was developed in the middle of the 1970s by John Holland and his colleagues and students at the University of Michigan. The GA imitates the principles of genetics and evolution. In other words, it imitates reproduction behavior in biological populations. The GA performs the principal of survival of the fittest. In its process, GA selects and generates individuals that form a population together and they are adapted to their environment which is design objectives and constraints.[6]

In GA, the parameters of the search space are represented in the form of chromosomes. In this way, the individuals can be compared with each other in fitness function in which the individuals have a fitness score and they will be selected for reproduction according to their fitness score. Initially, a random population is created, which is spread in the search space and the fittest individuals are selected for reproduction to produce offspring of the next generation.[7] After the selection of the first population, reproduction phase starts. GA uses two important operators to generate new individuals: crossover and mutation.
The crossover is more important because two chromosomes as parents mate with each other and generate a new child or offspring. The crossover will be done iteratively and good chromosomes will be repeated more in population and can lead to convergence to the best solution. Moreover, the crossover has a specific probability which helps the algorithm to create children who are not similar to the parents. Mutation does a random change in chromosomes and its probability of changing the characteristics of a gene is small so, the new genes are not very different from the old ones. But it has an important role because it can protect the population to have enough diversity and helps the GA to escape from local optima.

After reproduction, again selection procedure filters the individuals and determines the probability of their survival in the next generation. There are different types of selection methods in GA like a roulette wheel, tournament, ranking and proportional selection which are the most famous ones. Finally, the algorithm continues to iterate in order to find a desirable design or the population converges to an optimal solution. The loop of a genetic algorithm is shown in figure 4.7

The GA is suitable to solve complex design optimization problems because it can solve both discrete and continuous variables, and also it can handle nonlinear objectives and constrained functions without a need to gradient information. 6

\[\text{Figure 4: The loop of genetic algorithm}\]

### 2.3 Multi-objective optimization (MOO)

The world design, in reality, involves the optimization of multiple objectives at the same time. In fact, multi-objective optimization is completely different than single objective optimization. In a single objective problem, there is one global optimum (maximum or minimum) depending on the problem that we want to minimize or maximize it. But in multi-objective optimization problem, there is not only one solution regarding all objectives. In these kinds of problems, we have a set of so-
olutions which are better than the others. Each solution can satisfy all objectives more or less without being dominated by the other ones. These solutions are known as pareto optimal solutions. Moreover, because the solutions in pareto set are not better than the other, so all of them are acceptable solutions and designer decides about to choose which point regarding problem knowledge and environments.\textsuperscript{10}

To solve these kinds of problems, GA is a suitable solution because it can find a set of non-dominated solutions in a single run. GA is able to search for a different region in solution space at the same time. So, it can find a various set of solutions for complicated problems with discontinuous and non-convex solutions space. In addition, in multi-objective GA methods, it is not necessary to scale or weigh objectives. Therefore, GA is the best heuristic method to solve multi-objective problems.\textsuperscript{11}

The results of some research show 90 percent of methods to solve the multi-objective problems concentrate on true pareto front for fundamental problems and most of them used a heuristic technique like GA.\textsuperscript{12}

There are different types of multi-objective GA such as:

1. Vector evaluated GA (VEGA).
4. Weight-based Genetic Algorithm (WBGA).
5. Random Weighted Genetic Algorithm (RWGA).
7. Strength Pareto Evolutionary Algorithm (SPEA) and improved one (SPEA2).
10. Multi-objective Evolutionary Algorithm (MEA).
11. Rank-Density Based Genetic Algorithm (RDGA).
12. Dynamic Multi-objective Evolutionary Algorithm (DMOEA).

In general, multi-objective GAs are different regarding their fitness assignment procedure, elitism, or diversification methods. Moreover, there are two important duties that a multi-objective GA should be able to do well:

- Guiding the search region to obtain global Pareto-optimal.
- Protecting population diversity in the current non-dominated front.

In the following section, there is information about MOGA and NSGA-II in details, because the mission of this thesis is implementing one of these methods in Python and solve the problem with one of them.\textsuperscript{9} \textsuperscript{11}
2.4 NSGA-II and MOGA

2.4.1 MOGA

MOGA was the first multi-objective GA which integrated pareto-ranking and niching approaches together to help the search to reach the true pareto solution and maintain population diversity in the current non-dominated front.

Pareto-ranking approaches

Pareto-ranking approaches evaluate the fitness or give a probability to solution for selecting with pareto dominance approach. In fact, the population is ranked based on a dominance rule and each solution has a fitness value based on its rank in the population if the objectives should be minimized a better solution has a lower rank.

Fitness sharing and niching

Fitness sharing helps the search in sections of pareto front that remains unexplored. In fact, it can reduce the fitness of solution artificially in areas with dense population. To reach this goal, the areas with a large population are identified and a penalty method is able to penalize the solutions of these areas. The idea of fitness sharing was introduced by Goldberg and Richardson\[13\] in the research about multiple local optima for multi-modal functions. Fonseca and Fleming \[14\] proposed this idea to penalize solutions with the same rank which is described in the following:

Step 1: Euclidean distance between every solution pair \( x \) and \( y \) is calculated in the normalized objective space between 0 and 1 as equation 2

\[
dz(x,y) = \sqrt{\sum_{K=1}^{K} (\frac{z_k(x) - z_k(y)}{z_k^{max} - z_k^{min}})^2}
\]

where \( z_k^{max} \) and \( z_k^{min} \) are the max and min of objective function and \( z_k(.) \) is the values which are observed so far during the search.

Step 2: Niche count is calculated according to these distances as equation 3

\[
nc(x,t) = \sum_{r(y,t) = r(x,t)} \max\left(\frac{\sigma_{share} - dz(x,y)}{\sigma_{share}}, 0\right)
\]

where \( \sigma_{share} \) is niche size.
**Step 3:** Then, the fitness of each solution is computed as equation 4

\[ f'(x, t) = \frac{f(x, t)}{nc(x, t)} \]  

(4)

In equation 3, \( \sigma_{\text{share}} \) is a neighborhood of solution in objective space. so, the solutions in the same neighborhood can affect to their niche count. As a result, a solution in a crowded area has a higher niche count. Then, the probability of selection of that solution will decrease as a parent. Therefore, niching can help the algorithm to not select a large number of solutions in one specific neighborhood of objective function space.

Fitness sharing has two main disadvantages:

- The user should select a new parameter as \( \sigma_{\text{share}} \), but to solve this problem, a solution is developed which is able to estimate and dynamically update \( \sigma_{\text{share}} \).
- Extra computational effort for computing the niche counts lead to more cost but the benefits of this method usually can compensate this cost. [9]

**MOGA Procedure:**

**Step 1:** Initialize with random population \( P_0 \) and Set \( t = 0 \)

**Step 2:** When stop criterion is satisfied return \( P_t \).

**Step 3:** Fitness of the population will be evaluated through the following step:

Step 3.1: Each solution in \( p_t \) will be assigned by the rank \( r(x, t) \).

Step 3.2: Fitness values will assign to each solution regarding the rank of solution as equation 5

\[ f(x, t) = N - \sum_{K=1}^{r(x, t)-1} (n_k) - 0.5 \times (n(x, t) - 1) \]  

(5)

where \( n_k \) is the number of the solutions which rank is \( k \).

Step 3.3: The niche count \( nc(x, t) \) of each solution which belongs to \( p_t \) will be calculated.

Step 3.4: The shared fitness value of each solution will be calculated as we did in equation 4.

Step 3.5: The fitness values will normalize by shared fitness values as equation 6.

\[ f''(x, t) = \frac{f'(x, t) \times r(x, t)}{\sum_{r(y, t)=r(x, t)} f'(x, t)} \times f(x, t) \]  

(6)
Step 4: In this step, stochastic selection method will be used and according to $f''$, parents for the mating pool will be selected. Then crossover and mutation will apply on the mating pool until the offspring population fill the $Q_t$ with the size of $N$. Set $P_{t+1} = Q_t$.

Step 5: Set $t = t + 1$, go to Step 2.\[15\]

2.4.2 NSGA-II

The Non-dominated Sorting Genetic Algorithm (NSGA) introduced by Srinivas and Deb\[11\] was among the first evolutionary algorithms, but it had some main problems such as:

- Non-dominated sorting had high computational complexity.
- Lack of elitism.
- Need for specifying the sharing parameter $\sigma_{\text{share}}$.

Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II):

In NSGA-II method the problems of the first NSGA is solved and the number of different modules which composed NSGA-II are presented in the following:

A fast non-dominated sorting approach:

If the population size is $N$, and there is $m$ objective, each solution should be compared with other solutions in the population to find it is dominated or not. According to $m$ and $N$, there should be $O(mN)$ comparisons for each solution. The total complexity is $O(mN^2)$ when the process of comparison continues to find the first nondominated class for all members in the population. In this level, the first non-dominated front is found and for finding the other fronts, the solutions in the first front are ignored momentarily and the above process is repeated. In the worst case, the algorithm complexity will be $O(mN^3)$ but in most cases, it is $O(mN^2)$. Then it is better than NSGA in which the complexity was $O(mN^3)$ for most cases and it was really expensive for a large population. The fast non-dominated sorting procedure, when performed on a population $P$ and returns a list of the non-dominated fronts $F$.

Crowding distance

Crowding distance methods aim to achieve a uniform spread of solution in the best pareto front without the need to fitness sharing parameter. To compute the crowding distance, the average distance of the two points on either side of the specific point will be considered along with every objective. $i_{\text{distance}}$ is the size of the largest cuboid enclosing the point $i$ while it does not have any other point in it. In fact, this
distance is called crowding distance. Figure 5 shows the crowding distance of point i. In other words, crowding distance is the average side-length of the cuboid (dashed box). The main advantage of the crowding distance method is it can be computed without the need to define a parameter like $\sigma_{\text{share}}$. The important thing to know is about the crowded tournament selection operator in which two solutions will be selected randomly. Then if they belong to same non-dominated front, the solution with higher crowding distance will be selected. Otherwise, the solution with the lower rank is the winner.[16]

**NSGA-II Procedure:**

**Step 1:** Initialize with random population $P_0$ and Set $t = 0$.

**Step 2:** Perform crossover and mutation in $P_0$ to create offspring population $Q_0$ with size N.

**Step 3:** When stop criterion is satisfied return $P_t$.

**Step 4:** Set $R_t = P_t \cup Q_t$.

**Step 5:** Apply the fast non-dominated sorting algorithm and find the non-dominated fronts $F_1$ to $F_k$ in $R_t$.

**Step 6:** For $i = 1$ to $k$, the following steps will be done:
Step 6.1: Calculate crowding distance.
Step 6.2: Create $P_{t+1}$.

**Step 7:** Apply binary tournament selection regarding the crowding distance to select parents from $P_{t+1}$ and use crossover and mutation to $P_{t+1}$ for creating offspring population $Q_{t+1}$ in size N.
Step 8: Set $t = t + 1$, and go to Step 3.

It is important to note to this issue about NSGA-II that combined parent and offspring population includes more $N$ non-dominated solutions, then NSGA-II gets as a pure elitist GA in which only nondominated solutions participate in crossover and selection. The main benefit of maintaining non-dominated solutions in the population is implementation will be simple. In this part as conclusion, Table 5 shows the specification of MOGA and NSGA-II with their advantages and disadvantages and it can be a suitable conclusion.

<table>
<thead>
<tr>
<th>Specification</th>
<th>MOGA</th>
<th>NSGA-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness assignment</td>
<td>Pareto ranking</td>
<td>Non-domination sorting ranking</td>
</tr>
<tr>
<td>Diversity mechanism</td>
<td>Fitness sharing by niching</td>
<td>Crowding distance</td>
</tr>
<tr>
<td>Elitism</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Advantages</td>
<td>Simple extension of single objective GA</td>
<td>Single parameter(N) well tested, Efficient</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>Usualy slow convergence Problems related to niche size parameter</td>
<td>Crowding distance works in objective space only</td>
</tr>
</tbody>
</table>

2.5 DEAP

Evolutionary Computation (EC) is an enlightened field with various techniques and mechanisms, where even well-designed frameworks can become quite complex. The DEAP (Distributed Evolutionary Algorithms in Python) framework is built in the Python programming language that supplies the necessary glue for assembling sophisticated EC systems. Its goal is to deliver practical tools for quick prototyping of custom evolutionary algorithms which has explicit steps through the process and easy to read and understand.

DEAP core has two simple structures: a creator and a toolbox. The creator is a meta-factory that allows creating classes which can help us to reach the goals of your evolutionary algorithm. The classes can be composed of any kinds of the type such as list, set, dictionary. Moreover, it makes possible to implement genetic algorithms, genetic programming, evolution strategies, particle swarm optimizer and other things. The toolbox is a container for the tools which the user wants to use for Evolutionary algorithm(EA). The toolbox is manually set by the user when he selects the tools. For example, the user needs a crossover in the algorithm, but
there are several crossover types, he will choose the best one which is suitable for his specific case. The tools module also helps the user to define basic operators such as initialization, mutations, crossovers, and selections. This module also includes some components that collect useful information for evolution such as fitness statistics, genealogy, hall-of-fame for the best individuals.

To illustrate how DEAP works and can be used, the example of multi-objective feature selection which is available in DEAP documentation will be presented in following and the following code shows the solution. The individual type is a bit-string where each bit matches a feature that can be selected or not. There are two objectives and one objective is to maximize the number of well-classified test cases and the other one is to minimize the number of features used. On line 2, the relevant DEAP modules are imported. Then, in ”evalFitness” function the objectives are defined.

```python
import knn, random
from deap import algorithms, base, creator, tools

def evalFitness(individual):
    return knn.classification_rate(features=individual), sum(individual)

creator.create("FitnessMulti", base.Fitness, weights=(1.0, -1.0))
creator.create("Individual", list, fitness=creator.FitnessMulti)

toolbox = base.Toolbox()
toolbox.register("bit", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.bit, n=13)
toolbox.register("population", tools.initRepeat, list, toolbox.individual, n=100)
toolbox.register("evaluate", evalFitness)
toolbox.register("mate", tools.cxUniform, indpb=0.1)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selNSGA2)

population = toolbox.population()
fits = toolbox.map(toolbox.evaluate, population)
for fit, ind in zip(fits, population):
    ind.fitness.values = fit

for gen in range(50):
    offspring = algorithms.varOr(population, toolbox, lambda_=100, cxpb=0.5, mutpb=0.1)
    fits = toolbox.map(toolbox.evaluate, offspring)
    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit
    population = toolbox.select(offspring + population, k=100)
```

The first argument of the ”creator.create” method states the name of the derived class, the second argument defines the received base class. The third argument adds a new class attribute called weights and it is defined in a tuple and (1.0) shows the objective should be maximized and (-1.0) shows the objective should be minimized. Then, an Individual class is in the type of Python list and is built by created ”FitnessMulti” object. After this step toolbox object is created on line 9, on lines 10 to 16 will be and populated with aliases( The name the operator will take in the toolbox.) to initialize individuals and population, and identify the variation opera-
tors (mate, mutate, and select) and evolutionary loop will use the fitness evaluation function (evaluate). The toolbox register method receives different types of arguments. First one is the name of a function in the toolbox under the name alias and the second is the function that we want to associate with this alias. When the alias is called, the other arguments are passed to this function. For example, when we call ”toolbox.bit”, in fact, we call ”random.randint” with value of 0 and 1.

On line 11, initialization of individuals will be done by assuming 13 features selection problem, this bit function is called 13 times with the help of ”tools.initRepeat” method which accepts three arguments.

A container and a function and the number of times that we want to repeat initialization. Similar to the previous step, population initialization will be done with \( n = 100 \) individuals. Fitness evaluation is done on line 19 by mapping the evaluation function to elements that are available in the population container. Lines 20 and 21 replace the individuals’ fitness with their new ones. Finally, lines 23 to 28 show the evolutionary loop and it uses the ”\( \mu + \lambda \)" strategy, in this strategy \( \mu \) as parents (current population) will be mixed with \( \lambda \) as offspring (lambda line 24) for the selection process (NSGA-II) in order to generate the next generation of \( k = 100 \) parents.

The ”varOr” algorithm loops use either crossover (mate) with probability \( cxpb \), mutation (mutate) with probability \( mutpb \), or reproduction until produce \( \lambda \) offspring. This variation is named Or because an offspring is not the result from both operations crossover and mutation. The sum of both probabilities would be in \([0, 1]\), and the reproduction probability is \( 1 - cxpb - mutpb \).

In figure 6, all operators which are implemented in DEAP is available.

\[2.6 \textbf{Multidisciplinary design optimization}\]

The traditional process of design optimization can get slow and time consuming. To analyze a complicated system, considering several different disciplines can give a
better view of the whole system. Also, if all disciplines are optimized simultaneously, it gives a better and balanced optimum for the whole system. While the disciplines are optimized separately it can give a sub-optimal design. [23]. Multidisciplinary design optimization (MDO) is an approach can be divided into three main steps: First, the problem should be broken down into different disciplines. Because it can make easier to analyze the smaller sub-element. Then it should be defined how different variables are related to these disciplines. There are different types of variables such as local variables which are affecting a specific discipline, shared variables that are sent to more than one discipline. And coupling variables which are outputs of some disciplines and they are sent as inputs to another discipline. There can be also some design constraints and there are design objectives for the different disciplines. And third, is the optimization of the design problem, which is a sub-level optimization for different disciplines and is a system-level optimization for conflicts of the disciplines. [24]

2.7 OpenMDAO

OpenMDAO is an open high-performance software used for multidisciplinary optimization but also analyzes of different systems. It uses the programming language Python. The primary focus of OpenMDAO is to solve gradient-based problems in order to be able to evaluate large design spaces with a large number of parameters, but it is also possible to solve problems with other solution strategies such as traditional exploration of the design space. [25] In OpenMDAO, variables are calculated by defining them in different components. Each component has input and through calculations, each component will generate output. There are three different general components that are independent components, explicit components and implicit components depending on whether the different components require input data calculated in another component or not. [26] When the components are defined, they are linked together in a common framework where the flow of data is defined between the various components. Then some sort of solver is chosen, it can be an optimization algorithm or some explorer solver. Finally, the work-flow is defined before the problem can finally be solved. One of the ideas with OpenMDAO is that the work-flow is separated by the data flow. [27]

There are several advantages of OpenMDAO, and there are also ready-made libraries with different solvers and optimization algorithms. There are tools to use for meta-modeling and support for analytical derivatives. OpenMDAO has good opportunities to document generated data. Finally, it supports high-performance computer clusters and distributed computer processing and there is an extensive plug-in library. This, together with being an open-source program that can be used in Python platforms makes it a powerful and flexible tool that can be adapted for specific purposes. One disadvantage of openMDAO is that it is difficult to visualize results if it is compared with Modefrontier where it is easy to use lots of different visualization options. [27]
CHAPTER 3

3 Method: MOO-Toolbox

This chapter describes the work-flow and methodology used to complete multi-objective optimization toolbox. The work was structured in three steps. The first was pre-study or data collection and the second step was implement MOGA OR NSGA-II and the third step was specifying some case studies which can help to find out which condition is necessary to consider for a generic toolbox. The pre-study was done in three different ways; literature study, study of prior work and investigation about different ways of implementing GA in python. The second step was implementing NSGA-II or MOGA using DEAP and the third step consist of different case studies such as methods of solving the problem with homogeneous individuals, heterogeneous individuals, constraint handling and bridging OpenMDAO and DEAP. The output from the used methodology was a generic toolbox which can handle different kinds of MOO problems. Thesis methodology is presented in figure 7.

![Figure 7: Thesis Methodology](image)
3.1 Pre-study

3.1.1 Literature study

The literature study performed in the project was mainly based on different articles. At first, more study about the genetic algorithm in detail is done. Then some investigation about different methods that can help to solve multi-objective problems is done as explained in the theoretical background. Investigation about MOGA and NSGA-II and their differences.

3.1.2 Prior work

Earlier work has been done for Weland to do design optimization of spiral stair-cases. To solve the problem, they had two approaches. They solved the problem by using Modefrontier and the second method was to solve the problem with OpenMDAO. But there were some problems and deficiencies regarding the previous approaches that present in following briefly:

Modefrontier

- Optimization will be slow when calculations must be done in other programs such as Excel and Matlab.
- The program crashes sometimes.
- In the case of Weland problem, there were some specific needs for solving the problem such as output values had to be saved in vector shape and that was difficult to solve it with Modefrontier.
- There were some constraints and inputs that had to be entered manually and handling the situation was difficult.

OpenMDAO

OpenMDAO was better than Modefrontier because it is an open source software, it was easy to handle multi-dimensional problems, was easy to get output in vector shape and it was faster but it had also some shortcoming:

- More difficult to visualize how parameters and nodes are connected.
- Some limitation regarding multi-objective optimizer because it is necessary that users specifies the weight of objectives themselves to set the pareto front and make it hard for an algorithm to explore the whole design space.\[28\]

Therefore, it has been decided to create an open source MOO-toolbox which contains NSGA-II or MOGA to handle multi-objective problems.
3.1.3 Different packages in Python

In this step, there was some investigation about different packages in Python which can help to implement a genetic algorithm. There are different packages such as:

**DEAP** is Distributed Evolutionary Algorithms which has very good documentation and complete operators such as crossover and mutation.

**Pyvolution** [29] is a Pure Python Genetic Algorithms Framework which has good documentation but its operators such as crossover and mutation was not as complete as DEAP.

**pySTEP** is Python Strongly Typed gEnetic Programming [30] which has little documentation and operators.

**Pyevolve** is developed to be a complete genetic algorithm framework written in pure Python [31] but it has little documentation and operators in comparison with DEAP.

**inspyred** and **open BEAGLE** are also popular framework but DEAP developers claim DEAP is the only framework that is able to do a complete definition of the specific example in less than one hundred lines of code. For example, pyevolve needs 378 lines, inspyred needs 190 lines and open BEAGLE needs 477 lines while DEAP only needs 59 lines. Also some algorithm, force user to go through the framework in depth and change a hundred lines of codes. But DEAP is better than previous frameworks for fast prototyping of new algorithms and definition of custom types. [17]

3.2 Implement NSGA-II using DEAP

To implement NSGA-II in Python, DEAP has really suitable facilities that make possible to implement NSGA-II. In fact, it has necessary tools like crossover methods, mutation methods and selection. Therefore, NSGA-II is selected to be used for the MOO-toolbox. As we discussed in the theory chapter, NSGA-II uses non-dominated sorting and crowding distance to rank the solution and then uses tournament selection to select the individual for applying the crossover and mutation. DEAP has an operator which is called "selNSGA2" in which just assigning the crowding distance to the individuals will be done, not the actual selection and operator "selTournamentDCD" will be done based on dominance (D) between two individuals. If the two individuals do not dominate each other the selection is made based on crowding distance (CD). Each individual in the list of input won’t be selected more than twice. [18] The other parts related to initializing the individuals dependent on the problems and types of variables which can be a float, integer and Boolean and will be discussed in case studies. Crossover and mutation operator will be selected regarding the type of individuals and every method can be used for NSGA-II. But "cxSimulatedBinaryBounded" for crossover and "mutPolynomialBounded" for mutation are suggested by Deb who have developed NSGA-II. These methods will be explained in the next part.
3.3 Case studies

3.3.1 Homogeneous Individuals

Three kinds of individuals are considered in MOO-toolbox: Float, Integer and Boolean.

**Float Individuals:**
To solve the problems with float variables, these types of crossover and mutation are considered:

*For crossover:*
\[
\text{deap.tools.cxSimulatedBinaryBounded}(\text{ind1}, \text{ind2}, \text{eta}, \text{low}, \text{up})
\]
Implements a simulated binary crossover that changes in-place the input individuals. The simulated binary crossover receives sequence individuals of floating point numbers.

- \(\text{ind1}\): The first individual which participates in the crossover.
- \(\text{ind2}\): The second individual which participates in the crossover.
- \(\text{eta}\): Crowding degree of the crossover. A high \(\text{eta}\) can create children similar to their parents, while a small \(\text{eta}\) will create solutions much more different.
- \(\text{low}\): Lower bound of the search space.
- \(\text{up}\): Upper bound of the search space.
- Returns a tuple of two individuals.

*For mutation:*
\[
\text{deap.tools.mutPolynomialBounded}(\text{individual}, \text{eta}, \text{low}, \text{up}, \text{indpb})
\]

- \(\text{individual}\): Sequence individual to be mutated.
- \(\text{eta}\): Crowding degree of the crossover. A high \(\text{eta}\) can create children similar to their parents, while a small \(\text{eta}\) will create solutions much more different.
- \(\text{low}\): Lower bound of the search space.
- \(\text{up}\): Upper bound of the search space.
- Returns A tuple of one individual.

**Integer Individuals:**
To solve the problems with Integer variables these types of crossover and mutation are considered:

*For crossover:*
\[
\text{deap.tools.cxUniform}(\text{ind1}, \text{ind2}, \text{indpb})
\]
Implements a uniform crossover that change in place the two sequence individuals. The characteristics are exchanged based on the \(\text{indpb}\) probability.

- \(\text{ind1}\): The first individual which participates in the crossover.
• ind2: The second individual which participates in the crossover.
• indpb: Independent probability for each characteristic to be exchanged.
• Returns a tuple of two individuals.

For mutation:
deap.tools.mutUniformInt(individual, low, up, indpb):
Mutate an individual by replacing characteristic, with probability indpb, and it can accept an integer value which uniformly drawn between low and up.

• individual: Sequence individual to be mutated.
• :low: The lower bound.
• up: The upper bound.
• indpb: Independent probability for each characteristic to be mutated.
• Returns a tuple of one individual.

Boolean Individuals:
To solve the problems with Boolean variables, these types of crossover and mutation are considered:

For crossover:
deap.tools.cxTwoPoint (ind1, ind2):
implements a two-point crossover on the sequence individuals. The two individuals are changed in place and both keep their original length.

• ind1: The first individual which participates in the crossover.
• ind2: The second individual which participates in the crossover.
• Returns a tuple of two individuals.

3.3.2 Heterogeneous Individuals

To solve the problem with different kinds of individuals such as float, integer and Boolean there should be a special method to apply crossover and mutation operators successfully. In fact, there are different types of crossover and mutation and some of them are suitable for float individuals and some others are better for integer individuals among DEAP operators. In other words, there is not a specific operator which can handle different kinds of individuals at the same time. Therefore two approaches are considered to handle this issue:

First approach: Break individuals and apply operators separately:
To apply the crossover and mutation in GA, the algorithm put all individuals next to each other as a chromosome. In this approach. The chromosome will break and then the crossover and mutation will be applied on every individual separately. For example, a crossover which is suitable for float number will be used for float variable and another crossover will be used for integer variables. The Python code for crossover is:
def my_crossover(self, LL, UU, ind1, ind2):
    t = self.t
    typeOfInput = len(t)
    ind11 = []
    ind22 = []
    for i in range(typeOfInput):
        if t[i] == 'float':
            ind1var1, ind2var1 = tools.cxSimulatedBinaryBounded([ind1[i]], [ind2[i]], low=LL[i], up=UU[i], eta=20.0)
            ind11.append(ind1var1[0])
            ind22.append(ind2var1[0])
        elif t[i] == 'int':  
            ind1var2, ind2var2 = tools.cxUniform([ind1[i]], [ind2[i]], indpb=0.1)
            ind11.append(ind1var2[0])
            ind22.append(ind2var2[0])
        elif t[i] == 'bool':
            ind1var3, ind2var3 = tools.cxTwoPoint([ind1[i]], [ind2[i]])
            ind11.append(ind1var3[0])
            ind22.append(ind2var3[0])
    return ind11, ind22

The Python code for mutation is:

def my_mutation(self, LL, UU, individual):
    t = self.t
    typeOfInput = len(t)
    individual = []
    for i in range(typeOfInput):
        if t[i] == 'float':
            ind1 = tools.mutPolynomialBounded([individual[i]], low=LL[i], up=UU[i], eta=20.0, indpb=1.0/len(self.t))
            individual.append(ind1[0][0])
        elif t[i] == 'int' or t[i] == 'bool':
            ind2 = tools.mutUniformInt([individual[i]], low=LL[i], up=UU[i], indpb=0.9)
            individual.append(ind2[0][0])
    return individual

But this method is not efficient enough and gives the correct answer sometimes. Therefore, the second approach will be used.

Second approach: Convert individuals to binary numbers:
If the different individuals are converted to binary numbers, it is not important how much types of individuals are available, because after converting, at last, there will be only a binary number. Therefore, one type of crossover and mutation which are suitable for binary numbers can be used. The crossover which is selected for binary numbers is "cxTwoPoint" from DEAP documentation. Plus, "mutFlipBit" is selected for mutation. Float and integer numbers are little different from each other to convert to binary individuals.

For converting Float individuals 41 bits is considered. At first, in "binaryBuilder" function, some random points with the size of population will be created in these bits. The biggest number which is possible to create with 41 bits is 2199023255551 and it seems enough and creates every number for the algorithm. But it is important to know the main function in which the objectives are defined only can accept a real number so it is necessary to define a "deBinerize" function to
convert the binary numbers to real ones. In this function, at first, the binary numbers which were created in "binaryBuilder" will be converted to real numbers, then, they will be mapped between lower bound and upper bound by using the formula

\[ x = \frac{Real_{number} \times (upper\_bound - lower\_bound)}{2^{Nrofbits} - 1} + lower\_bound \]  

(7)

Where \( x \) is a number between lower bound and upper bound.

The following example can make the process more clear:
if \(-5 < x < 5\) and \(Nrofbits = 4\) then according to equation (7) the minimum number which can be created in binary shape corresponds to the lower bound and the maximum number corresponds to the upper bound. Figure 8 shows the cases.

![Figure 8: (a) Lower bound (b) Upper bound.](image)

As mentioned above, 41 bits is considered. For float number, increasing the number of bits can increase the accuracy of the algorithm because there are infinite float numbers between lower and upper bound. Therefore, more bits make possible to create more numbers between lower and upper bounds.

For converting Integer individuals whole procedure is similar to the previous one as it is done for float numbers. However, there is a different about the number of bits for integer numbers. For integer numbers, increasing the number of bits can decrease the accuracy of the algorithm because there are finite integer numbers between lower and upper bounds. In fact, increasing the number of bits lead to creating some repeated numbers which are not desirable for the algorithm to search and can cause a wrong solution. Therefore, it is necessary to consider the bits’ number as much as we need. As a result by using formula (8) we can reach the goal.

\[ Nrofbits = \log_2(upper\_bound - lower\_bound + 1) \]  

(8)

Then NrofBits will be rounded to up.

### 3.3.3 Constraint Handling

Some algorithms have the capability of handling constraints inbuilt in them but some other such as GA cannot. Then a new problem formulation is needed.

**Non-linear constraints:**
If we have the original formulation as equation (9):

\[
\begin{align*}
\min F(x) & \quad \text{Objective function} \\
g_j(x) & \leq 0 \quad j = 1, 2, 3, \ldots \quad \text{inequalityConstraints} \\
h_k(x) & = 0 \quad k = 1, 2, 3, \ldots \quad \text{equalityConstraints}
\end{align*}
\]

(9)

To handle constraints equation (10) to (16) shows the formulation with Penalty Functions:

\[
F = f(x) + \left[ \sum_{i=1}^{n} (w_i G_i) + \sum_{j=1}^{p} (v_i L_i) \right]
\]

(10)

\[
G_i = \max[0, g_i(x)]^\beta
\]

(11)

\[
L_i = [h_j(x)]^\gamma
\]

(12)

\[g_i(x)\] will be used if there is inequality constraints and \(h_j(x)\) will be used if there is equality constraints and \(\beta\) and \(\gamma\) will be considered 1 and more than 1. \(w_i\) and \(v_i\) are considered a big value. Moreover, if we want to maximize the function the equation will be:

\[
F = f(x) - \left[ \sum_{i=1}^{n} (w_i G_i) + \sum_{j=1}^{p} (v_i L_i) \right]
\]

(13)

**Linear constraints:**

The original formulation is considered as equation (14):

\[
\begin{align*}
\min F(x) & \quad \text{Objective function} \\
Ax & \leq b \quad \text{inequalityConstraints} \\
Aeqx & = beq \quad \text{equalityConstraints}
\end{align*}
\]

(14)

To handle constraints if \(f(x)\) should be minimized the equation (10) and if \(f(x)\) should be maximized the equation (13) shows the formulation with Penalty Functions where:

\[
G_i = \max[0, (Ax - b)]^\beta
\]

(15)

\[
L_i = [Aeqx - beq]^\gamma
\]

(16)

### 3.3.4 Bridging openMDAO and DEAP

In this step, some investigation and effort are done to bridge OpenMDAO and DEAP. OpenMDAO has some facilities to do problem formulation. To connect the outputs and inputs of different objectives which cause data to flow between two systems in a model and clarify the relation between different objectives. In fact, OpenMDAO has two main parts, one part is to set up the variables and define the problem formulation and the other part is solvers like different algorithms. And in this step, the goal is combining the first parts of OpenMDAO and this current
To be more clear the following example is solved within OpenMDAO:

\[
\begin{aligned}
\min F(x) &= (x - 3)^2 + xy + (y + 4)^2 - 3 \\
-50 \leq x, y \leq 50
\end{aligned}
\]

Objective function

Variable limits

The following code shows the solution in OpenMDAO. As shown in picture from line 4 to 12 the problem formulation is defined and from line 15 to 23 the optimization driver will be run.

In fact, the idea is to bridge OpenMDAO and DEAP to use the facilities of OpenMDAO to define the problem and use DEAP as optimization driver.

The Python code is:

```python
from openmdao.api import Problem, ScipyOptimizeDriver, ExecComp, IndepVarComp

# build the model
prob = Problem()
indeps = prob.model.add_subsystem('indeps', IndepVarComp())
indeps.add_output('x', 3.0)
indeps.add_output('y', -4.0)
prob.model.add_subsystem('paraboloid', ExecComp('f = (x - 3)**2 + xy + (y+4)**2 - 3'))
prob.model.connect('indeps.x', 'paraboloid.x')
prob.model.connect('indeps.y', 'paraboloid.y')

# setup the optimization
prob.driver = ScipyOptimizeDriver()
prob.driver.options['optimizer'] = 'SLSQP'
prob.model.add_design_var('indeps.x', lower=-50, upper=50)
prob.model.add_design_var('indeps.y', lower=-50, upper=50)
prob.model.add_objective('paraboloid.f')
prob.setup()
prob.run_driver()
```
CHAPTER 4

4 RESULTS: MOO-Toolbox

In this chapter, the results of creating the MOO-toolbox are presented including how the MOO-toolbox should be used as well as an evaluation of the MOO-toolbox and a comparison with another method.

4.1 The designed MOO-toolbox

The following code shows the main page which the user should work with and enter the information regarding the problems.
On the first line the user should enter the types of inputs as 'float' for float individuals, 'int' for integer individuals or 'bool' for Boolean individuals.
On the second line should specify the number of generation.
From line 3 to 6, should enter the coefficient of variables in linear inequality constraint and linear equality constraint.
On line 7 and 8, the lower and upper bound of variables should be entered.
On line 9, the duty of optimizer for objectives should be defined and the user defines 'min' for minimizing the objective and defines 'max' for maximizing the objective.
On line 10, the function of non-linear constraint will be defined.
On line 19, the main objectives will be defined.
On line 26, the main class in which the main code of MOO-toolbox is available is called.
On line 27, the inputs of this class are set up of the MOO-toolbox. The first four inputs (myfun, typeOfInputs, obj, Ngen) are mandatory to run the MOO-toolbox. But the others are optional and the user can put them as inputs if they are available but it is recommended to specify the lower and upper bound of variables because the algorithm cannot present a correct answer sometimes without lower and upper bounds.
Line 28 and 28 are some output that the user needs them to plot the pareto fronts. How to work with MOO-toolbox is explained in details in following.

```python
1 typeOfInputs = ['float', 'float']
2 Ngen = 100
3 A = []
4 b = []
5 Aeq = []
6 beq = []
7 lb = [0] + [0]
8 ub = [5] + [3]
9 obj = ['min', 'min']
10 def nonlcons(x):
```

28
4.1.1 How to use MOO-toolbox in more details

In this section, the use of MOO-toolbox is explained in more detail. The following code shows the solution of example in case study 3 in results chapter. Equation 22.

def myfun(x):
    x1=x[0]
    y=x[1]
    f1=...
    f2=...
    return f1, f2

my_ga = ga_generic()
my_ga.setup(myfun, typeOfInputs, obj, Ngen=Ngen, lowerB = lb, upperB = ub, A=A,b=b,Aeq= Aeq,beq=beq, nonlcons=nonlcons)
pop, logbook, ind = my_ga.main()
front = np.array([ind. fitness .values for ind in pop])
return f1, f2
my_ga = ga.generic()
my_ga.setup(myfun, typeOfInputs, obj, Ngen=Ngen, lowerB = lb, upperB = ub, A=A, b=b, nonlcons=nonlcons)
pop, logbook, ind = my_ga.main()
front = np.array([ind.fitness.values for ind in pop])
plt.scatter(front[:,0], front[:,1], c="b")
plt.axis("tight")
plt.xlabel("f1")
plt.ylabel("f2")
savefig('example3.png', format='png', dpi=1000)
plt.show()

In this example there are six variables which are float. And number of generation is 500. The constraints are written in the shape as equation [18] but for writing the coefficients of variables in matrix A and b they should be written as it is explained in chapter 3 in equation [14]. Then the constraints are converted to the shape as we need in equation [19]

\[
s.t = \begin{cases} 
g_1(x, y) = x_1 + x_2 - 2 \geq 0 & \text{linear} 
g_2(x, y) = 6 - x_1 - x_2 \geq 0 & \text{linear} 
g_3(x, y) = 2 - x_2 + x_1 \geq 0 & \text{linear} 
g_4(x, y) = 2 - x_1 + 3x_2 \geq 0 & \text{linear} 
g_5(x, y) = 4 - (x_3 - 3)^2 - x_4 \geq 0 & \text{nonlinear} 
g_6(x, y) = (x_5 - 3)^2 + x_6 - 4 \geq 0 & \text{nonlinear} 
\end{cases} 
\]

\[
s.t = \begin{cases} 
g_1(x, y) = -x_1 - x_2 \leq -2 & \text{linear} 
g_2(x, y) = x_1 + x_2 \leq 6 & \text{linear} 
g_3(x, y) = x_2 - x_1 \leq 2 & \text{linear} 
g_4(x, y) = x_1 - 3x_2 \leq 2 & \text{linear} 
g_5(x, y) = -4 + (x_3 - 3)^2 + x_4 \leq 0 & \text{nonlinear} 
g_6(x, y) = -(x_5 - 3)^2 - x_6 + 4 \leq 0 & \text{nonlinear} 
\end{cases} 
\]

Then, the matrix "A" for constraint \(g_1(x)\) become [-1, -1, 0, 0, 0, 0] because in \(g_1\) only there are x1 and x2 and the other variables coefficients are 0. And "b" will be -2 for \(g_1\). It is important to notice, when there are several constraints, to fill "A" the coefficients of every constraints should be placed in one list. But for "b", all numbers should be placed beside each other. After this step, in "nonlcons" function, \(g_5\) and \(g_6\) are defined. And objectives are defined in "myfun". As you see on line 33, for setup there are 8 inputs and the first four are mandatory and the second four are available because they are available for this problem. On line 35, we use front as data to plot the pareto front.

### 4.2 Evaluation of MOO-toolbox

In this section, different case studies are used to compare the results which obtain from current MOO-toolbox with the results of other tools such as MATLAB and Modefrontier.
Case study 1:

This example is defined in equation [20] and it is selected to test the MOO-toolbox ability for handling several inequality linear constraints, the result is shown in the figure [9]

\[
\begin{align*}
\text{Minimize} &= \begin{cases} 
  f_1(x, y) = x \\
  f_2(x, y) = \frac{1+y}{x} \\
  0.1 \leq x \leq 1 \\
  0 \leq y \leq 5 
\end{cases} \\
\text{s.t} &= \begin{cases} 
  g_1(x, y) = y + 9x \geq 6 \\
  g_1(x, y) = -y + 9x \geq 1 
\end{cases}
\end{align*}
\tag{20}
\]

![Figure 9](image)

Figure 9: (a) Result from MOO-toolbox, (b) Result from MATLAB.

Case study 2:

This example is defined in equation [21] and it is selected to test the MOO-toolbox ability for handling several inequality non-linear constraints, the result is shown in the figure [10]

\[
\begin{align*}
\text{Minimize} &= \begin{cases} 
  f_1(x, y) = x \\
  f_2(x, y) = (1 + y)\exp(-\frac{x}{1+y}) \\
  0 \leq x \leq 1 \\
  0 \leq y \leq 1 
\end{cases} \\
\text{s.t} &= \begin{cases} 
  g_1(x, y) = \frac{f_2(x,y)}{0.858\exp(-0.541f_1(x,y))} \geq 1 \\
  g_1(x, y) = \frac{f_2(x,y)}{0.728\exp(-0.295f_1(x,y))} \geq 1 
\end{cases}
\end{align*}
\tag{21}
\]
Case study 3:

This example is defined in equation 22 and 23 and it is selected to test the MOO-toolbox ability for handling both several inequality non-linear constraints and inequality linear constraints, the result is shown in the figure 11

Minimize = \[
\begin{align*}
& f_1(x, y) = -25(x_1 - 2)^2 - (x_2 - 2)^2 - (x_3 - 1)^2 - (x_4 - 4)^2 - (x_5 - 1)^2 \\
& f_2(x, y) = \sum_{i=1}^{6} (x_i)^2 \\
& 0 \leq x_1, x_2, x_6 \leq 10 \\
& 1 \leq x_3, x_5 \leq 5 \\
& 0 \leq x_4 \leq 6
\end{align*}
\] (22)

s.t = \[
\begin{align*}
& g_1(x, y) = x_1 + x_2 - 2 \geq 0 \\
& g_2(x, y) = 6 - x_1 - x_2 \geq 0 \\
& g_3(x, y) = 2 - x_2 + x_1 \geq 0 \\
& g_4(x, y) = 2 - x_1 + 3x_2 \geq 0 \\
& g_5(x, y) = 4 - (x_3 - 3)^2 - x_4 \geq 0 \\
& g_6(x, y) = (x_5 - 3)^2 + x_6 - 4 \geq 0
\end{align*}
\] (23)

Figure 10: (a) Result from MOO-toolbox, (b) Result from MATLAB.

Figure 11: (a) Result from MOO-toolbox, (b) Result from MATLAB.
Case study 4:

This example is defined in equation (24) and it is selected to test the MOO-toolbox ability for handling different kinds of individuals such as float and integer. The result is shown in the figure 12.

\[
\begin{align*}
\text{Minimize} & = \begin{cases} 
 f_1(x, y) = 2 + (x - 2)^2 + (y - 1)^2 \\
 f_2(x, y) = 9x - (y - 1)^2 
\end{cases} \\
\text{s.t} & = \begin{cases} 
 g_1(x, y) = x^2 + y^2 \leq 225 \\
 g_1(x, y) = x - 3y + 10 \leq 0 
\end{cases} \\
\end{align*}
\]

\[(24)\]

Figure 12: (a) Result from MOO-toolbox, (b) Result from MATLAB.

Case study 5:

This is a more complex example which is solved in Modefrontier and it is solved also in the MOO-toolbox to compare the results. In this example, is supposed to optimize simultaneously structural and aerodynamic performance of a wing that will be used for a Human Powered Aircraft. There are three disciplinary models which include Aerodynamics in which Lift and Drag is calculated, Sizing in which the wing geometry is calculated, Structure in which the strength of the wing is calculated. The optimization objectives are to minimize the weight while maximizing the lift to drag ratio and the constraint is stress which should be less than 300Mpa. Figure 13 shows the solution of example in Modfrontier and there is 1 node of MATLAB and 2 nodes of Excel which are used in Modefrontier.
To solve this problem with MOO-toolbox, bridging between MATLAB and Python is done but the formulas in excel file is extracted and is written in Python. However, bridging between Python and Excel is also possible. To read Matlab files in P, at first, the requirements are:

Matlab version 2014 or above.
Python version 2.7, 3.4 or 3.5.

Then, it is necessary to import `matlab.engine` as a module in Python. The following code shows how it is possible to run MATLAB from Python and extract an output from MATLAB to use in Python. For example, in this case, the function that should be run in MATLAB is Aerodynamics2 and Lift is the output which is extracted from MATLAB. [33]

```python
import matlab.engine
eng = matlab.engine.start_matlab()
result = eng.Aerodynamics2
Lift = result['L']
```

Figure 14 shows the results of example in the MOO-toolbox and Modefrontier.

![Figure 13: The framework of the problem in Modfrontier](image)

![Figure 14: (a) Result from MOO-toolbox, (b) Result from Modefrontier.](image)
4.3 Bridging MOO-toolbox and OpenMDAO

In this section, bridging between OpenMDAO and MOO-toolbox is done. In fact, defining the problem on the line 18 to 28 is done by using OpenMDAO and it is connected to the MOO-toolbox for optimization. This example was solved in chapter 3 (Method: MOO-toolbox) only with OpenMDAO. And now it is solved in this MOO-toolbox. The answer of optimization is the same in both methods.

```python
from openmdao.api import Problem, ScipyOptimizeDriver, ExecComp, IndepVarComp
generic6 import ga_generic

typeOfInputs = ['float', 'float']
Ngen=10
A = [] #matrix for inequality linear constraint
b = []
Aeq = []
beq = []
lb = [-50, -50]
ub = [50, 50]
obj=['min']
def nonlcons(individual):
    return g1, g2
def eqnonlcons(x):
    ceq = []
    return ceq
def myfun(x):
    prob = Problem()
    indeps = prob.model.add_subsystem('indeps', IndepVarComp())
    indeps.add_output('x', x[0])
    indeps.add_output('y', x[1])
    prob.model.add_subsystem('paraboloid', ExecComp('f = (x-3)**2 + x*y + (y+4)**2 - 3'))
    prob.model.connect('indeps.x', 'paraboloid.x')
    prob.model.connect('indeps.y', 'paraboloid.y')
    prob.setup()
    prob.run_model()
    return (prob['paraboloid.f'][0])
my_ga = ga_generic()
my_ga.setup(myfun, typeOfInputs, obj, Ngen=Ngen, lowerB=lb, upperB=ub)
```

One important thing about bridging between these two toolboxes is the time of optimization run. The following table compare the time of optimization based on number of generation for several example which were tested. In fact, this bridging maybe is not efficient regarding time but anyway it is an option for the users if they want to use this connection.
Table 3: Comparison of optimization time between using MOO-toolbox only and MOO-toolbox + OpenMDAO

<table>
<thead>
<tr>
<th>Method</th>
<th>Nr of generation</th>
<th>Optimization time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOO-toolbox only</td>
<td>1000</td>
<td>30 seconds</td>
</tr>
<tr>
<td>MOO-toolbox+OpenMDAO</td>
<td>1000</td>
<td>10 minutes</td>
</tr>
</tbody>
</table>
CHAPTER 5

5 DISCUSSION: MOO-Toolbox

This chapter contains the discussion about the MOO-toolbox, including analyses of the methodology used during this thesis and the result obtained.

5.1 MOO-Toolbox

The goal to create this toolbox was an open-source, user-friendly and fast toolbox. Moreover, being generic as much as possible. According to the results chapter and comparisons, The MOO-toolbox seems confident enough to use for more complicated problems. It can handle problems with different kinds of individuals and with different kinds of constraints. Also, bridging it with other toolboxes can give so much freedom to the user to solve different kinds of problems. For example, bridging with MATLAB, with OpenMDAO, reading and writing XML file and SQL databases are kinds of benefits of working in Python and usability of this toolbox.

5.2 Capabilities and Advantages

- MOO-toolbox is an open-source toolbox which can be used freely.

- MOO-toolbox can bridge with different toolboxes such as MATLAB, excel and OpenMDAO and this capability can make easier to solve different kinds of problems.

- Also it can solve the problem with the variables in vector-shape. For example, one variable can have several values as a vector in a list. As it is in Weland case which will be discussed in following chapters.

- Optimization time is not too much and it is fast.

- It is easier to use, especially compare to OpenMDAO or using DEAP directly.

- It is powerful to cover different types of multi-objective problems with different kinds of variables.
5.3 Future works

- MOO-toolbox contains genetic algorithm and as a generic toolbox, other algorithms can be added as an option.

- To solve unconstrained problems the MOO-toolbox needs at least an initial guess about the range of variables.

- In general, the formulation for handling the constraints can be more dynamic. Besides, handling equality constraints can improve to have more accurate results when the problem has equality constraints.

- The user should know exactly how inputs should be given and return the outputs from the function of objectives. For example, the user should not put them in a list.

- Bridging with OpenMDAO can make the MOO-toolbox very slow and maybe it is not efficient to use. Anyway, it is an option and it is up to the user to want to use it or not.

- MOO-toolbox does not have visualization facilities such as Modefrontier. These kinds of facilities can be added.

These limitations can be solved with more investigation and extending the code structure and make it more user-friendly and easier to use.
PART III
WELAND Design Optimization
CHAPTER 6

6 THEORY: Weland Optimization

In this chapter, the theory used to solve the Weland problem to optimize the cost and usability of spiral staircases is presented. First, some basic theory about the structure of spiral staircases which is used in the mathematical model for usability is given. After that, some short theory about the cost of products, SQL database and XML-file are presented. Moreover, the theory of how to rank the designs of pareto font and TOPSIS method is given.

6.1 Spiral Staircases

Weland’s spiral staircases are products that can be different in many different ways to meet different purposes. The stairs can be used both in the exterior and interior applications, such as escape stairs in an industrial environment or aesthetic appealing space-saving options in an indoor environment. Regardless of the purpose, function and appearance of the staircase, the staircase is designed in the same way with the same kind of parts such as steps, follower, center tube, railings, etc. with different configurations. Some of the different design parameters and standards which are used in problem formulation for these parts of staircases are explained in the following sections.[34]

6.1.1 Staircase Radius

The staircase radius is the measurement from the center of the staircase to the outer edge of the handrail. It can be 600 to 1500 mm with 100mm increment.

Figure 15: Visualizes how the staircase radius is measured.[34]
6.1.2 Step Rise and Depth

Step rise or step height is the distance between faces of two steps and it is linked to the height between the landings. A proper radius and the number of steps per revolution decide the step rise. The rise should be between 160 and 220 mm. The lower the rise is, the deeper the step needs to be. The normal rise is about 170 to 180 mm.

The formula for the usability or comfortability of staircases is:

\[ 2 \times h + W = 580 - 640 \]  

(25)

Figure 16(a) shows the step height and figure 16(b) shows imaginary walking line which is 250mm from the circumference and it shows how the step depth is measured.

![Figure 16: (a) Step Height (b) Where the step depth is measured.][34]

6.1.3 Turning of Staircase

A spiral staircase twist can be varied in the direction according to need, figure 17(a) is an illustration of a left and a right handed staircase. In other words, they can be clockwise or counterclockwise.

![Figure 17: (a) Clockwise or counterclockwise staircases (b) Headroom.][34]
6.1.4 Headroom

Headroom is measured between the top of a staircase straight up to the next part to ensure that there is enough space in the staircase to be able to walk in it. Figure 17(b) shows the headroom or free height which should not be less than 2000mm otherwise, it won’t be comfortable for tall people.

6.1.5 Child Safety

For child-proof staircases, the openings are restricted to prevent a hazard. This affects the clearing between steps and balusters. In this case, step height should be less than 200mm instead of 220mm which mentioned in step rise section. Figure 18

![Figure 18: Clearings that affect child safety.][34]

6.2 Cost

In business, calculating the total cost of products is important because it gives a more accurate view of profitability. Plus, cost reduction techniques can be vital for every business because can give them some competitive benefits in hyper-competitive market. And it is important how a company can reduce the cost without affecting the quality of the product.

The total cost of a product includes the following five types:

- **Direct material**: Cost of the materials which gets a major part of the final product. They are raw materials which are an integral part of the final product. For example, raw cotton in textiles, steel in the automobile body.

- **Direct labor**: Wages, benefits, and insurance of the workers who are involved in the production and manufacturing process. For example, workers on the assembly line.

- **Direct expenses**: These include any expense other than direct material and direct labor which directly affect a specific cost unit. For example, the cost of hiring specific machinery or special molds.

- **Factory overhead**: it defines as the cost of indirect materials, indirect labor and indirect expenses.
Indirect materials are the ones needed to complete the goods but its consumption is small or complex and they are production supplies such as lubricants, hand tools. Indirect labor are the ones do not affect the construction of final product such as cleaners, plant guards and material handlers. Indirect expenses cover all indirect expenses from starting of production to its transfer to the store.

- **Selling and distribution overhead**: Selling and distribution overheads refer to the expenses when the product is in a saleable condition. It contains the cost of delivery, advertising, packing, storage.[30]

### 6.3 SQL Database And Python

The database is a systematic collection of data which supports storage and manipulation of data and can make easy the data management. But the database can refer to the data or to the database management system. The Database management system is a software application for communication between users database. Users can be a human resource or other programs. For example, Python program can communicate as a user with a SQL database. The Python standard for database interfaces is the Python DB-API which is a common interface to access databases such as SQL. Structured Query Language (SQL) is a programming language which is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to create, update, retrieve data from a database.[37] SQLite is a simple relational database system, which saves data in regular files or in the internal memory of the computer. SQLite is fast and simple and it can be used for large databases.

### 6.4 XML File

An XML file is an XML (Extensible Markup Language) data file. Its format is like an HTML document, but it uses custom tags to define objects and the data in each object. XML files can be considered as a text-based database. The XML format stores data in a way which is machine-readable and human-readable. There are a lot of programs that can open XML files because they are like a text document. They can be viewed and edited by basic text editors. XML file is a standard way of storing and transferring data between different programs and over the Internet. In this case, data from the user interface in Unreal (sale configurator group) export as an XML file and optimization algorithm through Python read those data.[38]

### 6.5 Ranking Pareto Front

In the case of multidisciplinary optimization, a number of non-dominant solutions are obtained in the form of a Pareto front which is optimal, in order to select which
solution is best, according to weighting the objectives, the best design can be selected by designer. There are various techniques for performing a mathematical selection of solutions on a Pareto front. TOPSIS method is one way to rank the designs on pareto front and is described in following.

**Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS):**[39]

This method is based on calculating an ideal for the target functions, followed by calculating an Euclidean distance from the ideal for each given solution on the Pareto front. The chosen optimal solution should have the smallest Euclidean distance from the ideal solution(positive ideal). Also the largest Euclidean distance from the negative-ideal solution. The ideal solution is a combination of the best value of each objective in the given optimal solutions. On the other hand, the negative-ideal solution is a combination of the worst value of each objective. The first thing is in this technique the matrix of information from the Pareto solutions should be normalized by equation [26]. Where $F$ describes the normalized target function value, $f$ describes the target function value from the optimization, $m$ is the number of solutions and $i$ and $j$ represent which row and column in the matrix.

$$F_{i,j} = \frac{f_{i,j}}{\sqrt{\sum_{i=1}^{m} f_{i,j}^2}}$$ (26)

Then a weighted normalized matrix is calculated by multiplying each normalized target function value $F_{i,j}$ by a weighting for each target function, see equation [27]. Which means that the user needs to specify how important the target functions are compared to each other.

$$v_{i,j} = f_{i,j} \ast w_{i,j}$$ (27)

The next step is calculation of ideal solution, $A^+$ and negative ideal solution, $A^-$. For maximization objective, the best value is the largest value in the column of the objective matrix. Plus, for minimization objective, the best value is the smallest value in the column. These are given by equation [28].

$$A^+ = \{ (\max(v_{i,j}|j \in J), (\min(v_{i,j}|j \in J')|i \in 1, 2, 3..., m) = \{v^+_1, v^+_2, ..., v^+_n\}$$ (28)

where $J$ is the set of maximized objectives and $J'$ is the set of minimized objectives. Next, find the worst value of each objective, that is the smallest and largest value in the column of the objective matrix for maximization and minimization objective. These values form the negative-ideal solution given by equation [29].

$$A^- = \{ (\min(v_{i,j}|j \in J), (\max(v_{i,j}|j \in J')|i \in 1, 2, 3..., m) = \{v^-_1, v^-_2, ..., v^-_n\}$$ (29)

The next step is Calculation of the Euclidean distance between each solution and the ideal and negative-ideal solution:

Distance to positive ideal is:

$$S^+_i = \sqrt{\sum_{j=1}^{n} (v_{i,j} - v^+_j)^2}$$ (30)
where i=1,2,3,...,m
Distance to negative ideal is:

\[ S_i^- = \sqrt{\sum_{j=1}^{n} (v_{i,j} - v_j^-)^2} \]  

(31)

Finally, it is calculated how all solutions are close to the optimal solution.

\[ C_i = \frac{S_i^-}{S_i^- + S_i^+} \]  

(32)

The optimal solution with the largest \( C_i \) is the recommended solution.[39]
CHAPTER 7

7 METHODS:
Weland Optimization

In this chapter, the methods used to optimize the spiral staircases are discussed. At first, the problem is defined, then implementation of Mathematical model and cost model and formulation of the problem are discussed in details.

7.1 Define Optimization Problem

In this part, the spiral staircase problem is defined as an optimization problem. Objectives, variables and constraints are defined. As shown in figure 19, Ergonomy and cost are the main objectives of the problem. The variables are step radius, turning which can be clockwise or counterclockwise, start and end angles for each floor, clearance angle which is minimum of landing for each floor. And the staircase should be childproof or not. Floor height for each floor and the number of floor are not variables because they are decided by customer regarding the situation. There are three constraints which are related to free height that should be more than a specific value and the other constraints are step depth and step height that should not go beyond a specific interval. Mathematical model and cost model will be discussed in details in next sections.

Figure 19: Optimization problem structure
7.2 Ergonomic Model

The mathematical model of the staircase to optimize the usability or comfortability was available from previous works. In this section, the implementation of the model in the MOO-toolbox is discussed. Figure 20 shows the inputs and outputs of the mathematical model. As it is clear, the variables are different compared to previous sections because in this model all of these variables are introduced to control the behavior of the algorithm to have feasible solutions and optimization algorithm can converge more easily. In fact, they are related to main variables somehow in the model, for example, radius index is related to step radius, floor angle clearance placement factor has a relation with start angles, extra sweep tendency is related to step count and step angle and step depth. Constraints in this model are shown in following equation. The constraints intervals are decided according to Weland standards.

\[
\text{Constraint} = \begin{cases} 
-f_{\text{freeHeight}} + 2 \leq 0 & \text{if childproof NO} \\
\text{stepHeight} \leq 0.24 & \text{if childproof NO} \\
\text{stepHeight} \leq 0.2 & \text{if childproof Yes} \\
-\text{stepDepth} \leq -0.13 & \text{if childproof Yes}
\end{cases}
\] (33)

Figure 21 visualizes the model and It shows the main functions in the mathematical model which are Angle, Sweep step and usability. In fact, it shows the relation of different functions and variables within the model. The objective is a usability penalty which is one of the outputs of Usability function and it should be minimized because it is a deviation from usability rules. Then, when it is smaller, the staircase is more comfortable.
As it is discussed in chapter 1 about how to use MOO-toolbox, there are some information that should be filled in regarding the problem. The following code shows those information which are necessary to set up the MOO-toolbox inputs. For example, lower and upper bounds of variables and type of them which integer and float in this case. About the number of variables, for example, there is floor angle clearance scale factor as much as the number of floors and extra step tendency is, number of floor - 1, because it related to number of segments which is also number of floor - 1, because height 0 is considered as floor 1.

```python
1 typeOfInputs = ['int'] + ['float' for i in range(4*nrFl-2)] + ['int']
2 Ngen=100
3 lb = [0] + [0. for i in range(4*nrFl-2)] + [0]
4 ub = [len(radii) - 1] + [1. for i in range(4*nrFl-2)] + [1]
5 obj= ['min','min']
6
7 def nonlcons(x):
8     .
9     .
10     return -min_free_height+free_height_lower,(min_max_step_height[1]−Up_step_height)−
11       min_max_step_depth[0]+0.13
12 def ObjectiveFunction(x):
13     n=len(floor_height)
14     radius_index=np.array([x[0]])
15     floor_angle_clearance_scale_factor =np.array(x[1:1+n])
16     floor_angle_clearance_placement_factor =np.array(x[1+n:(1+2*n)])
17     extra_sweep_tendency= np.array(x[1+2*n:(3*n)])
18     extra_steps_tendency= np.array(x[3*n:(4*n-1)])
19     orientation_index =np.array(x[4*n-1])
20     .
21     .
22
```

Figure 21: Optimization problem structure
An important issue about the orientation of staircases (CW or Ccw) is this option can be variable or not. If the user selects the type of orientation at first, then it is not a variable but if user select option “Cw or Ccw” it is a variable in optimization framework. The following code shows different conditions of type of orientation and it shows when it is not a variable the lower bound and upper bound of this variable (last one) is equal to each other.

```python
if Turning=='Cw or Ccw':
    lb = [0] + [0. for i in range(4*nrFl−2)]+[0]
    ub = [len(radii )−1] + [1. for i in range(4*nrFl−2)]+[1]
elif Turning=='Cw':
    lb = [0] + [0. for i in range(4*nrFl−2)]+[1]
    ub = [len(radii )−1] + [1. for i in range(4*nrFl−2)]+[1]
elif Turning=='Ccw':
    lb = [0] + [0. for i in range(4*nrFl−2)]+[0]
    ub = [len(radii )−1] + [1. for i in range(4*nrFl−2)]+[0]
```

### 7.3 Cost Model

#### 7.3.1 Reading SQLite in Python

To use SQLite, at first, the module sqlite3 should be imported. To use a database, should create a connection object. The connection object represents the database. The argument of connection is where the data will be stored. Then, should read the external SQL script which name is weland-db.sql. Next, should create a Cursor object which uses the cursor method of the Connection object. Then, execute the SELECT statement. Finally, can call the fetchall() method of the cursor object to fetch the data.[40]
7.3.2 SQL Database

In this part, contents and tables of the database are discussed to be clear what is available in the database and how they can be related to each other and how the cost of staircases is formulated for optimization. There are 4 tables in database which should be connected to each other to estimate the cost, such as Part details, Operation details, Work center and BOM (bills of materials).

Figure 22 shows part of part details table. The columns that are needed for this research to calculate the cost are part number, part name, part type which can be 1, 2 or 3 and 1 is for purchased parts, 2 for manufactured parts and 3 is stocked parts, standard price is the price of part which is calculated by Weland for each part, stock balance can be 0 or 1 and 0 shows the parts with part types of 3 are not available in stock now and 1 shows the parts are available in the stock on that time. And EOQ economic order quantity which is a standard number is used for producing a large number of parts because Weland produces parts in large number instead of one by one and this issue can decrease the manufacturing cost. Figure 22 shows some parts of part details table.

Figure 23: Some parts of operation detail table

shows some parts of operation details table which consist of part number, operation name, work center is number of machine which should do that specific operation, set up time is time to set up the machine and unit time is the time that labour works on that operation.

Figure 24 shows the work center table which consists of work center number, work center name, machine cost in sek/hour and labour cost.

Figure 24: Some parts of work center table

shows some parts of work center table which consist of work center number, work center name, machine cost in sek/hour and labour cost.
Figure 25 shows the bills of material tables which consist of part number, BOM number which are children of each specific part, BOM name, quantity and BOM price.

7.3.3 How different tables are related

These tables should be connected to each other somehow to formulate the cost. The variable is considered for the cost is step radius in this research. In fact, the optimization algorithm calculate the cost of different options for radius of staircases and will select the best one regarding the better usability and cost.

At first, the specific radius should be selected in part details tables from part name column (in this step the part name is split in Python to find specific radius from the part name in table). This step is important because this is a first step to connect a variable somehow to the database. Figure 26 shows the name of step which is ”Steg, r1300, H4.20/2” and has a dimension of the radius (1300) in it. Then, the part details tables and operation details can be connected to each other.
Figure 26: Find radius from Part name

via part number which is common for specific part in both tables. Figure 27 shows different operations which should be done for specific step with radius 1300mm.

Then, operation details and work center can be connected via work center number

Figure 27: Different operation for specific step

which is common in these two tables. Figure 28 shows the work center number for each operation of specific step and figure 29 shows for example two of work center number(4410, 4415) in work center table. Also, BOM table and part details tables

Figure 28: Work center number in operation details

can be connected to each other via part number to calculate the material cost. Figure 30 shows the list of different materials of specific step with radius 1300mm.
7.3.4 Cost formulation

There are different types of cost to calculate the cost of a specific step. There is a manufacturing cost and material cost. Therefore, the total cost is the sum of these two costs. There is also a standard cost for each one which is calculated by Weland. And the total cost should be equal to this standard cost. The standard price is not available for all parts in database. So, the manufacturing cost and material cost should be calculated to estimate the total cost.

According to part type in part details table which can be 1 or 2 or 3. It will be decided which way should be considered to calculate the cost. The following code shows different conditions in details. For example, if part type is 1 (purchased) the standard cost will be considered for it. Some times, part type is 2 (manufacturing) and there is also standard price for it, then that cost is considered. But for some cases, part type is 2 but there is no standard price for it, so manufacturing and material cost should be calculated. For part type 3 (stocked) if stock balance is 1, it shows it is available now and if the standard price is available, then the cost is standard price, otherwise, manufacturing and material cost should be calculated.

```python
if 'part_type'==1:
    cost = 'Standard_price'*Nr_Of_Steps
```
elif 'part_type'==2 and 'Standard_price'==0:
    cost = (Manufactured_cost + Material_cost)*Nr_Of_Steps
elif 'part_type'==2 and 'Standard_price'!==0:
    cost = 'Standard_price'*Nr_Of_Steps
elif 'part_type'==3 and 'stock_balance'==1 and 'Standard_price'!==0:
    cost = 'Standard_price'*Nr_Of_Steps
elif 'part_type'==3 and 'stock_balance'==1 and 'Standard_price'==0:
    cost = (Manufactured_cost + Material_cost)*Nr_Of_Steps
elif 'part_type'==3 and 'stock_balance'==0:
    cost = (Manufactured_cost + Material_cost)*Nr_Of_Steps

Manufacturing Cost:

Manufacturing cost is calculated by equation [34]

\[
\text{Manufacturing cost} = \frac{(\text{Setuptime}+(\text{EOQ} \times \text{Unitime}))}{\text{EOQ}} \times (\text{Machinecost} + \text{Labourcost}) \div 60
\] (34)

The division by 60 is because the times are expressed in minutes while the costs are per hour.
For example, to calculate the manufacturing cost for step with radius 900mm is like following equation.

\[
\text{Manufacturing Cost} = \frac{(37.04+1400\times4.26)}{1400} \times (450 + 250) = 50.01 \text{SEK}
\] (35)

Material Cost:

Material cost is calculated by the following equation:

\[
\text{Material Cost} = \sum \text{Quantity} \times \text{BOM price}
\] (36)

where quantity and BOM price are available in BOM tables and every part has different materials with specific price and quantity.
For example for step with radius 900 mm figure [31] shows the list of materials and their price and quantity. And equation [37] shows the way of calculation.
Figure 31: List of materials for step 900mm.

\[
Material_{Cost} = (5.85 \times 1) + (21.71 \times 1) + (22.25) + (22.25 \times 1) + (2331.79724 \times 0.0385) = 161.83
\]

Therefore, material cost for radius 900 is 161.79 SEK and Manufacturing cost is about 50 SEK and the total cost is about 211.86 SEK while the standard price of step 900 is 166 SEK, this difference is because Weland database standard price is not up to date.

7.3.5 Summary of Cost formulation

The figure 32 shows the summary of cost model.

Figure 32: Summary of cost model.
7.4 Reading XML file in Python

To read XML file which includes inputs of optimization framework, ElementTree presents a simple way to process XML files. At first, the module should be imported. The module is imported in the following code as a simplified name (ET in this case). Then, tree structure is created with the parse function, and its root element is obtained. Then, root consists of children which has child tag, child text and child attributes. For example the following line is one line of the XML file:

$$<\text{RadiusLower}> 1100 < /\text{RadiusLower} >$$  \hspace{1cm} (38)

In this line, RadiusLower is child tag and 1100 is a child text.
So, with command "find", it is possible to find a specific child tag and extract its child text and use the data for optimization framework.

```python
import xml.etree.ElementTree as ET

tree = ET.parse('Advanced1.xml')
root = tree.getroot()
floor_height = []
for child in root:
    word = child.tag
    heightFlag = word.find('Height')
    if heightFlag > -1:
        floor_height .append(float(child .text))
```

```
CHAPTER 8

8 RESULTS:
Weland Optimization

In this chapter, the results of optimizing the spiral staircases are discussed. Two of Weland cases are considered as examples to discuss the results of optimization and comparison with the real one in Weland.

8.1 Case 1 (3 floor)

This case is a real case in Weland with 3 floors. And the goal of this section is to compare the results of design optimization and Weland design. Figure 33 and 34 shows the picture of staircase from side and top view.

![Figure 33: Weland case with 3 floors and 2 segments.](image)

8.1.1 Defining Optimization Problem

At first, the optimization problem is defined regarding this case and inputs of optimization set up according to this example. Figure 35 shows the problem definition. Clearance angle is selected according to the top view of the staircase and the
angle of landing which is 90 degree. There is 2 landing but in inputs, there is three number because the first one is necessary for the model to calculate the free height. The start and end angles interval are also selected regarding the top view. In this case, orientation is not a variable.

8.1.2 Pareto front and solution selection

The pareto front regarding cost and usability for this case is shown in figure 8. As it is discussed in the theory section, TOPSIS method is used to select a solution from pareto front. In this method, for each objective should consider a weight to rank the solutions. In this case, one design with the cheapest price and one design which is the most comfortable one are selected. For example, if the cheap design should be selected the weight of usability is 0.05 and the weight of comfortability is 0.95. Therefore the usability will be more important to be selected from pareto front. The following table compares some of design parameters of cheapest design, most comfortable one and real one.
### Table 4: Comparison of Real case, cheap case and comfortable case

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Real case</th>
<th>Cheapest case</th>
<th>Comfortable case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of steps (floor1,floor2)</td>
<td>(20,19)</td>
<td>(16,16)</td>
<td>(21,21)</td>
</tr>
<tr>
<td>Step depth (mm)</td>
<td>(295,255)</td>
<td>(342,294)</td>
<td>(257,235)</td>
</tr>
<tr>
<td>Step angle (deg)</td>
<td>(18,15)</td>
<td>(20.6,17.76)</td>
<td>(15.54,14.22)</td>
</tr>
<tr>
<td>Step height (mm)</td>
<td>(188,75, 196.05)</td>
<td>(235,232)</td>
<td>(179,177)</td>
</tr>
<tr>
<td>Start angles (deg)</td>
<td>(20,90)</td>
<td>(45,94)</td>
<td>(44,85)</td>
</tr>
<tr>
<td>End angles (deg)</td>
<td>(0,0)</td>
<td>(355,1.4)</td>
<td>(355,9.52)</td>
</tr>
<tr>
<td>Landing angles (deg)</td>
<td>(90,90)</td>
<td>(100,98.4)</td>
<td>(90,90)</td>
</tr>
<tr>
<td>Step radius (mm)</td>
<td>1200</td>
<td>1200</td>
<td>1200</td>
</tr>
<tr>
<td>Cost (SEK)</td>
<td>8775</td>
<td>7200</td>
<td>9450</td>
</tr>
<tr>
<td>Usability deviation</td>
<td>0.4</td>
<td>1.4</td>
<td>0</td>
</tr>
</tbody>
</table>

### 8.2 Case 2 (7 floor)

This case is more complicated one with 7 floors and 6 segments, figure 37 shows the case from side and top view.
Figure 37: Weland case with 7 floors and 6 segments.
8.2.1 Defining Optimization Problem

Figure 38 shows problem definition. In this case orientation is a variable (Cw or Ccw) and for entry and exit angles both orientation are considered as inputs. Also, It is not child proof.

<table>
<thead>
<tr>
<th>Objectives</th>
<th>Cost</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius</td>
<td>600&lt;Radius&lt;800</td>
<td></td>
</tr>
<tr>
<td>Turning</td>
<td>Cw or Ccw</td>
<td></td>
</tr>
<tr>
<td>Clearance angle</td>
<td>(90, 30, 30, 30, 30, 90) Degree</td>
<td></td>
</tr>
<tr>
<td>Start angle Ccw,Cw</td>
<td>Cw: [180, 180, 180, 180, 180, 180]</td>
<td></td>
</tr>
<tr>
<td>Start angle Ccw,Cw</td>
<td>Cw: [0, 3770, 7770, 11770, 15770, 19770, 25470] mm</td>
<td></td>
</tr>
<tr>
<td>End angle Ccw,Cw</td>
<td>Cw: [90, 90, 90, 90, 90, 90]</td>
<td></td>
</tr>
<tr>
<td>End angle Ccw,Cw</td>
<td>Cw: [270, 270, 270, 270, 270, 270, 270]</td>
<td></td>
</tr>
<tr>
<td>Height</td>
<td>Free height&lt;2000mm</td>
<td></td>
</tr>
<tr>
<td>Step depth</td>
<td>130&lt;Step depth&lt;380mm</td>
<td></td>
</tr>
<tr>
<td>Step height</td>
<td>150&lt;Step height&lt;240mm</td>
<td></td>
</tr>
<tr>
<td>Child proof</td>
<td>Not child proof</td>
<td></td>
</tr>
</tbody>
</table>

8.2.2 Pareto front and solution selection

The pareto front regarding cost and usability for this case is shown in figure 39. In this case similar to to previous case, one design with cheapest price and one design which is most comfortable are selected.

The following table compares some of design parameters of cheapest design, most comfortable one and real one.
### Table 5: Comparison of Real case, cheapest case and comfortable case

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Real case</th>
<th>Cheapest case</th>
<th>Comfortable case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of steps (floor1,floor2)</td>
<td>(16,17,17,17,17,25)</td>
<td>(16,17,17,17,17,24)</td>
<td>(19,20,20,20,21,29)</td>
</tr>
<tr>
<td>Step depth (mm)</td>
<td>(172,172,172,172,172,190)</td>
<td>(179,181,172,173,173,193)</td>
<td>(173,171,175,171,175,186)</td>
</tr>
<tr>
<td>Step angle (deg)</td>
<td>(20,20,20,20,20,24)</td>
<td>(22.8,23,21.9,22,22,24.6)</td>
<td>(18,17.8,18.3,17.8,18.2,19.4)</td>
</tr>
<tr>
<td>Step height (mm)</td>
<td>(235.6,235.3,235.3,235.3,235.3,228)</td>
<td>(235.6,235.2,235.2,235.2,235.2,237)</td>
<td>(198,200,200,200,190,196)</td>
</tr>
<tr>
<td>End angles (deg)</td>
<td>(110,150,170,180,200,110)</td>
<td>(111,165,186,210,233,115)</td>
<td>(115,159,188,198,233,92)</td>
</tr>
<tr>
<td>Landing angles (deg)</td>
<td>(30,30,30,30,30,100)</td>
<td>(43,30,30,30,36,120)</td>
<td>(65,40,30,30,35,129)</td>
</tr>
<tr>
<td>Step radius (mm)</td>
<td>700</td>
<td>700</td>
<td>800</td>
</tr>
<tr>
<td>Orientation</td>
<td>Ccw</td>
<td>Ccw</td>
<td>Ccw</td>
</tr>
<tr>
<td>Cost (SEK)</td>
<td>14497</td>
<td>14364</td>
<td>19221</td>
</tr>
<tr>
<td>Usability deviation</td>
<td>0.4</td>
<td>1.6</td>
<td>0.87</td>
</tr>
</tbody>
</table>
CHAPTER 7

9 DISCUSSION: Weland Optimization

This chapter contains the discussion about the optimization for Weland case, including analyses of the methodology used during this thesis and the result obtained.

9.1 Problem Formulation

To formulate the problem, the mathematical model is used to define a rule to have a more comfortable staircase and this model calculates step height, step depth, angles and all specifications of a staircase. This model was done by previous works and in this research, some constraints such as childproof are added and the problem formulation gets more organized and clear. Plus, the model was implemented in the new toolbox which is more user-friendly and it has better results because of the genetic algorithm within the toolbox.

On the other hand, to formulate the cost, the information in the SQL database is used to connect different tables and formulate the calculation of the cost of different parts. Even if, in this thesis, the only variables that are related to the database is step radius. But, a generic cost model for all parts in the database is defined which can be useful for future works when the number of variables will be increased because the model can work for all parts to calculate the cost.

9.2 Optimal Results

Based on the optimization result, two stairs were taken in both case 1 and 2, one inexpensive with the smallest possible number of steps but with fulfilled target values for comfortable stairs and one which is more comfortable with higher expenses. If a cheap staircase wants to be obtained then the step depth will increase, however, advantageously it has less number of steps.

The relationship between step height and step depth, based on Weland’s step template is that a comfortable staircase with a depth of 262 mm should have a step height between 165-185 mm, it means that if a cheap staircase wants to be obtained, the step height goes above the target value and thus is given one less comfortable stairs to walk in. But if the staircase should not be used daily This may be a good
enough value.

In case 1, design parameters of real case of Weland are between the cheapest case and comfortable case but closer to the comfortable case which means Weland has preferred a staircase which is a trade-off between cost and comfortability. But with higher attention to comfortable rules. The advantage now is that optimization has evaluated alternative solutions that could give a better result.

In case 2 which is more complicated with 7 floors, Weland real case is close to the cheapest case which makes sense because in this complicated and huge case, the customer prefers to have a product with better cost. Therefore, optimization results can cover different requirements of customers if they want to prefer a cheap staircase or more comfortable one or something between both of them.

Regarding entry and exits angles, in both cases, the start and end angles are optimized. But it can affect the dimension of landing angles. In most of the cases, Weland prefers to have a landing with 90 degree which is a standard part for them to manufacture. And it depends on designer decision to want to have a landing with 90 degree or optimize the entry and exit angles somehow. In fact, in this case, it is necessary to add some variables in a problem to calculate the cost of landing with different angles.

9.3 Future Works

- To improve the mathematical model regarding rules of comfortability and standard intervals of Weland for design parameters.

- To improve the mathematical model regarding angles and formulate it in a better way because now user should enter lots of angles and their lower and upper bounds in both orientations. Therefore, formulate these angles can help the user to enter less information.

- To extend the database to increase variables of optimization because, in this research, the only variable for cost model is step radius, so increasing the variables can give better results.

- Considering handrails, different landings and other parts of staircases as variable to improve the cost model.

- Add delivery time as objectives which need some more information about the time of manufacturing.
CHAPTER 10

10 CONCLUSION

10.1 Research Question 1
How the multi-objective optimization toolbox can be generic and easy to use?

As it is discussed completely in method chapter of part 1, the MOO-toolbox has a user interface which can make it easy to use. In fact, if a user wants to solve an optimization problem with different packages in Python, he should learn exactly what happens in the package and how the algorithm works. But this toolbox has everything in behind and the user just needs to give the information of a problem to the MOO-toolbox interface, then this toolbox can solve it regarding the problem specifications. Therefore, MOO-toolbox is easier and faster for the user to solve a specific problem.

Besides, MOO-toolbox is generic. In fact, it can solve different kinds of problems with different kinds of constraints and variables. The MOO-toolbox is designed to cover different kinds of problems as much as possible. As it is discussed previously, the MOO-toolbox includes different conditions to cover different kinds of constraints such as linear and non-linear ones and it has some function which is able to convert individuals to binary ones. In fact, this method can make the MOO-toolbox powerful to handle problems with heterogeneous individuals. Besides, it is able to manage problems with homogeneous individuals. Another specification which can make this toolbox generic is genetic algorithm. In fact, genetic algorithm is a powerful and fast algorithm to manage multi-objective problems.

10.2 Research Question 2
How can optimization algorithm communicate with sales configurator?

In this research communication between optimization algorithm and sale configurator is done with the help of an XML file. In fact, to make this communication, it was necessary to read and write XML files. At first, the optimization algorithm needs some inputs which are decided by customers and these data should be entered to the user interface of the sale configurator. Then, an XML file which contains these data is produced and will be read by the optimization algorithm in Python.
The next step is writing an XML file which contains optimization results and optimal parameters of spiral staircases. Therefore, this XML file will be read by the sale configurator to build the models with optimal parameters. In the future, better communication can be done to run the optimization from the sales configurator and make the model.

10.3 Research Question 3
How can the database be used together with optimization and a sales configurator to customize staircases?

In fact, the database is not linked to sales configurator directly. The database could help us to formulate the cost and optimize the design parameters of staircases then sales configurator can build the model according to optimization results and visualize an optimal staircase. Therefore, the database is used directly in optimization algorithm and is related to sales configurator indirectly.

10.4 Research Question 4
How can the database be used for solving the optimization problem?

The database includes the information of different parts of the staircase such as operation, their time and price which can be formulated as a math model to optimize the variables. Different data and tables were at first exported as a SQL file, This file was read by Python, programming tools can make everything easy to work with data and formulate the problem. In this case, after importing SQL file, a formula to calculate the cost is defined then different tables and their data are connected to each other to achieve that specific formula for cost calculation.
References


A  First appendix
Product library

$\infty$ nr of stair configurations

Design space

$x$ nr of stair configurations by requirements

Solutions

$y$ nr of optimal stair configurations

1 selected stair configuration

Sales quotation of selected stair configuration
Optimization

- Sales configurator: Customer Requirements
  - Design Variables
  - Constraints

Mathematical model for stair configuration
- Constraints
- Ergonomics objective

ERP
- Cost objective
- Delivery time objective

Sales configurator: Selection of stair configuration

CAD configurator: Detailed modelling of selected stair configuration

Product library
- ∞ nr of stair configurations

Design space
- x nr of stair configurations by requirements

Solutions
- y nr of optimal stair configurations

1 selected stair configuration

Sales quotation of selected stair configuration
**Sales configurator:**

- Customer Requirements
  - Floors (nr. of, heights of each floor)
  - Child proofing (y/n)
  - Turning (cw/ccw)
  - Entry & exit angles
  - Step type (threading)
  - Floors (nr. of, heights of each floor)

**Optimization**

- Design Variables
- Constraints

**ERP**

- Mathematical model for stair configuration
- Free height
- Ergonomics model
- Ergonomics objective

**Item X (of configuration)**

- Stock
- Operations
- Work center 1
- Lead time, item X
- Work center 2
- Lead time, item X

**Cost model**

- Cost objective

**Delivery time model**

- Delivery time objective
Sales configurator
Optimization
CAD configurator
ERP
xml
design
variables
xml
optimized variables
write
read
xml
part info
database
read
read
BOM
write
detailed 3D models
write
Values of design variables are entered in the sales configurator. Design variables are written to an XML file from the sales configurator and then read by the optimization framework. An executable launching the optimization is initiated by the sales configurator. Optimization is run with design variables read from the XML file and part data read from the database XML file. A set of resulting pareto optimal variable values is created in the optimization framework. Configurations from the optimal set are generated, from which one is chosen. The chosen configuration’s variables are written to an XML file and then read by in the CAD configurator, and the ERP interpreter. SQL database is updated with part info from the chosen configuration.

The SQL file database with part info is updated by fetching current data from the ERP system. Values of design variables are entered in the sales configurator. Design variables are written to an XML file from the sales configurator and then read by the optimization framework. An executable launching the optimization is initiated by the sales configurator. Optimization is run with design variables read from the XML file and part data read from the database XML file. A set of resulting pareto optimal variable values is created in the optimization framework. Configurations from the optimal set are generated, from which one is chosen. The chosen configuration’s variables are written to an XML file and then read by in the CAD configurator, and the ERP interpreter. SQL database is updated with part info from the chosen configuration.

ERP
Sales configurator
CAD configurator
UI-input
The SQL file database with part info is updated by fetching current data from the ERP system.
Values of design variables are entered in the sales configurator.
Design variables are written to an XML file from the sales configurator and then read by the optimization framework.
An executable launching the optimization is initiated by the sales configurator.
Optimization is run with design variables read from the XML file and part data read from the database XML file.
The chosen configuration’s variables are written to an XML file and then read by in the CAD configurator, and the ERP interpreter.
SQL database is updated with part info from the chosen configuration.

UI-input
Launch of the CAD-configurator
A detailed CAD model of the chosen configuration is generated in the CAD configurator.
Detailed CAD model, production data (machine code etc.) and drawings are exported from the CAD-configurator. Detailed part info is sent to the ERP interpreter (for update of database).
SQL database is updated with detailed part info from the chosen configuration.