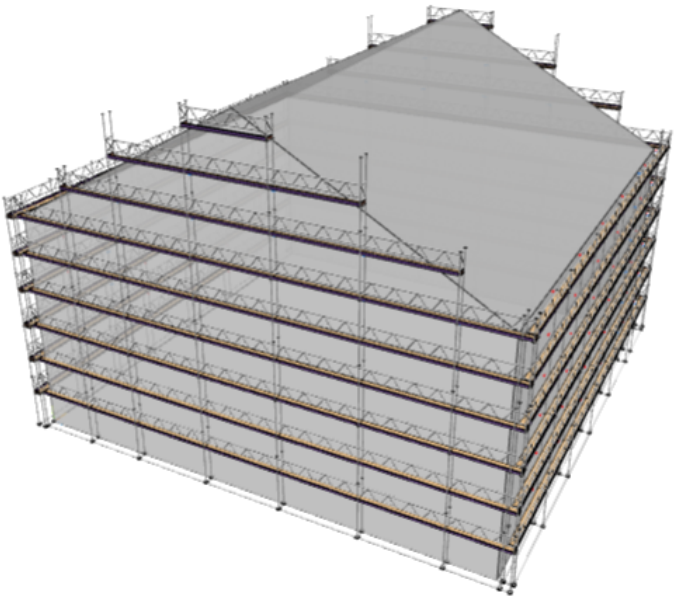


Automated generation of scaffolding topologies

Master's thesis in Systems, Control and Mechatronics

Markus Jernek & Oliver Johansson

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Automated generation of scaffolding topologies

Markus Jernek & Oliver Johansson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks
Chalmers University of Technology
Gothenburg, Sweden 2022

Automated generation of scaffolding topologies

Markus Jernek
Oliver Johansson

Supervisor: Björn Langborn, Doctoral student in Hardware-constrained communications
Examiner: Erik Ström, Head of Division of Communications, Antennas and Optical Networks

Master's Thesis 2022
Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Scaffolding solution produced by the developed algorithm

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Automated generation of scaffolding
Markus Jernek
Oliver Johansson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis presents the development and performance of an algorithm consisting of a multi-objective genetic algorithm, used in conjunction with linear programming, to solve scaffold optimization problems. This is done in order to simplify the design process and automatically generate optimized solutions to increase safety for people within the scaffolding industry. To the author's best knowledge, no equivalent algorithm exists for this application. Three main objectives were selected for the optimization and evaluation of the generated scaffolding solution, namely material usage, scaffolding length to wall length, and excessive scaffolding over the roof edges.

By evaluating the performance of the algorithm both visually and quantitatively to that of scaffolding created manually by users of the ScaffCalcs web application, the results reflected a robust and competitive solution for square buildings with a maximum total wall distance of 48 meters and 1 scaffolding roof story, while time-constrained to 2 seconds, implying a reduction in generation time by a factor of 270. For larger buildings with a total wall distance of 120 meters and 3 scaffolding roof stories, the algorithm was not allowed to fully converge with an imposed time constraint of 2 seconds, but on average generated equivalent solutions compared to the users', with the presence of outliers that would prevent a guaranteed optimal solution in every run. By lifting the time constraint for larger buildings, the algorithm generated optimal solutions with 96% repeatability for an average generation time of 40 seconds. Thus, an algorithm has been successfully developed for the automatic generation of scaffolding topologies, which generates competitive results in a time-efficient manner.

Keywords: MOGA, LP, Scaffolding, Optimization, NSGA-II,

Acknowledgements

First of all, we would like to extend our gratitude to our supervisor Björn Langborn for his continuous support and assistance throughout this thesis.

Many thanks to the members of ScaffCalc who have truly welcomed us in a smashing fashion and invested both time and resources to assist us in our work.

Markus Jernek & Oliver Johansson, Gothenburg, June, 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AFS	Swedish Work Environment Authority's Statute Book
FEM	Finite Element Method
GA	Genetic Algorithm
LP	Linear Programming
MOGA	Multi Objective Genetic Algorithm
MOOP	Multi Objective Optimization
SiS	Swedish Standard Institute
SOGA	Single Objective Genetic Algorithm

List of Scaffolding terms

Below is the list of scaffolding terms that have been used throughout this thesis:

Diagonal brace	Further diagonal support linked between corners
Decks	Component used for standing on which are attached to ledgers
Deck gap	Height between two decks
Ledger	Horizontal link for scaffolding between standards
Consoles	Scaffold module for closing scaffolding gaps to wall facades
Standards	Vertical support of the scaffolding
Roof-fit	Objective name for distance between scaffolding compartment and roof edge
Wall-fit	Objective name for distance between scaffold and wall

Contents

List of Figures

List of Tables

1
Introduction

In this chapter, a brief background to the building industries usage of scaffolds are introduced, combined with some additional methods of how the industry goes about retrieving information and calculations about scaffolds. Thereafter, current software available for visualizing and performing calculations on scaffolds is presented, followed by the main aim of this thesis and the proposed research questions. The chapter is concluded by discussing the delimitation of the thesis and an overview of the remaining chapters.

1.1 Background

Every year, a total of 50 scaffolds collapse, with more than 200 people injured due to unsafe scaffolding projects, in Sweden alone [1]. The majority of injuries are due to falls or faulty scaffolding, both of which can be the result of incorrect scaffold modeling. This in turn can be the result of unqualified scaffolding entrepreneurs, incorrect use of components, or incorrect calculations, something that current digital tools would help reduce.

In today's construction, scaffolding companies are hired to deliver scaffolding solutions to building projects. Such projects can include painting and facade renovation, as well as new construction. One of the ways scaffolding engineers can model the scaffold is to do iterative sketching on a building blueprint with regard to the available scaffolding modules. To guarantee scaffolding safety, they must follow the Swedish industry guidelines and safety regulations mentioned in AFS and SiS, while modeling which have recommendations on scaffolding dimensions related to forces, as well as anchoring recommendations [2], [3]. Another way to model and plan scaffolding for these scaffolding companies is to utilize available software. Most software tools include options for modeling the building and then use graphical tools to place scaffolding modules to match the dimensions of the building. The software and its scaffolding modules are either modeled by following the regulations of AFS and SiS or are totally modifiable to adjust for the scaffolding contractors' own preference or expertise.

Existing software utilizes built-in CAD tools to place bodies of structures/buildings, for which scaffolding modules can be placed manually around said building in a visual representation of reality. By utilizing these software, it is possible to retrieve information that would be of interest to the user. Some software provides calculations

for wind forces with needed reinforcement recommendations, and general practice is to include a tableau of materials needed for the modeled scaffolding. Scaffolding companies can therefore rely on the software to provide sufficient calculations and dimensions of components according to safety guidelines such as AFS [2]. These features make the software very attractive to use for construction companies as it minimizes both time and money spent on scaffolding planning, whilst maintaining and improving upon the safety of the scaffolds by more easily providing and sharing knowledge within construction companies regarding scaffolding modeling and set-ups.

Existing computer aided work

In the current market, several software products are available to provide an easier solution for the scaffold market. Some of the software provides an automated suggestion of a scaffold for the customer [4][5][6]. Two of these three automatic generation software utilize very easy optimization regarding wall-fitting of scaffolding, by using the longest length available for the whole wall and switching out one ledger for a smaller for a better fit at the end. The third option of these software uses the same principle but can switch out more than one ledger to another to achieve a good fit. There are also other software that exist with the only option of manually modeling the scaffold in a digital environment [7][8][9][10].

There have been studies carried out to automatically generate scaffolding surrounding wall faces, as well as optimize the planning phase of scaffolding setups [11]. These studies have not necessarily focused on optimizing scaffolding on separate walls and how well the scaffold fits the wall, but rather operate as a visualization and planning tool to minimize work, scaffold transportation, and cost.

These software bridges the gap between current scaffolding modeling methods and complete digitization within the construction industry. Existing software is consistent with providing clear visualization with different scaffolding module modeling techniques. However, to our knowledge, current software does not perform any multi-objective optimization toward material minimization with respect to the fitting of the scaffold to the facade, which in turn would increase safety by reducing loads and scaffolding gaps while maintaining good affordability and low material requirement.

1.2 Aim and Research Questions

The aim of this thesis is to construct and use an algorithm consisting of a Multi-Objective Genetic Algorithm (MOGA) and Linear Programming (LP) to be used to solve scaffold optimization problems that users can encounter in the ScaffCalcs web application [10]. The constructed algorithm should automatically generate a scaffolding solution around specified squared building dimensions that will be visualized for the user. The research questions that have been posed are then:

- Can the scaffolding solution generated from the developed algorithm improve upon a manual user solution, with respect to material usage and overall fit, while constrained to a 2 second generation time?
- How large of a performance increase with respect to the generation of optimal solutions would the algorithm experience if the time constrain was lifted?

1.3 Delimitations

This thesis will not perform any new calculation for safety measures or make any modifications to the general structure of a scaffolding compartment with respect to safety or load resistance, which is not already implemented in ScaffCalcs software. The construction pattern implemented is in accordance with the Swedish Work Environment Authority Statute Book (AFS) [2] and will be followed in this thesis. The generated solution from the algorithm will not undergo any load calculations, but will focus on optimizing the defined objectives. The work is limited to the current version of the ScaffCalc software with respect to visualization, data collection, and potential building patterns.

1.4 Outline of Report

To further familiarize the reader with optimization and genetic algorithms, Section 2 highlights the important theory behind these concepts, as well as the difference between single and multi-objective optimization. Familiarizing the reader with the algorithmic foundation and fundamental operators used is essential to understand the implications of the generated results. Section 3 outlines the different methods used to evaluate the performance of the algorithm, as well as providing an understanding of the algorithmic logic. Section 4 provides a detailed understanding of the algorithmic structure and the formulation of constraints. Sections 5 and 6 present and discuss the results generated by performing the tests established in Section 3. Finally, Section 7 provides the conclusion of the most important findings throughout this thesis.

2

Theory

In this chapter a thorough introduction to optimization is provided, followed by the distinction between deterministic and stochastic optimization. This is accompanied by an introduction to evolutionary algorithms, where a clear description of a genetic algorithm will be defined along with all its operators. The chapter is then concluded by applying the discussed concept to define a multi-objective genetic algorithm, namely: NSGA-II, followed by a common normalization method for multi-objective problems.

2.1 Single Objective Optimization

Optimization is commonly referred to as finding the optimal solution to an optimization problem by minimizing or maximizing a function over a given set of feasible alternatives, Ω. Generally, this is formulated as

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in \Omega \,. \end{aligned} \tag{2.1}$$

where f is referred to as the objective function and will be considered for problems over $\mathbb{R}^n$ which further implies $f : \mathbb{R}^n \to \mathbb{R}$. The vector $\mathbf{x}$ contains n decision variables such that $\mathbf{x} = [x_1, x_2, ...x_n]^\top \in \mathbb{R}^n$ and will vary over the feasible set of alternatives to either minimize or maximize the objective function. Whether to minimize or maximize the objective function is generally problem dependent, but the difference between the two approaches is merely separated by the multiplication of -1, establishing the following relation.

$$\min_{\mathbf{x}} \; -f(\mathbf{x}) \equiv \max_{\mathbf{x}} \; f(\mathbf{x}) \,. \tag{2.2}$$

Furthermore, most optimization problems only allow $\mathbf{x}$ to take on values within a certain range or region. This is most often referred to as the *feasible region* that contains the full set of alternatives for the decision variables over which the objective function can be optimized. The feasible region is mapped by the construction of *constraints*, which consists mainly of inequality/equality constraints, also known as constrained functions. In general, inequality functions and equality functions are defined as (2.3) and (2.4).

$$g_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, q \tag{2.3}$$

$$h_i(\mathbf{x}) = 0, \quad i = 1, \ldots, p\,. \tag{2.4}$$

By applying the constraints to the problem, all feasible points must satisfy all the $p + q$ constraints, and by doing so Ω in (2.1) becomes the subset of $\mathbb{R}^n$ containing all feasible points, $\Omega \subseteq \mathbb{R}^n$. If, however, $\Omega = \mathbb{R}^n$ then the problem is referred to as an *unconstrained* optimization problem.

Local and Global Optima

The objective of optimization is to locate the global optima of a given problem. To know whether the found solution is a local or global optima, it is important to establish their definitions.

Local Minimum

Assuming the same conditions as in (2.1) it is possible with a strong inspiration to the definition of a strict local minimum as presented in [12] to form the expression for a *local minimum* as:

$$\exists\, \epsilon > 0 : f(x) \geq f(x^*) \quad \forall x \in \Omega \cap \{x : ||x - x^*|| \leq \epsilon\}\,. \tag{2.5}$$

In (2.5) an arbitrary neighborhood ϵ is defined within the set Ω and for the point x^* to be considered a point of local minimum, it has to satisfy the constraint of constructing the lowest objective within the selected neighborhood, thus fulfilling the constraint $f(x) \geq f(x^*)$. Assuming a one-dimensional problem, several points within the objective function could occur that satisfy (2.5), e.g., a horizontal objective function that defines the neighborhood ϵ, as shown in Figure 2.1 between x_3 and x_4. A harsher condition for optimality is a strict local minimum.

Strict Local Minimum

A continuation on (2.5) is displayed in (2.6), and highlights a point x^* that strictly defines the lowest objective and thus claims strict local optima within the neighborhood ϵ [12].

$$\exists\, \epsilon > 0 : f(x) > f(x^*) \quad \forall x \in \Omega \cap \{x : ||x - x^*|| \leq \epsilon\}, \quad x \neq x^*\,. \tag{2.6}$$

As highlighted in Figure 2.1 x_1 and x_2 are two strict local minimums since there exist an $\epsilon > 0$ for each point that satisfies (2.6). Any x within the interval $[x_3, x_4]$ is a local minimum but not a strict minimum as there does not exist an $\epsilon > 0$ that fulfill the condition in (2.6).

Global Minimum

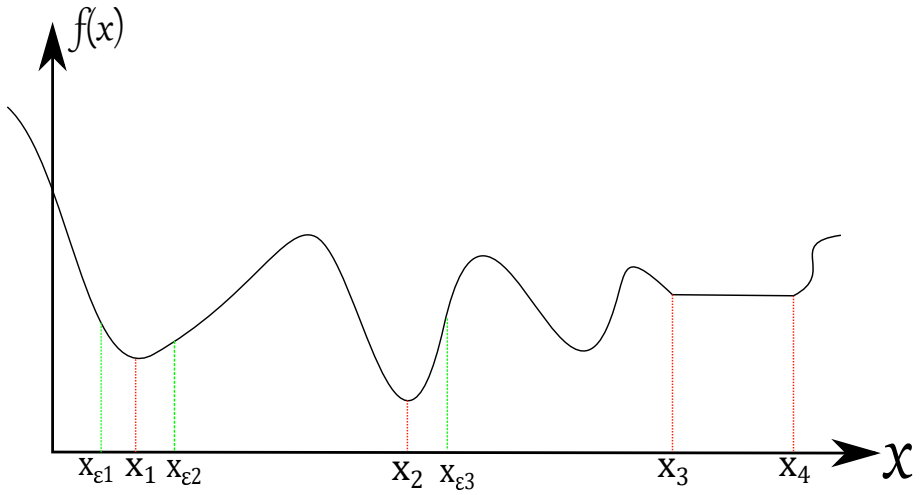


Figure 2.1: *Visualization of local minimum, strict local minimum and global minimum, all dependent on the neighborhood ϵ.*

Similar to the local minimum, the global and strictly global minimum is defined in the set Ω, constructing the following equation for a global and strictly global minimum:

$$f(\mathbf{x}) \leq f(x^*) \quad \forall x \in \Omega \tag{2.7}$$

$$f(\mathbf{x}) < f(x^*) \quad \forall x \in \Omega\,. \tag{2.8}$$

In Figure 2.1 a potential global minimum is found at x_2, but x_1, x_3, x_4 cannot be a global minimum.

Relationship Between Minimum and Maximum

Using (2.2) the local and global maximum of a function could be found by reversing all the inequality signs in (2.5)-(2.8). Furthermore, a function is never always guaranteed to locate one optima; a good example of this would be the function $\sin(x)$ that consists of several global minimum and maximum, and thus no strict global optima.

2.2 Deterministic and Stochastic Optimization

Mathematical Optimization

Mathematical optimization is a deterministic approach to solving optimization problems and contains several methods of optimization tools depending on the problem at hand. One such method is linear programming, which is a powerful tool that has

been present for several decades and is widely used in a variety of different fields [13]. When utilizing linear programming the objective function, equality and inequality constraints have to be linear and the set of all feasible points is a convex polytope, that is, a set of points that satisfy a finite number of affine inequalities. The general structure for linear programming is defined in canonical form as follows.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{a}_i^\top \mathbf{x} + b_i \leq 0, \quad i = 1, \ldots, q \\ \text{and} \quad & x_i \geq 0, \qquad\quad i = 1, \ldots, n\,. \end{aligned} \tag{2.9}$$

Here, $\mathbf{c}$, $\mathbf{x}$, and $\mathbf{a}$ are all vectors of dimensions n, where $\mathbf{x}$ represent the unknown variables that will take values in the feasible set $\mathbb{R}^n$. A common special case of (2.9) is linear integer programming where $x_i \in \mathbb{Z}$. The constraint applied in x_i to only take positive values from (2.9) is motivated by the majority of problems mainly solved by linear optimization. Such being the optimization of parts in production, the number of stops in a complex routing problem, where in most cases the quantities involved are non-negative integers [12].

Stochastic Optimization

The major difference between deterministic and stochastic optimization is the introduction of randomness to the optimization problem, an element that is utilized to gather a large range of different solutions to the optimization problem and influence the algorithm to converge toward an optima. Usually, the random element is introduced in the objective function or the constraint set [14]. Similar to mathematical optimization, stochastic optimization is built upon selecting a set of decision variables to optimize and adopting an appropriate algorithm to carry out the optimization. If the given optimization problem is small and linear, then it is usually common practice to employ linear programming. However, if the problem is complex and highly non-linear then stochastic optimization is favored due to the element of randomness that can assist the stochastic optimization algorithm from escaping a local optima and instead find a global optima. Something that is common as the complexity of the problem grows. Furthermore, higher-dimensional problems containing several decision variables or objective functions are generally associated with non-trivial constraints where the usage of stochastic optimization algorithms are favored over the deterministic optimization [15]. Similar, high-dimensional problems may contain multiple local optima in which deterministic algorithms are more prone to get stuck [16].

The field of stochastic optimization is a large sphere containing several optimization algorithms that have been developed over the years for a variety of different fields [14]. One such domain is evolutionary algorithms that contain optimization algorithms inspired by Darwinism and are population-based metaheuristics [17]. In the next section, the concept of evolutionary algorithms will be further explored.

2.3 Genetic Algorithm

Evolutionary algorithms contain a large number of algorithms intended to solve optimization problems by simulating biological processes encountered during evolution. One such occurrence is the special case of genetic algorithms (GA) that mimic biological evolution but in the domain of numbers. Similarly to Section 2.1 the objective function and decision variables for a given problem that is intended to be solved with the help of an evolutionary algorithm can be structured in a similar way. Once the problem has been identified and correctly formulated, the selection of components and operators has to be recognized along with the choice of realistic hyperparameters which all will be problem dependent. When using evolutionary algorithms, there is no choice of parameters that generate an optimal solution for each and every problem; hence, it is important to understand the usage and effect of these different operators and parameters, which will be discussed in more detail in this section.

Initialization

The first step in solving an optimization problem utilizing a GA is to initialize a population of chromosomes $\mathbf{x}_i$ where $i = 1, 2, \ldots, N$, with N being the number of individuals in a population. There are multiple approaches when conducting the initialization, and in most cases it is problem-dependent, which implies that it is vital to select the corresponding encoding operator of the GA that supports the problem at hand.

2.3.1 Encoding

The premise of utilizing the encoding operator is to enable the mapping of objective variables to a string that can easily be interpreted by the GA. Over the years, several encoding operators have been developed; however, due to the nature of this work, the most suitable approach would be that of value encoding.

Value Encoding

Value encoding is utilized when the search space allows for complicated values, such as real numbers, and when the use of binary coding would be very difficult. The chromosomes in value encoding take the form of strings that can contain a vast variety of values, ranging from real numbers to characters to some complicated object, as demonstrated in Table 2.1.

Table 2.1: *Three different examples of value encoding are highlighted in an attempt to show the power and diversity in value encoding.*

Chromosome A	2.3456 1.0145 8.0975 2.1235 9.1975 5.5576
Chromosome B	ACTGGAOMHEKMNENFJANEBCADDFIEFF
Chromosome C	(left), (forward), (forward), (back), (right), (forward)

2.3.2 Fitness Value

The objective function is problem-dependent and constructed in a way similar to that displayed in (2.1). Evaluation of a certain chromosome is carried out by minimizing or maximizing a given objective function $f(\mathbf{x}_i)$ for all decision variables in that chromosome. When evaluating the objective function, it is feasible to rank each chromosome by assigning it a fitness value. Furthermore, the fitness value will play an important role in later steps during the GA, as it is an output that reflects the goodness or fitness of a given solution to the optimization problem. In (2.10) the general equation for a chromosomes' fitness value is defined.

$$F_i = f(\mathbf{x}_i)\,. \tag{2.10}$$

2.3.3 Selection

The selection process is a stochastic method designed to reflect the breeding that occurs between generations in nature, where the fittest parents have a greater probability of mating and forming an offspring, while the worst still have a small probability of being selected. It is crucial that the algorithm does not completely disregard the worst individuals as this would restrict the search process to a local domain, hindering the algorithm from converging towards a global optima [18]. Furthermore, weaker individuals can show features that can be extremely useful after the recombination steps. Similarly to Section 2.3.1, there are several viable selection methods that all display different methods to select suitable parents for breeding. However, all techniques reflect the biological phenomenon of "survival of the fittest". During this section two different selection methods will be presented, namely: roulette wheel and tournament selection.

Roulette Wheel

Roulette wheel selection builds upon assigning each individual a fitness value that is intended to reflect the probability of being chosen as a parent. Thus, the larger the fitness an individual displays, the higher the probability of being chosen, and the action of spinning the wheel can be seen as the selection process. When implementing this selection method, the wheel itself acts as a metaphor, while in practice it is the cumulative relative fitness value, ϕ_j, that is determined for each parent.

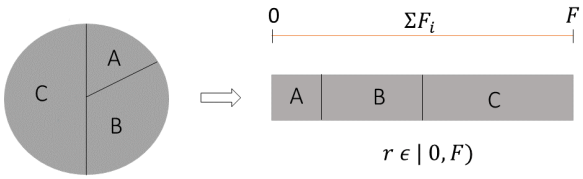


Figure 2.2: *Graphical representation of how roulette wheel selection is conducted.*

$$\phi_j = \frac{\sum_{i=1}^{j} F_i}{\sum_{i=1}^{N} F_i}, \quad j = 1, \ldots, N\,. \tag{2.11}$$

In (2.11) F_i is the fitness of individual i (see (2.10)), this cumulative fitness sum is summed across the entire population N. To determine which parent to select, a random number $r \in [0, 1]$ is generated and the individual with the smallest j such that (2.12) is satisfied and is then selected.

$$\phi_j > r\,. \tag{2.12}$$

As shown in Figure 2.2, it can be easier to visualize the roulette wheel as a domain in $[0, F]$ where F is the numerator in (2.11), with individuals placed along its horizontal axis, from which an individual's size within this domain is proportional to its fitness. This method of selection allows for all individuals to run a chance of being selected and propagated through the GA. However, the risk of premature convergence to local optima is present due to the nature of fitness-proportionality where individuals with dominant fitness will dictate the future of the population [19].

Tournament Selection

In tournament selection, the process of selecting a suitable parent emulates the typical competition between animals better than that seen for roulette wheel selection. This method involves the random selection of t_n individuals from the population, most commonly referred to as the tournament size, that will be "competing" for the role of being selected as a parent.

The process of selecting a single individual for tournament selection occurs with a probability of $\frac{1}{N}$, which implies that no sorting of fitness values is required. Once t_n individuals have been selected, they enter the tournament state where a random number $r \in [0, 1]$ is generated and compared to p, which is a hyperparameter usually set around $0.6 - 0.7$ [12]. If the probability for tournament selection is larger than that of r, the fittest individual is selected. However, if this does not hold, then the

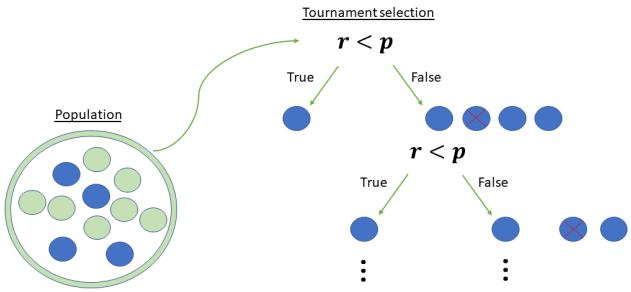


Figure 2.3: *Graphical representation of how tournament selection is conducted.*

fittest individual is disregarded and a new r is generated. These steps are demonstrated in Figure 2.3 and if there is only one individual remaining, it is automatically chosen.

2.3.4 Crossover

Crossover is the GAs equivalence to sexual reproduction and is intended to allow partial solutions from distinct parts of the problems' search space to be explored, thus enabling the algorithm to not converge towards a local search region. However, in general, the population size of a GA is never in the millions as seen in nature, but instead somewhere around 30-1000 [12]. With a smaller population pool, it is vital to regulate the crossover process since a successful individual will spread its genetic material rapidly, implying a reduced diversity, and thus risking the GA to become trapped in a local optimum. To mitigate this risk, a crossover probability is selected, which is problem-dependent but correlated with the size of the population. For a population size ranging from 50-100, it has been suggested that the optimal crossover probability for a single crossover point is 0.6 [20].

Single-Point Crossover

In single-point crossover, a random crossover point is generated independently for two individuals among $n-1$ possible points in a chromosome of length n. As shown in Figure 2.4, the genetic material left of the crossover point is exchanged between the two parents to form two new children.

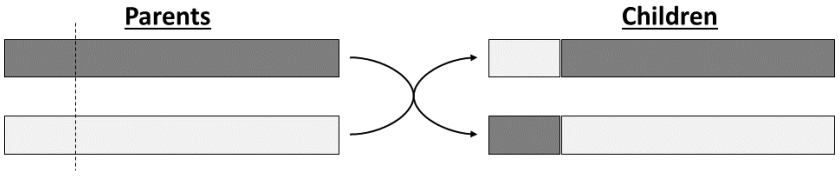


Figure 2.4: *Visualization of a single-point crossover between two parents, with the dashed lines symbolizing the randomly generated crossover point.*

2.3.5 Mutation

To further enable the genetic algorithm to explore more of the search space, mutation is introduced. As this process allows for an increase in exploration, mutation rarely has an instantaneous positive effect on fitness, but rather showcases advantages once in later generations [12]. Using mutation, lost genetic material can be recovered and new genetic material is randomly distributed, making it an insurance policy against irreversible loss of genetic material [21].

When applying mutation to GAs the probability of it occurring is usually set as a parameter beforehand and takes the form of (2.13).

$$\mathrm{p}_{mut} = \frac{c}{n}\,. \tag{2.13}$$

Here, c is a constant of order 1 and n represents the length of the chromosome. Similar to crossover a randomly generated number $r \in [0, 1]$ is generated and compared with p_{mut} for each gene, which implies that r is generated n times for each chromosome. Generally, the process of mutation can involve several different operators, some of the most common being scramble mutation and random resetting. In the next section, these operators will be introduced.

Scramble Mutation

When utilizing scrambled mutation, either a subset of the chromosomes' genes can be selected or the entire domain. The selected zone can be visualized in Figure 2.5 as the yellow area, and once this section is determined, a random permutation of all genes is implemented, resulting in a shuffle of the selected genes.

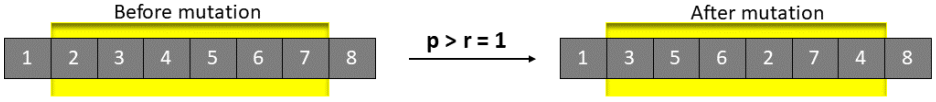


Figure 2.5: *A visual representation of how the scramble mutation operation works, with (p) being the probability of mutation.*

Random Resetting

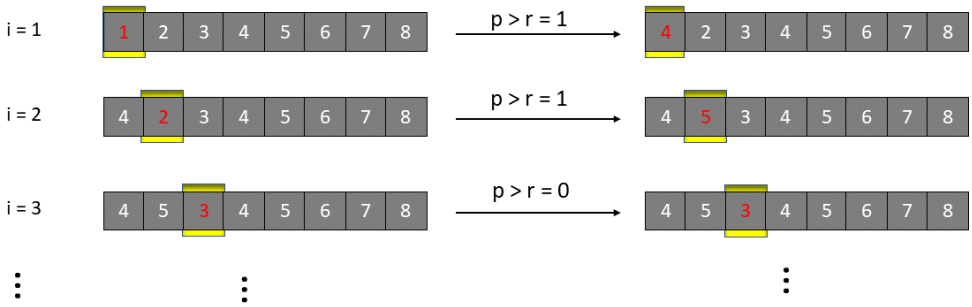


Figure 2.6: *A visual representation of how the random resetting operation works, with (p) being the probability of mutating a given gene i.*

Another heavily used mutation method is the random resetting operator, whose main task is to iterate through the entire chromosome and evaluate each chromosome at a time. As shown in Figure 2.6, i is the count representing each gene, and as the algorithm iterates through the chromosome for each i a random value $r \in [0, 1]$ is generated and compared with a predetermined probability p_{mut}. If the condition $p_{mut} > r$ is true, then a new number is generated from the search space of the objective variable. If, however, the condition does not hold, then that gene remains unchanged, and the iteration continues.

2.3.6 Replacement

If a given search domain is relatively small, the risk of producing duplicates within a generation is possible. In this scenario, the replacement operator is a useful tool for removing duplicates. If the offsprings produced by the parents are identical to the parents, then the need for re-evaluating these individuals is redundant, and these offsprings are instead replaced. When duplicates are located, they are processed through the crossover section once again until a unique set of individuals is generated.

2.3.7 Elitism

In Single-Objective Genetic Algorithms (SOGA) the method of elitism is utilized to preserve a copy of the fittest individual from each generation to be carried over to the next generation without undergoing any change [12]. The intended use of this process is to speed up convergence by continuously taking advantage of the fittest individuals genetic material to not lose potential individuals that could guide the algorithm towards an optima. When the optimization problem consist of more than one objective function, such as in multi-objective optimization problems elitism is indirectly implemented in the algorithm and does not carry out its own step as seen for SOGA-problems, this concept of indirect implementation will be further explored in the Section 2.4.

2.4 Multi-Objective Optimization

As the name suggests, a multi-objective optimization problem (MOOP) is constructed of several objective functions that are to be optimized over a given search space. Compared to single-objective optimization (2.1) MOOPs can have several optima for a given problem, which requires the user to have prior knowledge about the relative importance of the objectives. The general MOOP with n decision variables and m objective variables can be formulated as follows.

$$\left.\begin{array}{ll} \min_{\mathbf{x}} & \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})]^\top \\ \text{s.t.} & g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \ldots, q \\ & h_j(\mathbf{x}) = 0, \quad j = 1, 2, \ldots, p \end{array}\right\} \tag{2.14}$$

Where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top \in \Omega \subseteq \mathbb{R}^n$ is an n-dimensional decision vector, with an n-dimensional decision space, Ω. $\mathbf{f}(\mathbf{x})$ represents the objective vector containing all the objective functions defined within the problem, $g_i(\mathbf{x})$ constitutes the inequality constraints, and h_j are the equality constraints [22]. Unlike a single-objective optimization problem (SOOP), the selection and ranking of individuals require a more complex method to establish the fittest candidates. To do so, the concept of Pareto-dominance and Pareto-optimality will be further explored in the following sections.

Pareto-Dominance

Based on (2.14) and given two feasible solutions $\mathbf{x}_A \wedge \mathbf{x}_B \in \Omega$, the solution $\mathbf{x}_A$ is said to dominate $\mathbf{x}_B$ if and only if it satisfies the following conditions.

$$(\forall i \in \{1, \ldots, m\} : f_i(\mathbf{x}_A) \leq f_i(\mathbf{x}_B)) \wedge (\exists j \in \{1, \ldots, m\} : f_j(\mathbf{x}_A) < f_j(\mathbf{x}_B)) . \tag{2.15}$$

If (2.15) is satisfied, the dominance is generally indicated in the following way:

$$\mathbf{x}_A \succ \mathbf{x}_B \equiv \mathbf{x}_A \text{ dominates } \mathbf{x}_B . \tag{2.16}$$

Pareto-Optimality

By utilizing (2.16) a solution $\mathbf{x}^* \in \Omega$, is referred to as the Pareto-optimal solution if the following condition holds:

$$\nexists\, \mathbf{x} \in \Omega : \mathbf{x} \succ \mathbf{x}^* . \tag{2.17}$$

If there exists a solution $\mathbf{x}^*$ that satisfies (2.17) this solution in non-dominated and objectively better than those solutions that do not satisfy this constraint. Having a method of objectively ranking the solutions, it is possible to construct a Pareto-optimal set that contains all Pareto-optimal solutions and is defined according to (2.18).

$$\mathbf{P}^* \triangleq \{\mathbf{x}^* | \nexists\, \mathbf{x} \in \Omega : \mathbf{x} \succ \mathbf{x}^*\} . \tag{2.18}$$

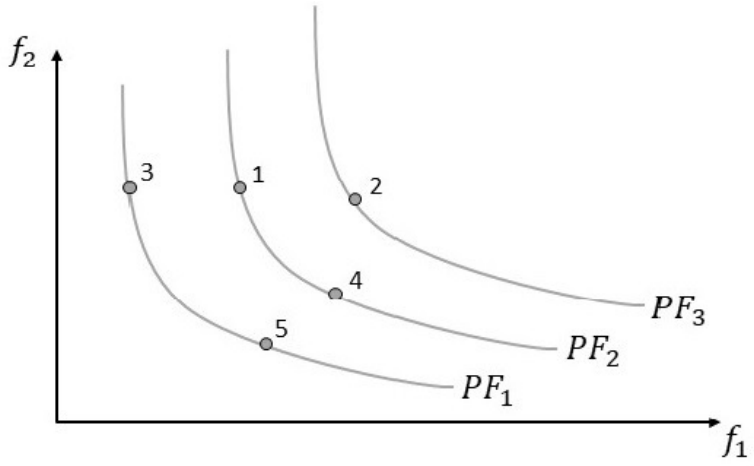


Figure 2.7: *A visualization of how the ranking of solutions can be ordered and placed into different front-levels, where PF_1 is the front containing the non-dominated solutions* (3) *and* (5)*.*

Iterating through the solutions given to a MOOP it is possible to construct different fronts by applying (2.18) and storing the solutions to separate front-levels as visualized in Figure 2.7.

The process of sorting the solutions to different fronts is constructed in an iterative way by initializing an empty set $\mathbf{PE}'$ and filtering the non-dominated solutions. The definition of a non-dominated solution is that it is not worse in any objective while superior in at least one, in accordance with (2.15). Furthermore, all available solutions are stored in the set $\mathbf{PE}$, which according to Figure 2.7 would be $\mathbf{PE} = \{1, 2, 3, 4, 5\}$. By constructing two loops, one selecting a specific solution ($i \in \mathbf{PE}$) and the other looping over all other solutions except for the selected one ($j \in \mathbf{PE} \wedge j \neq i$) a process for finding non-dominated solutions can be established. By storing the non-dominated solutions in $\mathbf{PE}'$ each complete iteration over all available solutions will store the front-level with an index represented of each complete iteration cycle. By removing the non-dominated solutions from $\mathbf{PE}$ when locating the next front-level the process can be carried out until $\mathbf{PE}$ is empty. Each complete set $\mathbf{PE}'$ represent a new front level $\mathbf{PF}_l$, with $l = 1, 2, \ldots L$, where L is the number of fronts, as seen in Figure 2.7. The first complete iteration cycle containing $\mathbf{PF}_1$ is commonly referred to as the Pareto-optimal front and is formally defined as the following:

$$\mathbf{PF}^* \triangleq \left\{ \mathbf{f}(\mathbf{x}) = \left[f_1\left(\mathbf{x}^*\right), f_2\left(\mathbf{x}^*\right), \ldots, f_m\left(\mathbf{x}^*\right) \right]^\top | \mathbf{x}^* \in \mathbf{P}^* \right\}. \tag{2.19}$$

Crowding Distance

The crowding distance is a measurement that entails the density of solutions surrounding a given solution, which is an important quantity to calculate when trying to preserve the diversity within a given population. The crowding distance is further calculated by firstly sorting all the solutions according to their objective values

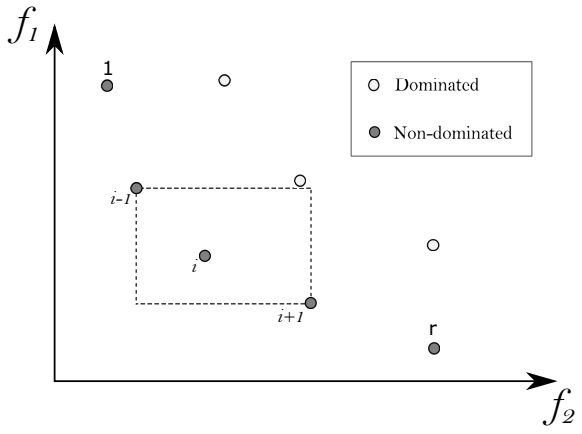


Figure 2.8: *Illustration of how the crowding distance is calculated for a solution i.*

within a given front $\mathbf{PF}_l$. So, for a front containing r solutions there will be m sorted lists ($\mathbf{I}^i, \quad i = 1, \ldots, m$) each containing all r solutions in ascending objective value with respect to f_i. With the sorting performed each solution is assigned a crowding distance value of zero, $cd_k = 0, \quad k = 1, 2, \ldots, r$ and the boundary solutions in each sorted list, $cd_{I_1^i}$, $cd_{I_r^i}$ are assigned a crowding distance value of infinity [23]. In Figure 2.8 these two solutions are the non-dominated solutions marked 1 and r which are given a crowding distance value of infinity since they are at the far edges of the solution space thus contributing to the preservation of diversity. The crowding distance is calculated for each possible solutions by iterating over all the objectives as presented in (2.20).

$$cd_k = \sum_{i=1}^{m} \sum_{j=2}^{r-1} \frac{I_{j+1}^i - I_{j-1}^i}{I_r^i - I_1^i} \,. \tag{2.20}$$

By introducing a crowded comparison operator ($\prec_n$) solutions can now be compared in regards to two aspects, namely:

1. Non-domination rank (i_{rank})
2. Local crowding distance (cd_k)

The first criteria separates solutions over different fronts ($\mathbf{PF}_l$) based on non-domination which in turn means that solutions within fronts of the lowest value of l are preferred. The second criteria separates solutions within the same front based on their crowding distance value. Furthermore, the definition for the crowded comparison operator is defined as in [24].

$$a \prec_n b \,\text{if}\, (a_{\text{rank}} < b_{\text{rank}}) \vee ((a_{\text{rank}} = b_{\text{rank}}) \wedge (a_{\text{cd}} > b_{\text{cd}})) \,. \tag{2.21}$$

Between two solutions, a and b with different non-domination ranks the lower rank is preferred. And for two solutions belonging to the same front and thus having an equal rank the solution with the larger crowding distance value is preferred as this will increase the diversity within the population.

NSGA-II

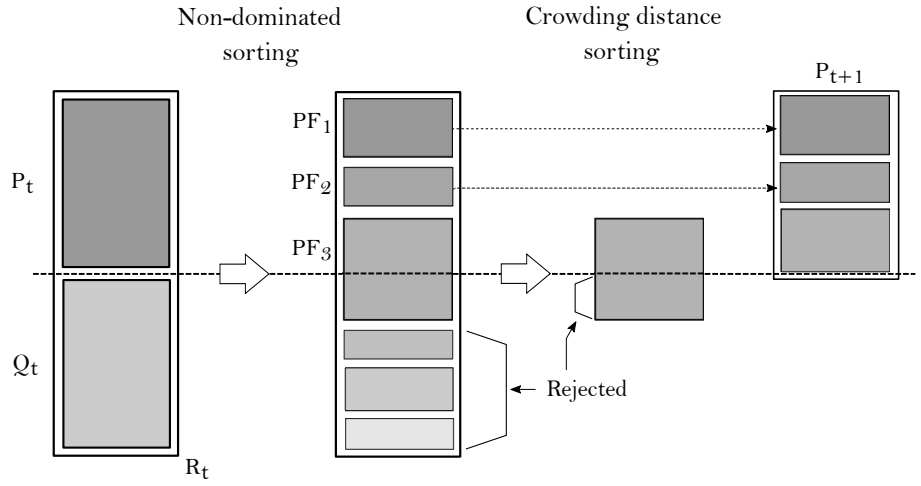


Figure 2.9: *A basic outline of the steps involved in NSGA-II, which displays the formation of $R_t = P_t \cup Q_t$, that undergoes non-dominated sorting and crowding distance sorting to form a new parent population P_{t+1}.*

A widely used algorithm to solve MOOPs is the Non-Dominated Sorting Genetic Algorithm II (NSGA - II), developed by Deb et al. [24]. This algorithm is used to find multiple Pareto-optimal solutions in a MOOP and demonstrates three key characteristics stated in [25]:

1. Usage of an elitist principle,
2. Focuses on non-dominated solutions, and
3. Utilizes an explicit diversity preservation mechanism, namely; crowding distance

For the first iteration, $t = 0$, a parent population P_t is initialized of size N and similar to SOGA problems, undergoes operators such as selection and mutation to create a children population Q_t of size N. For $t \geq 1$ the algorithm follows the steps shown in Figure 2.9. For each generation, the current population is of size $2N$, consisting of the current parent population and its children population, forming the union $R_t = P_t \cup Q_t$. Once R_t has been constructed, the population is sorted according to non-domination, forming different fronts containing individuals based on their fitness and level of domination, this process is illustrated in Figure 2.7. The new parent population P_{t+1} is formed by adding individuals from the first front and moving up in front-levels until its size exceeds N. As seen in Figure 2.9 all individuals from the front PF_1 and PF_2 are transferred to the new parent populations. However, since not all individuals in front PF_3 can be selected for P_{t+1}, crowding distance sorting is performed, and only those individuals that preserve diversity the best are chosen. Furthermore, the new parent population is used for selection, crossover and mutation to form Q_{t+1} and so the process repeats. Therefore, the total complexity of one generation of the algorithm can be divided into three steps[26]:

1. Non-dominated sorting $O(MN^2)$

2. Crowding distance assignment $O(MN \log N)$
3. Sorting on $\prec_n$ is $O(2N \log(2N))$.

Which establishes a total complexity of $O(MN^2)$.

2.5 Normalization of MOGA-Values

When working with MOGA problems, it is common that the data generated for each individual objective function do not share a common unit of measurement. To simplify the process of evaluation, a form of robust normalization is commonly conducted where every objective value takes a value between 0, and 1 in a so-called min-max normalization [27].

$$f_i^{norm}(\mathbf{x}) = \frac{f_i(\mathbf{x}) - obj_i^{\circ}}{obj_i^{\max} - obj_i^{\circ}}, \quad i \in [1, m] . \tag{2.22}$$

In equation (2.22) m is the number of objective functions and obj_i° represents the utopia point, which is equivalent to where each objective function is in its most optimized state. Generally, this point can also be approximated based on engineering intuition and is then referred to as an aspiration point or a target point. Furthermore, $obj_i^{\max}$ is the maximum value each objective function can experience and similar to obj_i° it can also be approximated based on engineering intuition. When each function value $f_i(\mathbf{x})$ is evaluated according to the normalization equation, a normalized value $f_i^{norm}(\mathbf{x})$ is produced.

2.6 Scaffolding

Throughout this thesis, a common building pattern will be used on assembled scaffolds, which is common practice and follows the general guidelines and safety measures taken by the Swedish Work Environment Authority and SiS [2], [3]. This pattern can be seen in Figure 2.10. Because of needed simplification in how materials are counted going onward throughout this thesis, one material is the equivalent to one whole compartment in width with its respective height. The scaffold displayed in the figure is consisting of two materials. This is done as total scaffolding components used scale directly with the amount of compartments.

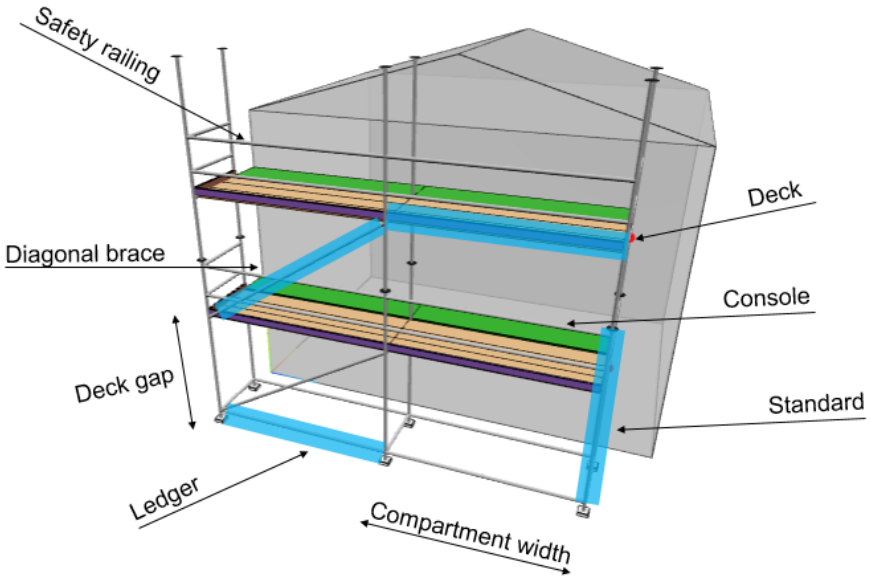


Figure 2.10: *Common scaffolding structure with commonly used names for scaffolding dimensions and components.*

The material components used for constructing a scaffolding are diagonal brace, standards, ledgers and consoles, the diagonal brace supports between compartments diagonally while ledger supports compartments horizontally and the standards are vertical supports. The consoles are optional modules for a scaffold to provide wall access for a scaffold when the gap between wall and scaffold is large. There are standards and recommendations to follow when building a scaffold and the dimensions of its properties. One of those that will be used in optimization is the gap between a scaffold and its corresponding wall face, this should be as small as practicable and not exceed 0.3 meters according to Section 31 of AFS [2]. If the gap is any bigger than 0.3 meters one must use additional modules in the form of consoles to fill this space as shown in figure 2.10.

3
Methodology

In the methodology chapter two main segments will be presented involving the milestones for the algorithmic construction, and the approach for evaluating the algorithmic performance, both included in the first segment and will be conducted with an unconstrained inventory. All developed tests for this first segment are supported by ScaffCalcs visualization tools and can therefore be processed and evaluated. The second segment entails algorithmic logic for unconventional buildings and how to handle wall sequencing, and a constrained material inventory. The constrained inventory will not be implemented in any visual or user test cases, but solely presented in terms of a feature, remaining attributes in the second segment will not be further evaluated in the result section as they are not supported on ScaffCalcs web application.

3.1 Milestones for Algorithmic Construction

The structure and goal of this thesis has been to develop an algorithm to solve an entire scaffolding optimization problem by successfully developed solutions to all of its milestones. By following a timeline fashion these milestones were evaluated to the milestones' respective difficulty, seen from the writers point of view. The milestones that were evaluated along the thesis and further presented in the result section are;

1. Length optimization
2. Length and material optimization
3. Connecting scaffolds around multiple walls
4. Optimizing scaffolds on gable walls

For visualizing purposes all images provided in this section were manually created by the writers and are no reflection of the algorithmic performance. Furthermore, walls 1-4 are presented in a counterclockwise order, implying that wall 1 is always seen in the front view of each images containing the buildings gable unless otherwise stated.

3.1.1 Milestone 1 - Length Optimization

The first milestone to consider is to optimize a scaffolding and its length in regards to a given wall facade. The scaffolding can be built from materials consisting of specific available lengths, mostly depending on which supplier is chosen with examples shown

in Table 3.1. The objective is to minimize the length difference of the sum of chosen materials and the wall length. For visualization of scaffolds, Layher material was used.

Table 3.1: *Available ledge lengths, in meters, for different material suppliers.*

Layher	Haki	Pluseight
0.73	0.7	0.5
1.09	0.77	0.7
1.40	1.05	1
1.57	1.25	1.25
2.07	1.655	1.5
2.57	2.05	1.75
3.07	2.55	2
4.14	3.05	2.5
		3
		3.5

By utilizing a SOGA and solely optimizing on a given length the goal is to generate a scaffolding structure similar to that given in Figure 3.1.

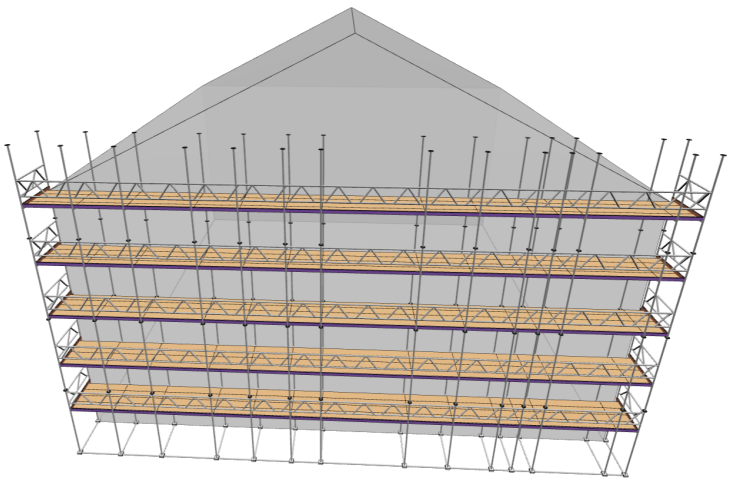


Figure 3.1: *Milestone 1, displaying an example of a successful solution to a SOGA problem where length is the only objective function.*

3.1.2 Milestone 2 - Length and Material Optimization

A clear and natural extension to the first milestone is for the generated solution to use less amount of material, which in turn means to use wider compartments in forms of longer ledge lengths. To accomplish this the algorithm will become a MOGA with

two objective functions, namely length, and material. A successful implementation would generate a figure similar to that displayed in Figure 3.2, which displays a clear difference in the amount of material whilst still maintaining the length compared to that in Figure 3.1.

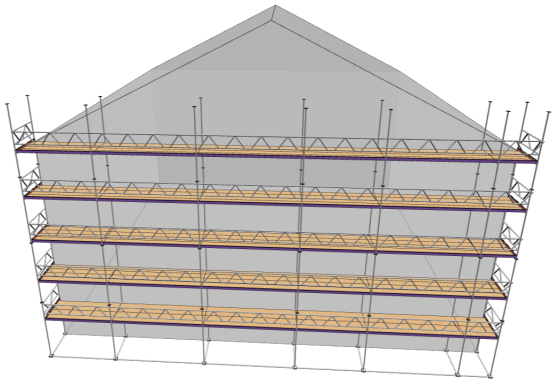
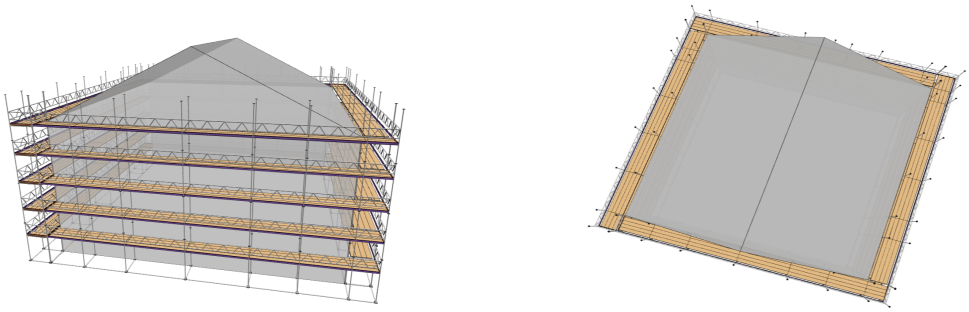


Figure 3.2: *Milestone 2 - displaying an example of a successful solution from a MOGA with two objective functions, being; length and material optimization.*

3.1.3 Milestone 3 - Connecting Scaffolds Around Multiple Walls

The next milestone is for the algorithm to be able to cover multiple connected walls and have respective scaffolding around these. This leads to some manipulation of targeted lengths for optimization as well as certain constraints further explained in Section 3.3. A successful implementation surrounding a given facade can be seen in Figure 3.3. Due to the symmetric nature of wall 1 and 3, the algorithm is intended to be run on wall 1, from which the result is directly mirrored on wall 3. Walls 2 and 4 differ in the regard that wall 2 and 3 share a fixed size corner piece that connects them which wall 4 and wall 1 does not. Due to wall 4 and wall 1 not necessarily sharing a corner piece the last compartment of wall 4 should extend just past the deck width of scaffolding for wall 1 which can cause these walls to differ with respect to their connection to the following wall in sequence and would mean that wall 4 can use one less compartment.



(a) *Front view* **(b)** *Top view*

Figure 3.3: *Milestone 3, displays a successful example of connecting multiple scaffolding compartments around multiple wall facades.*

3.1.4 Milestone 4 - Optimizing Scaffolds on Gable Walls

To improve and increase the usage of the algorithm, more functionality is needed. Additional access for the gable roof facade present on two out of four walls for a square building is needed. Furthermore the MOGA will only be run on wall 1, whereas the second wall and fourth wall in this case uses linear optimization programming to be solved. Due to the absence of a gable along this wall and thus no ordered optimization problem, linear programming provides a faster and more consistently well performing solution in comparison to using the MOGA on these walls. To grant this additional roof access, the height needed for each compartment within the scaffold to its respective roof edge must be calculated and additional scaffolding stories must be added accordingly from the extra height of an added roof. An example of granting sufficient access can be seen in Figure 3.4.

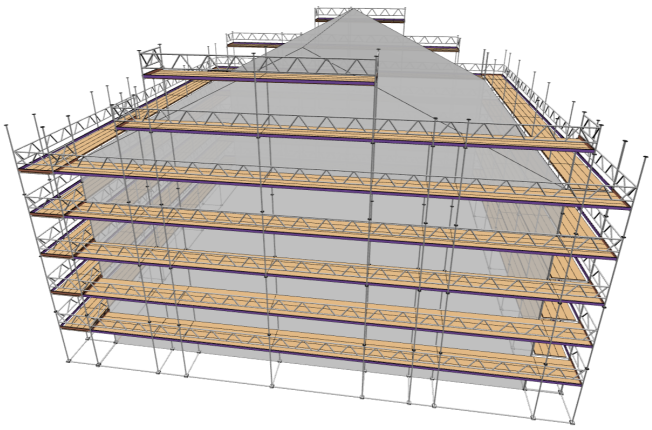


Figure 3.4: *Milestone 4, displays an example of successful implementation of scaffolds to allow for roof access.*

Roof types in the current version of live software are restricted to gable roofs. The development of the algorithm focused on implementation of gable roofing with additional option to grant access to tilted roofing. Gable roofing will always have a symmetrical look over the wall facade with midpoint corresponding to the walls' midpoint, while tilted roofing either ascends or descends in height, starting on the left side from the walls' front view.

When a roof facade is at least two meters higher than the wall facade at any given point, the scaffold should give access to this area above. This is done by providing an additional story of scaffolding stretching between the roof ends where the height difference is two meters or more. This extra scaffolding adds enough stories so that there at no point is a roof edge which is more than 2 meters above the highest scaffolding compartment floor. The scaffolding should also be extending past the outer edges of the roofing it is granting access to, as to fully cover the additional construction area. Figure 3.5 shows how additional scaffolding stories would be constructed to allow access to the roof.

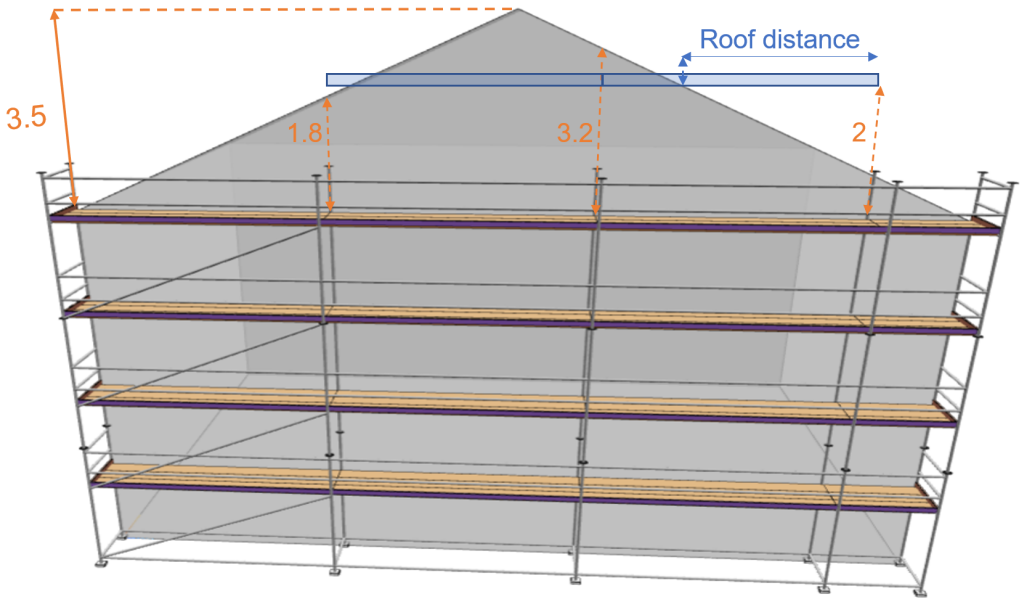
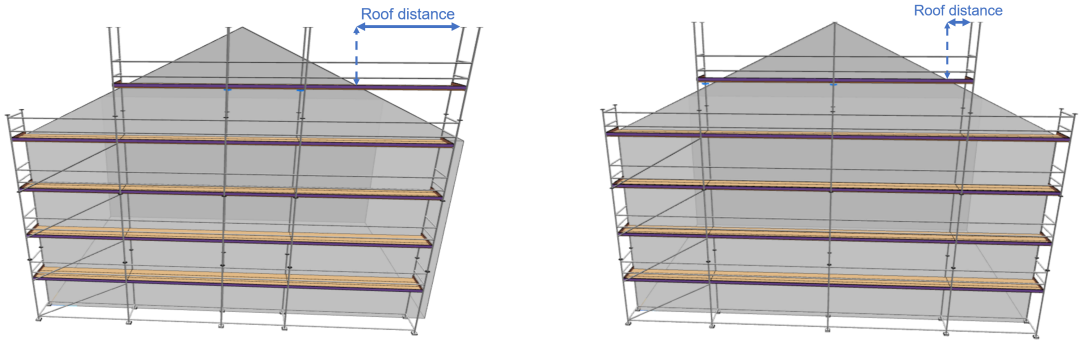


Figure 3.5: *Milestone 4, visualization of additional roof scaffolding access with given heights.*

It can be seen that the rightmost extra story of scaffolding is extending quite a bit past the roof edge. This is the distance from the roof which should be minimized to reduce forces from any oncoming wind. The exposed area of scaffolding extended past roof edges and its area size directly influences structural forces from wind calculations [3]. Additionally, having the anchoring point closer to the wall increases the stability of the scaffold. The first extra story in this example case, displayed as a blue box is set up two meters above previous story to guarantee head room clearance in accordance to minimum clearance of 1.9 meters by SiS [3]. There are in general two things that must be taken into account for the algorithm to solve this effectively. Firstly, check if any additional stories must be constructed on the specific compartment width. Secondly, if any additional story is needed then a new objective function must be defined in order to construct this new scaffolding structure efficiently. This was done by introducing another objective, minimizing the distance on the scaffold compartments extended beyond the roof edge. This constructs a more optimized and safe scaffolding structure with reduced loads with respect to wind forces.

In Figure 3.6, an example case of two possible solutions is displayed with one being a worse option over the other with respect to the total distance to the roof edge.



(a) *Poor roof scaffold construction* **(b)** *optimized roof scaffold construction*

Figure 3.6: *Milestone 4, comparison between an optimized and a poor scaffold solution of minimizing distance to the roof edges.*

3.2 Evaluation of Algorithmic Performance

For the evaluation of algorithmic performance three test cases were utilized when identifying an optimal number of generations for a given facade input. All three cases was buildings consisting of four walls and a saddle roof, which is the form of input at the time of writing supported by ScaffCalcs visualization tool. Furthermore, data logged from real users on ScaffCalc was utilized and compared to the algorithm, as the goal for the algorithm is to generate a similar or better scaffold. Metrics such as the amount of material and generation time was compared to what the algorithm produced. Additionally, a quantitative test was constructed where two different test cases of buildings was compared to the output of the algorithm. Lastly, any time constraints on the algorithm were removed and fully converged results were evaluated by comparing it to results generated by users and algorithmic output under time constraint. All tests was conducted on a computer with an Intel Core i7-1065G7 processor running at 1.3GHz, 4 cores.

3.2.1 Locate Optimal Number of Generations

Since the algorithm is time constrained to ≤ 2 seconds it is vital to utilize the correct amount of generations to optimize performance. To add algorithmic flexibility, three test cases, each with different input sizes were evaluated over 50 separate runs, for 30, 50, and 75 in population. These three test cases and their dimensions were based on data taken from ScaffCalc calculations, using a percentage distribution of total scaffolding length surrounding a building. From this distribution, a threshold for an upper limit with respect to a scaffolds total length was chosen for a small, medium and large scaffold to represent the distribution in three different sized test cases and their respective building sizes, as seen in Figure 3.7.

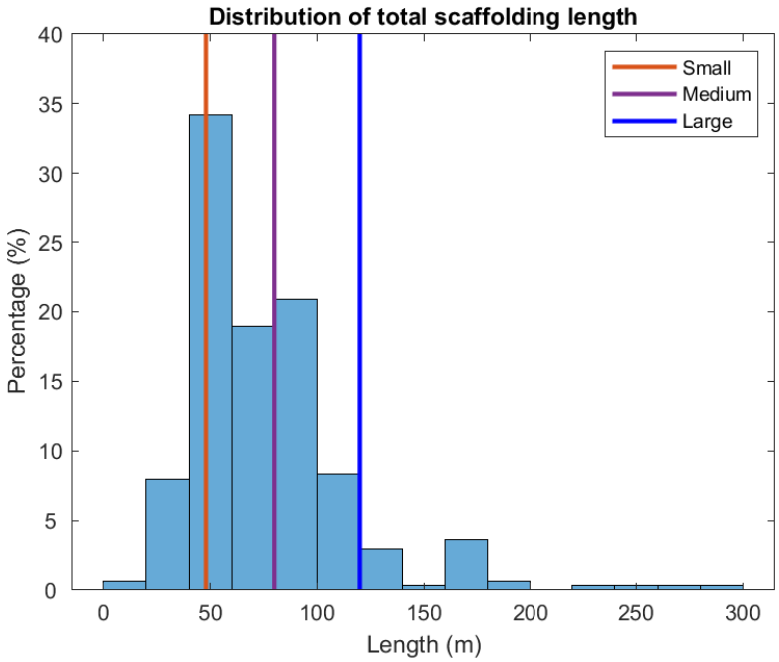


Figure 3.7: *Distribution of total scaffold length surrounding four walls from ScaffCalcs database with marked lines for small, medium and large scaffolding thresholds.*

The goal was to find a time vs generations correlation and a weighted average of the objectives given at each generation for different populations and building sizes. In Table 3.2 the metrics for each test case is provided, along with a reference building showcased in Figure 3.8.

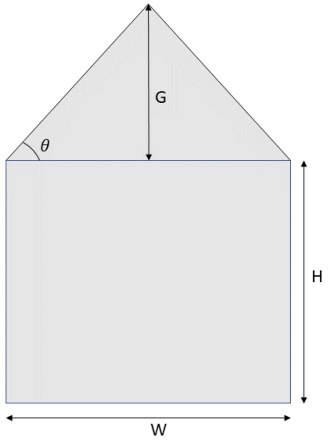


Figure 3.8: *Reference model of wall 1 used for all three test cases when locating the optimal number of generations.*

Building	θ	G	W	H
Small	$\arctan(0.5)$	3	12	5
Medium	$\arctan(0.5)$	5	20	8
Large	$\arctan(0.5)$	7.5	30	12

Table 3.2: *Metrics presented in meters for all lengths, given for small, medium, and large building used for locating the optimal number of test cases.*

Furthermore, each building was of square dimensions implying that each test case had a dimension of $W \times W$. By choosing the metrics demonstrated in Table 3.2 the buildings was be comprised of 1, 2, and 3 roof compartments along the height of G as seen in Figure 3.8. Lastly, all data was normalized according to equation (2.22), to be properly compared. In Table 3.3 the maximum, and Utopian points

for each objective and test case is introduced. These three presented objectives are wall distance which is how much the scaffold extends past the building corner with respect to any additional padding specified by a user in meters, materials which is the quantity of scaffolding compartments used in the scaffolding and lastly roof distance which is the total sum of the length that scaffolding compartments extends past a roof edge in meters.

Table 3.3: *Displays the Utopian values as well as the maximum values used when normalizing the data for each test case for wall 1.*

Test Case	obj_1° (m)	obj_2° (qty)	obj_3° (m)	$obj_1^{\max}$ (m)	$obj_2^{\max}$ (qty)	$obj_3^{\max}$(m)
Small	0.15	5	0	1	17	4
Medium	0.15	8	0	1	28	8
Large	0.15	12	0	1	42	12

3.2.2 Visual Comparison Vs User-Built Scaffolds

Logged data containing user-built scaffolds around four walls were evaluated on construction time and visual discrepancies to similar results generated by the algorithm. Due to limitations in data logging, the true dimensions for the buildings had not been stored, which forced the writers to replicate the length of the walls to the best of their ability. Since the lengths of all compartments and images of the user-built scaffolds were stored, similar replicas were recreated and used solely for visual comparisons. The approximate metrics for the two recreated buildings are presented in Table 3.4, assuming a reference model according to Figure 3.9.

Table 3.4: *Metrics for object O1, and O2 where all lengths are given in meters.*

Object	θ	G	W	L	H	Available Lengths
O1	$\arctan\left(\frac{3}{5.5}\right)$	3	11	10	5	Layher[1] (3.1)
O2	$\arctan\left(\frac{1}{2}\right)$	5	20	35	5	Haki (3.1)

[1]All lengths except 4.14 are included

Furthermore, each object was subjected through the algorithm for 10 independent runs, and results such as material used and time taken was presented for both O1 and O2.

3.2.3 Quantitative Comparison of Scaffolding Generation

Members of ScaffCalc were given two test cases, classified as B1 and B2, consisting of one smaller (B1) and one larger (B2) building. The task was for each individually to create scaffolds around each test case to the best of their ability. A specific restriction provided to each user was that each roof compartment had to protrude horizontally from the roof edges similar to the algorithm. The dimensions and metrics of B1 and B2 are found in Figure 3.9 and Table 3.5.

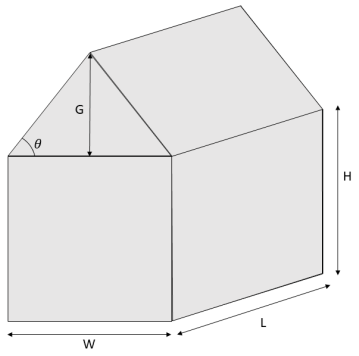


Figure 3.9: *Reference model used for both test cases when members from ScaffCalc are to construct scaffolds.*

Test Case	θ	G	W	L	H
B1	$\arctan\left(\frac{5}{6.5}\right)$	5	13	11	8
B2	$\arctan\left(\frac{7}{12.5}\right)$	7	25	25	12

Table 3.5: *Metrics for test case B1, and B2 where all lengths are given in meters.*

Based on the results of Section 3.2.1 a suitable number of generations was chosen that is applicable to the algorithmic runs on building dimensions for B1 and B2. Furthermore, the algorithm was run for 100 independent runs, each under the same conditions, and the output of each run will be compared with the results generated from ScaffCalcs members modeling. All previously states objectives, as well as time, will be presented and evaluated.

3.2.4 Time-Unconstrained Algorithmic Performance

The same test cases as presented in section 3.2.3 was used to evaluate the performance of the algorithm while not following any constraint on run-time. To execute the algorithm under no time constraint, the algorithms' stopping criterion was set

to compare the Pareto front from generations in a 1000 interval. If the Pareto front has not changed and thus has not found a better solution or improved objective values within 1000 generations, the algorithm was stopped. This stopping criterion was checked for in every 100 generations. By using a convergence-based stopping criterion, the algorithm should find either a local optima or a global optima. A total of 100 independent runs were carried out using a time-unconstrained stopping criterion. The results generated from the time-unconstrained algorithm was compared to those generated in Section 3.2.3.

3.3 Algorithmic Logic for 2D Buildings

Given an input building, there are several key factors to consider when generating scaffolds around its selected walls. Firstly, the scaffold is generated in a globally counterclockwise order around the building. To give correct input for the algorithm, some pre-processing has to be done between the user input and algorithm. The selected walls have to be evaluated to see if they share a corner. If a corner is shared, it must be determined if the corner is facing inwards or outwards, as this influences the given length constraint. Another element to detect for each wall is whether there is any roofing facade that should be accessed. This leads to further constraints and optimization regarding its appearance.

3.3.1 Wall Sequence Processing

If two walls are sharing an outward corner the algorithm input for the first wall should be constrained to guarantee a suggestion going past the corner as to guarantee an interlocking between the two walls and their respective scaffolds as seen in Figure 3.10. The corner piece shown in the figure is not part of the scaffolding length objective as this is a fixed size of scaffolding depending on compartment depth that interlocks two scaffolds around corners. It should also consider any possible inputs from the user on the distance between scaffold placement and its wall defined as wall distance, L_{wall}, as this leads to added or subtracted distance depending on the corner configuration and its constraint. The input is therefore given according to equation (3.1) for an outwards corner between walls.

$$\begin{aligned} L_{\text{total}} &= L_{\text{wall}} + L_{\text{distance}} \\ L_{\text{scaffold}} &\geq L_{\text{total}} \end{aligned} \tag{3.1}$$

For an outwards corner there are four length variables present which includes L_{wall} which is the distance of the wall, L_{distance} which is the length between the scaffold and its wall. L_{scaffold} is the total scaffolding length which must be greater or equal to L_{total}. This constraint is dependent on the building geometry and the interlocking between two walls.

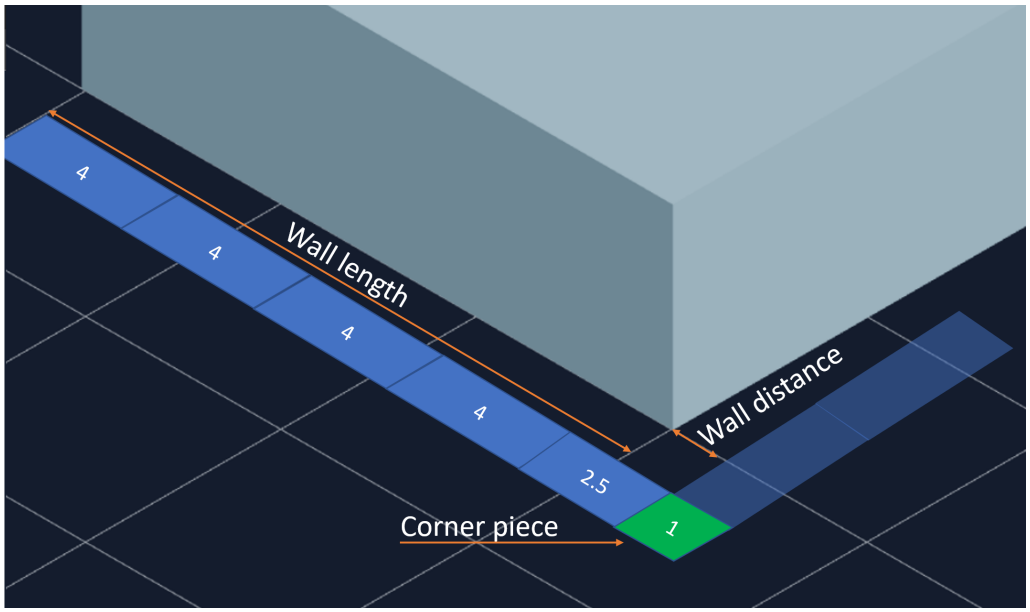


Figure 3.10: *Construction geometry a scaffold around two walls sharing an outwards corner with wall length as W_{wall} and wall distance as $L_{distance}$ and corner piece dimensions as L_{corner}.*

If two walls were to share an inwards corner the total scaffold length of the first wall should account for this next scaffold sequence regarding optimization. It is influenced by wall distance and a corner piece as well as constrained on its maximum length as seen in Figure 3.11. If it is not the first wall in a sequence then the wall distance can be neglected from subtraction as it evens out over the two corners as defined in equation (3.2).

$$\begin{aligned} L_{\text{total}} &= L_{\text{wall}} - L_{\text{corner}} \\ L_{\text{scaffold}} &\leq L_{\text{total}} \end{aligned} \tag{3.2}$$

Here, the added distance L_{corner} is the size of the square scaffolding piece placed in the connection of two walls.

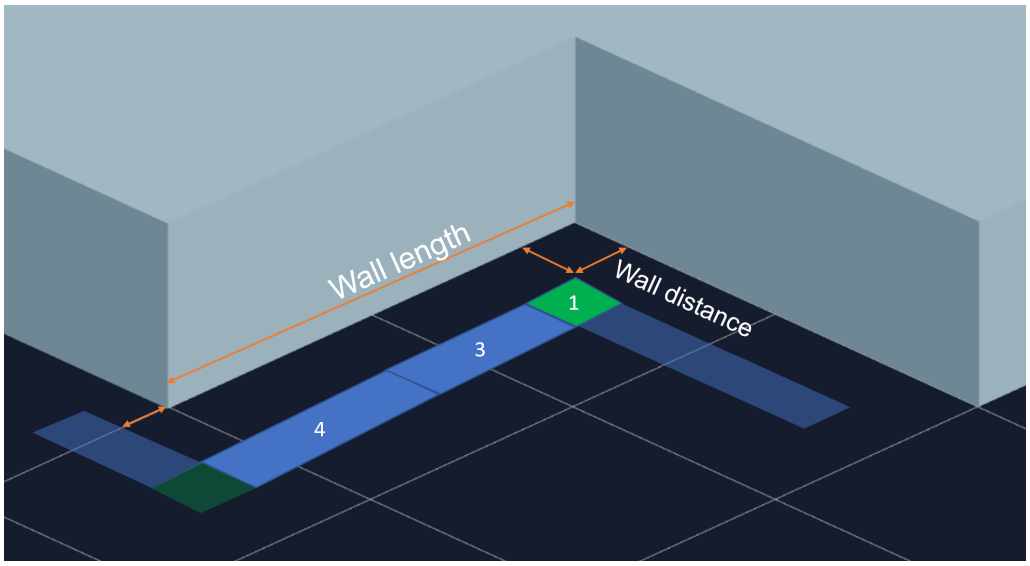


Figure 3.11: *Scaffolding sharing inwards corner with wall length as W_{wall} and wall distance as $L_{distance}$*

However, if it is the first wall for a given sequence the wall distance must also be subtracted, as presented in equation (3.3).

$$\begin{aligned} L_{\text{total}} &= L_{\text{wall}} - L_{\text{distance}} - L_{\text{corner}} \\ L_{\text{scaffold}} &\leq L_{\text{total}} \end{aligned} \tag{3.3}$$

The second wall in sequence sharing this corner would thereafter also subtract the depth of the scaffolding from its total scaffolding length and check for connecting walls and if the next corner is inwards or outwards. A longer sequence of wall connections is displayed in Figure 3.12 with its respective L_{scaffold}, in this example $L_{\text{scaffold}} = L_{\text{total}}$ from previous examples.

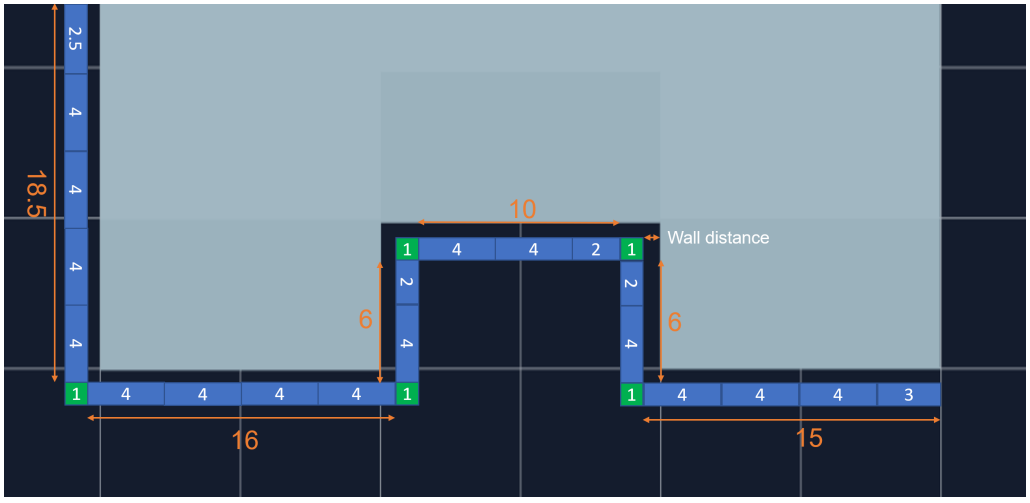


Figure 3.12: *Displays how wall sequencing would be performed depending on inwards/outwards corners.*

From this figure two special cases are displayed where there are walls connected to double outwards and double inwards corners with their respective L_{total} displayed in (3.4), (3.5). These walls are wall number 2 and 4 counted from the left most wall with a length of 18,5. When more walls are sequenced it is needed to account for the previous walls and their respective miss match in regards to L_{scaffold} and L_{total} when accounting for total length in next wall. An example of matching total length for wall 3 is shown in equation 3.6 which depends on wall 1 and its miss match labeled L_{diff1}.

$$\begin{aligned} L_{\text{total2}} &= L_{\text{wall2}} + 2L_{\text{distance}} \\ L_{\text{scaffold2}} &\geq L_{\text{total2}} \end{aligned} \tag{3.4}$$

$$\begin{aligned} L_{\text{total4}} &= L_{\text{wall4}} - 2L_{\text{distance}} - 2L_{\text{corner}} \\ L_{\text{scaffold4}} &\leq L_{\text{total4}} \end{aligned} \tag{3.5}$$

$$\begin{aligned} L_{\text{diff1}} &= L_{\text{scaffold1}} - L_{\text{total1}} \\ L_{\text{total3}} &= L_{\text{wall3}} + L_{\text{diff1}} - L_{\text{corner}} \\ L_{\text{scaffold3}} &\leq L_{\text{total3}} \end{aligned} \tag{3.6}$$

3.3.2 Constrained Material Inventory

To add further algorithmic flexibility and take into account the inventories of the user, the algorithm should use no more than the specified available material for its suggested solution. This implies that the optimization is constrained to the amount of material available according to equation (3.7) where M_{used} is the material used for the solution and M_{inv} is the material available in the inventory.

$$M_{\text{used}} \leq M_{\text{inv}} \tag{3.7}$$

This test aims to showcase how well the algorithm works when restricting its search space. The results presents the inventory before and after the algorithm is run to be compared to the results of the unconstrained run, where both are evaluated for test case B2, see Table 3.5. Furthermore, all Layher lengths from Table 3.1 will be available for both runs.

4
Algorithm

This section covers the usage of genetic algorithms with custom objective functions and constraints tailored for the problem at hand in combination with integer programming when an ordered optimization is not needed to increase performance. For the algorithm, this section covers all the objectives and constraints present in the final iteration. It further explains functions for tracking a user's inventory with a supply of materials and applying these in the form of constraints.

4.1 Algorithm Frameworks

The algorithmic focus is to generate a scaffolding solution with respect to optimizing several objectives, which is done using two different frameworks. The first is Pymoo, which is used for the genetic algorithm and its minimization of objectives [28]. The second framework is PuLP, which is used for integer programming, where one of its solvers is used to minimize the optimization objectives [29]. Integer programming is very close to linear programming, with the sole exception that optimization variables are constrained to be integer values. The variables are constrained to integer values because a scaffold cannot use a fraction of the length of the material.

4.1.1 Pymoo

The Pymoo framework offers many single- and multi-objective optimization algorithms, with the majority of available algorithms being stochastic optimization algorithms through a Python framework. Pymoo specifically offers a lot of alternatives for a genetic algorithm when solving both a single- or multi-objective optimization problem. The algorithm developed throughout this thesis utilizes the minimization methods of the NSGA-II algorithm, including objective minimization and ranking through the crowding distance with Pareto fronts as described in Section 2.4. The evolutionary methods described in Section 2.3 used within the NSGA-II algorithm had to be modified and adjusted to fit the problem and to be in accordance with the objective of this thesis. The NSGA-II algorithm is used for the multi-objective scaffold optimization regarding three objectives, including wall fit, amount of material, and roof distance. The GA is used for this over other optimization algorithms, as the roof distance objective depends on the arrangement of ledge lengths order for evaluation of its fitness.

4.1.2 PuLP

PuLP is a linear programming solver capable of solving integer programming problems. The algorithm utilizes PuLP default solver CBC to solve the bi-objective problem regarding wall fit and material amount when a roof facade is not present on a specific wall as then a certain arrangement of the variables is not needed to determine the fitness of said solution. Using the PuLPs' LP solver for these instances, the algorithm can find a better solution faster than a genetic algorithm, which both reduces runtime and brings consistent performance on walls without a roof gable area.

4.2 Minimization problem

The multi-objective minimization problem is defined as in (4.1) where each objective function is explained individually in Section 4.2.1 and constraints in Section 4.2.2.

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{f(x)} = [f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x})]^T \\ \text{s.t.} \quad & g_i(x) \geq 0, \quad i = 1, 2, \ldots, 6 + q + n \\ & x_i \in \mathbb{R}, \quad i = 1, 2, \ldots, n \end{aligned} \tag{4.1}$$

4.2.1 Objective functions

For the MOGA there are three objective functions which are to be minimized. These functions calculate the scaffold length compared to wall length, the amount of compartments used and how much a scaffold extends past roof edges. The first objective function is to minimize the total scaffolding length with respect to the length of the wall which is defined in (4.2).

$$f_1(\mathbf{x}) = \left| \left(\sum_{i=1}^{n} x_i \right) - L_{\text{wall}} \right|, \quad L_{wall} > 0\,. \tag{4.2}$$

Here x_i is each gene within the chromosome $\mathbf{x}$ of length n containing a set of suggested compartment lengths, and L_{wall} is the total length of the wall. The second objective is with regards to the material and calculates the total amount of compartments used within a scaffold design as seen in (4.3).

$$f_2(\mathbf{x}) = n\,. \tag{4.3}$$

For the third and final objective function the total extended scaffolding compartment length past the roof edge must be calculated for two cases as defined in (4.4) and (4.5) with accompanying figures.

First case

Each roof is divided into a left and right side of roof intersections between roof edge and scaffolding compartment floors. L_i, and H_i are the extended roof distances for

each compartment of either left or right side triangle, V_i is the global coordinate for the left-most point in compartment i for either side, θ the roof-angle and Z_{Li} or Z_{Ri} is the global intersection coordinate between compartment i and the slanted roof on left or right side. For the specific case shown in Figure 4.1 the variables h, n take the values of $4, 7$, where h is the index of the compartment that goes through the midway coordinate of the roof and n is the index of the last compartment.

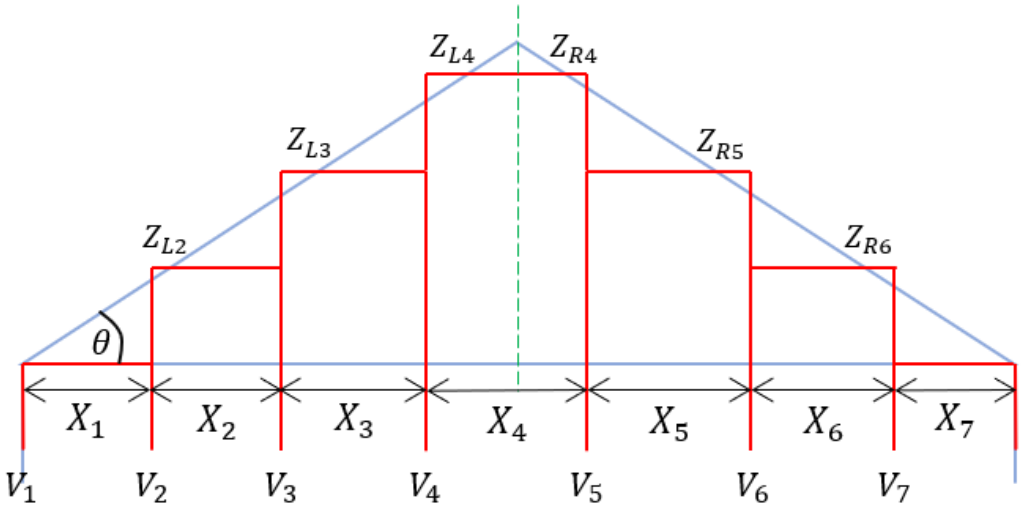


Figure 4.1: *First instance of compartment geometry for objective 3.*

$$f_3(\mathbf{x}) = \begin{cases} \sum_{i=1}^{h} L_i(\mathbf{x}) \Rightarrow L_i & = \begin{cases} -V_i(\mathbf{x}_{1:i}) + Z_{Li}(\mathbf{x}_{1:i}, \theta), & \exists Z_{Li} \\ 0, & \nexists Z_{Li} \end{cases} \\ \sum_{i=h}^{n} H_i(\mathbf{x}) \Rightarrow H_i & = \begin{cases} V_i(\mathbf{x}_{1:i}) - Z_{Ri}(\mathbf{x}_{1:i}, \theta) + x_i, & \exists Z_{Ri} \\ 0, & \nexists Z_{Ri} \end{cases} \end{cases} \tag{4.4}$$

Second case

In the second case of objective function f_3 the difference is that the middle part of compartments is split up so that no compartment shares both left and right side intersection of Z and therefore is not used twice. This means each compartment should only be used once for their respective left or right side intersection of roof through compartment floor to calculate f_3 as seen in (4.5). For the second case shown in Figure 4.2, h, n takes the values of 4 and 8, where h is the index of the compartment with respective V at the left side of the roofs midway coordinate.

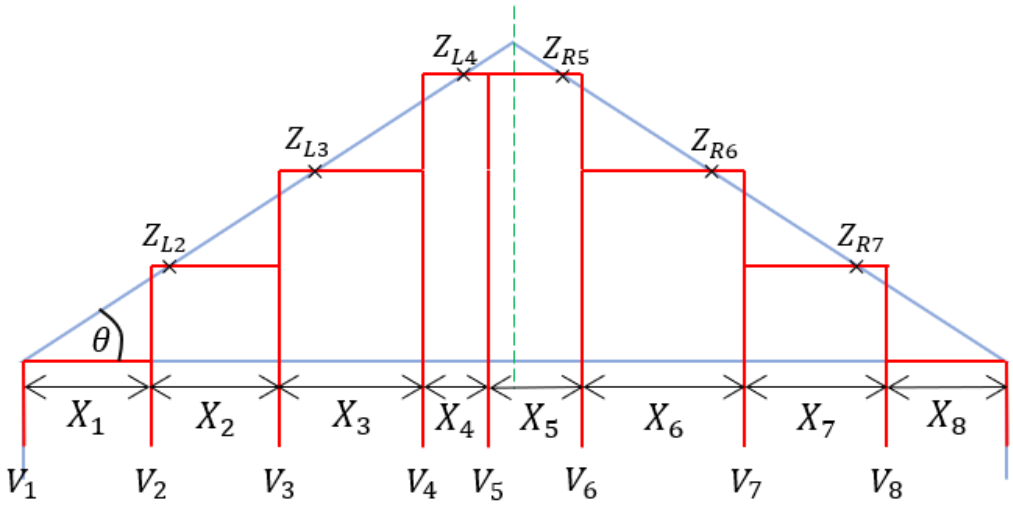


Figure 4.2: *Second instance of compartment geometry for objective 3.*

$$f_3(\mathbf{x}) = \begin{cases} \sum_{i=1}^{h} L_i(\mathbf{x}) \Rightarrow L_i & = \begin{cases} -V_i(\mathbf{x}_{1:i}) + Z_{Li}(\mathbf{x}_{1:i}, \theta), & \exists Z_{Li} \\ 0, & \nexists Z_{Li} \end{cases} \\ \sum_{i=h+1}^{n} H_i(\mathbf{x}) \Rightarrow H_i & = \begin{cases} V_i(\mathbf{x}_{1:i}) - Z_{Ri}(\mathbf{x}_{1:i}, \theta) + x_i, & \exists Z_{Ri} \\ 0, & \nexists Z_{Ri} \end{cases} \end{cases} \tag{4.5}$$

4.2.2 Constraints

For the algorithm to produce feasible solutions, it must meet some constraints. These constraints will be connected with the available material, the length of the structure, the roof scaffolding, and the specifics to grant certain access points. The first constraint set is to guarantee that a scaffold will either extend past a wall to enable scaffolding to round corners if required or stop just short as to not collide with an upcoming inwards corner, this constraint is formulated in (4.6).

$$\begin{cases} g_1(\mathbf{x}) = \sum_{i=1}^{n}(x_i) - L_{\text{wall}}, & \text{if outwards corner} \\ g_1(\mathbf{x}) = L_{\text{wall}} - \sum_{i=1}^{n}(x_i), & \text{if inwards corner.} \end{cases} \tag{4.6}$$

The second constraint is to increase the quality of any proposed solution with respect to scaffold length compared to wall length. A maximum acceptable deviation between scaffold and wall was set to 1 meter as defined in (4.7).

$$\begin{cases} g_2(\mathbf{x}) = L_{\text{wall}} - \sum_{i=1}^{n}(x_i) + 1, & \text{if outwards corner} \\ g_2(\mathbf{x}) = \sum_{i=1}^{n}(x_i) - L_{\text{wall}} + 1, & \text{if inwards corner.} \end{cases} \tag{4.7}$$

To include inventory tracking constraints with regards to available materials had to be set up as defined in (4.8).

$$g_{2+i}(\mathbf{x}) = I_i - 2H_iT_i(\mathbf{x}), \quad i = 1, 2, \ldots, q\,. \tag{4.8}$$

Where inventory of available ledge length $L_1, L_2, ..., L_q$ is denoted as I_i ledges of length L_i. Let $T_i(\mathbf{x})$ be the number of ledges in $\mathbf{x}$ that have length L_i. It is needed to have $2H_iT_i(\mathbf{x})$ ledges for compartment i, where H_i is the number of compartment stories and the constant 2 is accounting for the depth of the scaffold.

For proposed scaffolding solutions it is necessary to include proper access points in accordance to AFS [2] which means to provide staircase access at least every 25 meters. This is incorporated in the algorithm by providing enough ledge lengths with possibility of staircase access within a solution so that placement or access points can be done in an appropriate order to fulfill the access point constraint of 25 meters. The amount of ledge lengths needed within a scaffold is formulated to be more than a scaffold length divided by 25 as seen in (4.9). The ledge lengths that can grant staircase access is formulated as $S_{\text{staircase}}$.

$$\begin{aligned} L_{\text{scaffold}} &= \sum_{i=1}^{n}(x_i) \\ g_{3+q}(\mathbf{x}) &= \left\lceil \frac{L_{\text{scaffold}}}{25} \right\rceil - \sum_{i \in S_{\text{staircase}}} T_i(\mathbf{x}) \end{aligned} \tag{4.9}$$

The last constraint formulated to improve upon solutions quality with regards to roof distance is formulated so that no compartment should extend more than 2 meters past a roof edge for the left or right side of wall compartments for a gable roof. For the first case of scaffolding structure shown in figure 4.1 the constraint in formulated in (4.10). For the second case of scaffolding structure shown in figure 4.2 the constraint is formulated as in (4.11).

$$\begin{aligned} \mathbf{g}_{4+q+i}(\mathbf{x}) &= 2 - L_i, \quad i = 1, 2, \ldots, h \\ \mathbf{g}_{4+q+i+1}(\mathbf{x}) &= 2 - H_i, \quad i = h, h+1, \ldots, n \end{aligned} \tag{4.10}$$

$$\mathbf{g}_{4+q+i}(\mathbf{x}) = \begin{cases} 2 - L_i, & i = 1, 2, \ldots, h \\ 2 - H_i, & i = h+1, \ldots, n \end{cases} \tag{4.11}$$

4.3 Genetic Algorithm

The genetic algorithm used for optimization is modified in relation to Pymoos NSGA-II to allow for the specific parameters used for this project and the encoding of the ledge length values. It is important to keep track of the arrangement of ledge lengths for each solution, as different permutations lead to different fitness results.

4.3.1 Sampling

The sampling process of the algorithm generates a population with a specified number of individuals. These individuals are initiated as a random sequence of available ledge lengths as a first step of the optimization algorithm. The available ledges are sent as input to the sampling process. The availability of these ledge lengths depends on the scaffold loading class for the scaffold and the user inventory. To make the algorithm faster and more efficient on this optimization problem, which in reality has a quite small search space, the algorithm has been limited by reducing the span of size on sampled individuals correlating to the amount of ledges in a scaffold. The minimum size of a sampled individual is determined by the length of the longest available ledge length as in (4.12) where L_{ledge} is the longest ledge length, L_{wall} is the wall length to optimize toward and N_{min} is the minimum individual size. The size of the sampled individual is also limited to the maximum number of ledge components depending on the optimization length as in equation (4.13) where N_{max} is the maximum individual size. The walls length influence the maximum amount of ledge components so that for shorter walls it is assumed that the maximum amount of ledges is equal to the rounded up integer of the walls length as this would still mean the amount of combinations for the ledge components are still small. For longer walls of 6 meters or above, the maximum length is the same as the walls length divided by two to keep the combinations to a minimum and to easier find solutions prioritizing longer ledges which in general is more beneficial for longer walls. Furthermore, population size used for the genetic algorithm is 30 and the available ledge lengths depend on the supplier, seen in Section 3.1 and number of length choices is defined as n.

$$N_{\text{min}} = \left\lceil \frac{L_{\text{wall}}}{L_{\text{ledge}}} \right\rceil . \tag{4.12}$$

$$\begin{aligned} N_{\text{max}} &= \lceil L_{\text{wall}} \rceil , \quad L_{\text{wall}} \in [0, 6) \subseteq \mathbb{R}^{+} \\ N_{\text{max}} &= \left\lceil \frac{L_{\text{wall}}}{2} \right\rceil , \quad L_{\text{wall}} \in [6, \infty) \subseteq \mathbb{R}^{+} . \end{aligned} \tag{4.13}$$

$$P = 30 \tag{4.14}$$

Algorithm 1 Sampling

Input: q, $N_{\max}$, $N_{\min}$, P
Output: Population

```
i ← 0
Population ← List()
Population.size ← P
while i < P do
    Individual ← List()
    j ← 0
    size ← random.choice(range(N_min, N_max))    ▷ Determines initial scaffolding compartments
    while j < size do
        Length ← random.choice(q)    ▷ Compartment assigned a random length from supplier
        Individual.Append(Length)
        j ← j + 1
    end while
    Population.Append(Individual)
    i ← i + 1
end while
```

4.3.2 Fitness

To evaluate individual solutions, a fitness function is needed. The fitness function is the inverse objective function and is to be maximized within the algorithm where the best solution has the highest fitness in that category. For this project, there are three fitness functions that correspond to the objective functions f_1, f_2, f_3 mentioned in Section 4.2.1 for the genetic algorithm. Each population individual has access to its objective function value and the fitness values are specified as follows:

Algorithm 2 Fitness

Input: Population
Output: (Population, Fitness)
  $i \leftarrow 0$
  $Fitness \leftarrow List$ ▷ Creates a fitness list for the population
  while $i < Population.size$ **do**
    $fitness\_length \leftarrow 1/Population(i).f_1$
    $fitness\_material \leftarrow 1/Population(i).f_2$
    $fitness\_roof \leftarrow 1/Population(i).f_3$
    $Fitness.append((fitness\_length, fitness\_material, fitness\_roof))$
    $i \leftarrow i + 1$
  end while

4.3.3 Selection

For the genetic algorithm, the selection process is where two parents are selected to then produce two offspring. These parents are determined by using a binary tournament selection where for each parent, two individuals are being compared and the one with the higher fitness value is chosen as one of the parents.

Algorithm 3 Selection

Input: (Population, Fitness)
Output: Parents
  $i \leftarrow 0$
  $Parents \leftarrow List()$
  while $i < \lceil Population.size/2 \rceil$ **do**
    $pair \leftarrow List()$ ▷ List for saving best potential parents
    $j \leftarrow 0$
    while $j < 2$ **do**
      $random1 \leftarrow random.choice(Population)$
      $random2 \leftarrow random.choice(Population)$
      if $random1.fitness > random2.fitness$ **then**
        $pair.append(random1)$
      else
        $pair.append(random2)$
      end if
      $j = j + 1$
    end while
    $Parents.append(pair)$
    $i \leftarrow i + 1$
  end while

4.3.4 Crossover

The crossover produces two offspring from two parents selected from the binary selection process. For the algorithm to work better, it was necessary to constrain some of the crossovers for shorter optimization lengths. Where it is restricted to not using crossover when the individual consists of only 3 or fewer ledge lengths defined as $K_{\min}$. Due to the smaller search space, the crossover probability was set relatively high at a value of 50%.

$$\begin{aligned} K_{\min} &= 3 \\ P_{\text{cross}} &= 0.5 \end{aligned} \tag{4.15}$$

Algorithm 4 Crossover

Input: (Parents, P_{cross}, $K_{\min}$)
Output: Population

```
r ← rand()
Population ← List()
if Parents(1).length ≤ K_min or Parents(2).length ≤ K_min then
    offspring1, offspring2 ← Parents(1), Parents(2)
else if r < P_cross then
    length1 ← randint(1, Parents(1).length)        ▷ Determines split position
    length2 ← randint(1, Parents(2).length)

    part1, part2 ← Parents(1).split(length1)       ▷ Splitting parent in 2
    part3, part4 ← Parents(2).split(length2)

    offspring1 ← part1 + part3                     ▷ Combining parent1 and parent2
    offspring2 ← part2 + part4
else
    offspring1, offspring2 ← Parents(1), Parents(2)
end if
Population.append(offspring1)      ▷ Assigning individuals to new population
Population.append(offspring2)
```

4.3.5 Mutation

Mutation for this specific case is done in two ways, either by scrambling the order of the individual lengths or by changing a random number of the lengths to a new length with 40% probability for each mutational operator. Therefore, the total probability of changing an individual is 80% as the random number is generated only once and no individual can be both scrambled and change ledge lengths in the same mutation process. The ledge lengths choices is defined as n.

$$P_{\text{mut}} = 0.4 \tag{4.16}$$

Algorithm 5 Mutation

Input: Population, P_{mut}, q
Output: Population

```
i ← 0
while i < Population.size do
    Individual ← Population(i)
    r1 ← rand()
    if r1 <= P_mut then                                   ▷ Scramble mutation
        New_Individual ← random.permutation(Individual)
    else if P_mut < r < (2 × P_mut) then                  ▷ Compartment lengths mutation
        j ← 0
        New_Individual ← List()
        while j < Individual.length do
            r2 ← rand()
            if r2 < (1/Individual.length) then
                New_Individual.append(random.choice(q))
            else if r2 >= (1/Individual.length) then
                New_Individual.append(Individual(j))
            end if
            j ← j + 1
        end while
    else if r >= P_mut then                               ▷ No mutation
        New_Individual ← Individual
    end if
    Population(i) ← New_Individual
    i ← i + 1
end while
```

4.3.6 Elitism

To further improve the convergence of the stochastic optimization and to easier find an optimal solution, elitism is used. This saves the best overall fitness solution (individual) between iterations, leading to further evolutionary improvement on the best solutions from each generation. Since this thesis algorithm builds upon the NSGA-II algorithm, elitism has to consider one Pareto front in each generation as the best where multiple individuals are saved between generations. Some solutions are better at certain objectives, and therefore multiple solutions are saved as their total objective values are considered equal.

Algorithm 6 Elitism

Input: (Population, Fitness)
Output: Best_Individuals

```
i ← 0
Pareto_best ← list()
best_fitness ← max(Fitness)
while i < Population.size do
    if Fitness(i) == best_fitness then
        Pareto_best.append(i)            ▷ Saving best individuals as Indices
    end if
    i ← i + 1
end while
Best_Individuals ← Population(Pareto_best)        ▷ Saving best individuals
```

4.4 Scaffolding Solutions Filtering

The MOGA does produce multiple viable solutions, which are considered equal from the optimization run and its Pareto front. However, only one solution can be presented in the visual environment at a time. To choose a solution within the ones available, a filtering process was constructed. The first filtering step was to reduce solutions to those that have no larger than a 0.3 meters wall-fit values, representing the gap between the scaffold and the wall, as this is a recommendation from AFS [2]. By using this criteria no additional consoles is needed to breach the gap between the scaffold and the wall facade which is mostly favored. If however the algorithm does not find a sufficient solution in regards to this wall-fit the algorithm will propose a solution which may violate the 0.3 meter criteria and therefore additional consoles would be needed if not any regeneration is done. The materials used is of the highest priority after wall-fit in the filtering process, and then the reduced roof distance is taken into account.

1. $f_1 \leq 0.3$
2. f_2
3. f_3

If no scaffolding solutions are within 0.3 meters of the wall, then the priority is in the order of material, wall-fit, roof-fit where the maximum value for wall-fit would be 1 meter from algorithmic constraints which means the gap between scaffold and wall facade can always be minimized under 0.3 meters using additional consoles.

1. f_2
2. f_1
3. f_3

5
Results

In this section the results from locating the optimal number of generations, given a small, medium and large test case will be presented. These findings will then be used to construct a visual comparison of the algorithmic solution with that produced by a scaffolding contractor, followed by quantitative test results when comparing the algorithms' performance to employees at ScaffCalc. Subsequently, the performance of an inventory-constrained algorithm will be compared with an unconstrained one, and lastly, the results of a time-unconstrained algorithm for two test-cases will be presented.

5.1 Locate Optimal Number of Generations

In Sections 5.1.1 - 5.1.3 results highlighting the correlations between the number of generations and computational complexity will be presented. In addition, results exploring algorithmic performance for varying populations over a set number of generations will be presented. For each test case presented in Section 3.2, 50 independent runs will be averaged for every 10^{th} generation for three different populations and summarized graphically. For all contour plots, $obj1$ is the normalized length objective function, $obj2$ the normalized material objective function, and $obj3$ the normalized roof objective function.

5.1.1 Small Test Case

The metrics of the figure for the small test case can be found in Table 3.2, where the data presented in Figures 5.1 and 5.2 demonstrates the first set of results for time complexity versus the number of generations and algorithmic convergence, respectively.

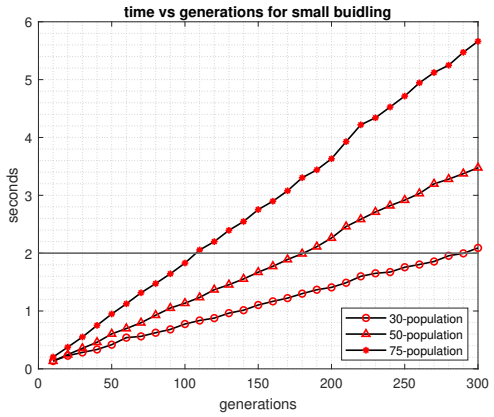


Figure 5.1: *Average time for one run over 50 independent runs, with 30, 50, and 75 generations for the small test case*

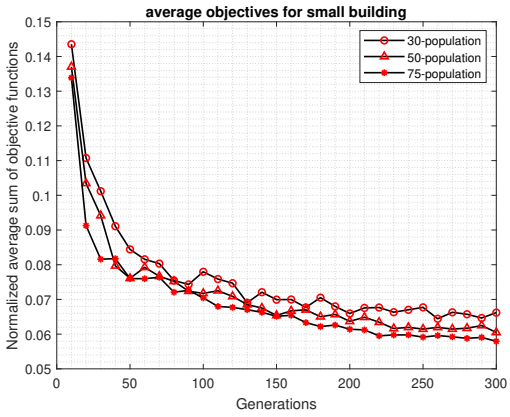


Figure 5.2: *Normalized average sum from objectives, 1,2, and 3, for 30, 50, and 75 in populations for the small test case*

In Figure 5.1 a strong linear correlation is shown between the algorithmic generational time and the number of generations for 30, 50, and 75 populations. Here, a larger number in populations demonstrates a larger slope coefficient as the distance between 75 population marginally grows in distance from 30, and 50 population with an increase in number of generations. Furthermore, Figure 5.2 presents a normalized average result according to (2.22) for all three objectives, with the same number of populations as used in Figure 5.1. Due to an efficient initialization correlating with metrics of the test case the algorithm starts with a relatively small discrepancy from its utopian point, with a worst value of approximately 14.4 for 30 populations. Moreover, the algorithm rapidly convergence between generations 10-50 for each tested number of populations, to stagnate in convergence rate surpass 200. Unlike the results seen in Figure 5.1 the difference in performance between 30, 50, and 75 tends to be less significant, as the trends displayed in Figure 5.2 follow a similar behavior and performance.

During each out of the 50 independent runs, every Pareto front was stored and averaged to be presented in Figure 5.3, where the spread of 10, 100, and 200 generations are displayed.

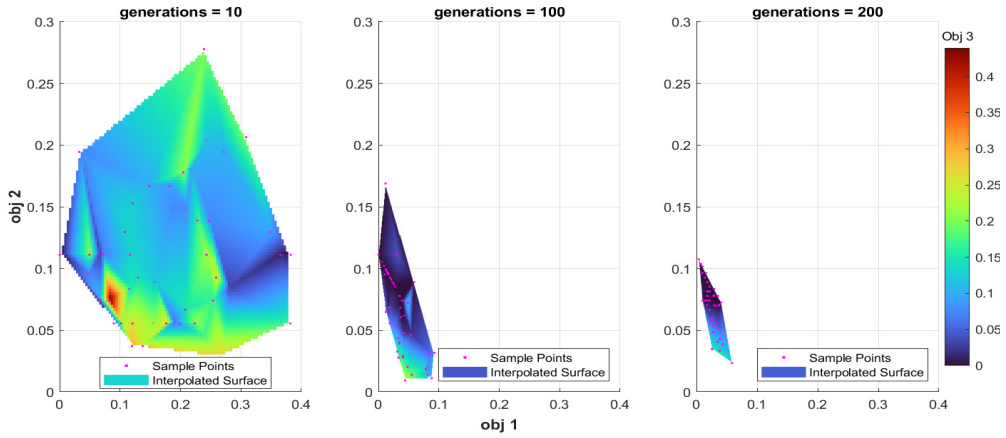


Figure 5.3: *Contour plot for 10, 100, and 200 generations displaying the average Pareto result from 50 independent runs on the small test case with 30 selected as population size. Here, obj1 = normalized length-fit, obj2 = normalized material, and obj3 = normalized roof-fit.*

A clear convergence from generation 10 to 200 can be seen as the objective values for *obj* 1-3 decrease and progress toward origin. Similarly, for 100 and 200 generations the upper area of the contours is darker shaded than the areas closer to origin, which would imply that the algorithm sequentially prioritizes the poorest optimized objective function whilst running.

5.1.2 Medium Test Case

Continuing with the medium test case according to Table 3.2, equivalent evaluations as seen in subsection 5.1.1 were performed and summarized in Figures 5.4, and 5.5.

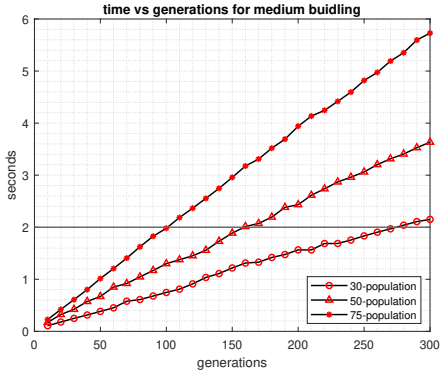


Figure 5.4: *Average time for one run over 50 independent runs, with 30, 50, and 75 generations for the medium test case*

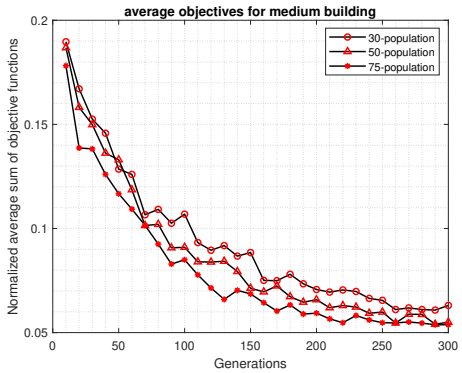


Figure 5.5: *Normalized average sum from objectives, 1,2, and 3, for 30, 50, and 75 in populations for the medium test case*

Similar to the results shown in Figure 5.1, all three trends in Figure 5.4 show a linear relation between time complexity and the number of generations. Another common similarity is the difference in slope between the three populations, where a

large population tends to indicate a larger slope coefficient, as shown in both Figures 5.4 and 5.1. Although given similar trend lines in time complexity between the small and medium test case, there is an approximate difference in time of 1.2% between 30 populations, 11.5% with 50 generations, and approximately 6.2% with 75 in population, over the span of 300 generations. Results indicating a correlation between time complexity and building size.

Furthermore, Figure 5.5 demonstrates similar results as seen in Figure 5.2 with a relatively small discrepancy in overall performance for 30, 50, and 75 in population. By allowing the algorithm to run for 300 generations, the overall performance of each population size outperforms those of the small building. Moreover, a contour plot showing the convergence for the average Pareto front over 50 independent runs is displayed in Figure 5.6.

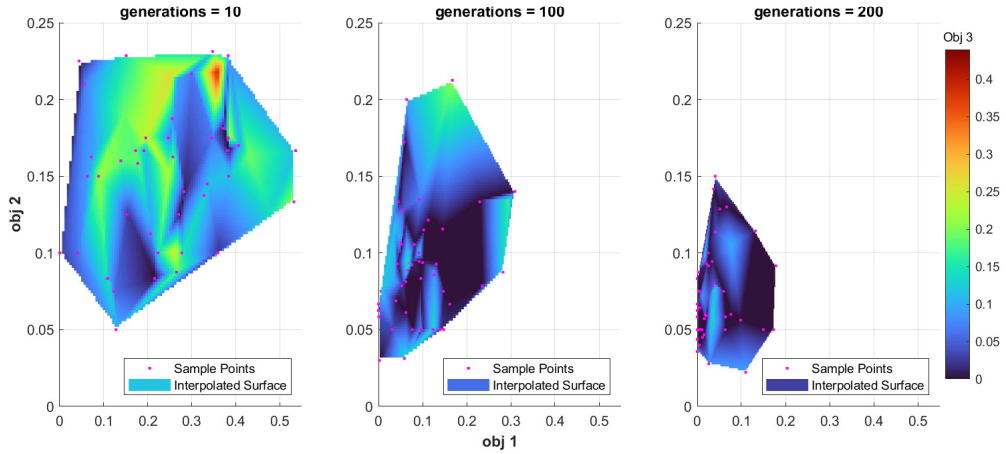


Figure 5.6: *Contour plot for 10, 100, and 200 generations displaying the average Pareto result from 50 independent runs on the medium test case with 30 selected as population size. Here, obj*1 *= normalized length-fit, obj*2 *= normalized material, and obj*3 *= normalized roof-fit.*

Unlike the results seen in Figure 5.3, the algorithm tends to show a lower deviation among the objective functions as highlighted for 100 generations, where more optimized objective functions 1 and 2 (closer to origin) also demonstrate lower, more optimized fitness values for objective function 3. Additionally, less optimized values for objective functions 1, and 2 tend to also display a poorer fitness value for objective 3, which can be seen by the greener area located farthest from origin, given 100 generations. However, by allowing the algorithm to continuously explore, it can be seen how the overall performance for all objective functions is becoming more optimized, similar to that seen for the small test case.

5.1.3 Large Test Case

The final test case employed the metrics presented for the large test case in Table 3.2, and identical evaluations as those previously performed for the small and medium

test case were carried out. In Figures 5.7 and 5.8 the results are presented for both time complexity and overall performance.

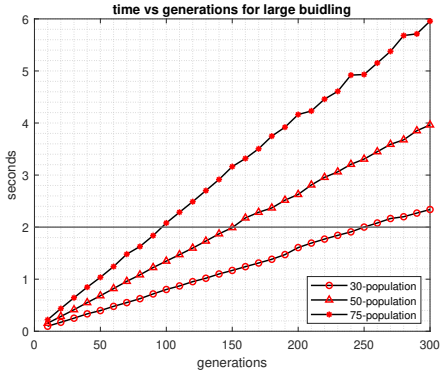


Figure 5.7: *Average time for one run over 50 independent runs, with 30, 50, and 75 population for a large building*

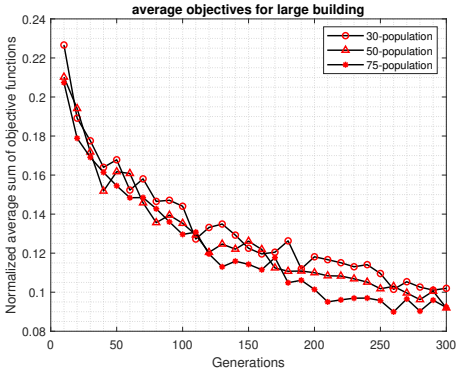


Figure 5.8: *Normalized average sum from objectives, 1,2, and 3, for 30, 50, and 75 in populations for a large building*

Similar to previous results, Figure 5.7 demonstrates linear relations between time complexity and the number of generations for all three sizes in population. By comparing the results in Figure 5.7 with those collected from the medium test case, and shown in Figure 5.4 there is a total difference in time between the large and medium test cases of approximately 3.2 % for 30 population, 5% for 50 in population and 4.2% with 75 population. This would further imply that there is a slight increase in time usage for the large test case compared to the medium test case, supporting the comparison made in Section 5.1.2 that time complexity scales with the input size of the build.

Furthermore, Figure 5.8 demonstrates a successful overall minimization with a converging trend towards the utopian points. However, unlike the performance shown in Figures 5.2 and 5.5 where the convergence has leveled out around 0.05 for all populations, the trend shown for the large test case follows a decreasing trajectory, implying that more generations would be needed before the overall average would stagnate. By observing the contour plots represented in Figure 5.9, it is possible to establish a clear discrepancy compared to the contour plots showcased for the small (5.3), and medium (5.6) test cases.

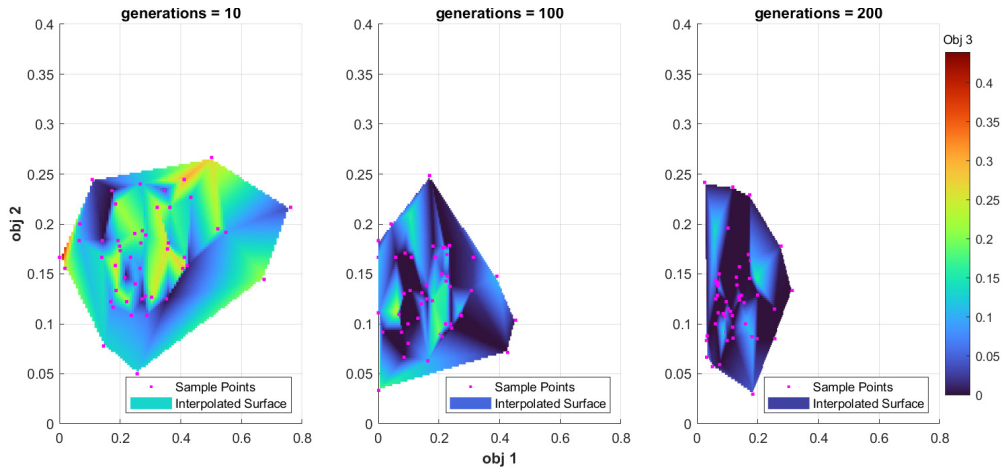


Figure 5.9: *Contour plot for 10, 100, and 200 generations displaying the average Pareto result from 50 independent runs with 30 selected as population size. Here, obj*1 *= normalized length-fit, obj*2 *= normalized material, and obj*3 *= normalized roof-fit.*

First, the minimization of *obj*1 still contains Pareto solutions in generation 200 equivalent to that found in generation 10. However, by observing the spread of the sample points, it is clear that the density is vastly more spread in generation 10, compared to that shown for generations 100 and 200, where the sample points cluster around the origin. Furthermore, due to the linear interpolation of the contour plots, outliers such as those shown in Figure 5.9, push the surface area to each objective value, implying that the overall area of the surface depends on the worst solution of each objective function. The existence of these outliers is further demonstrated in Figure 5.8, as the trend for each populations has not leveled out, suggesting there is room for improvement.

5.2 Visual Comparison Vs User-Built Scaffolds

In the following section, results will be presented according to Section 3.2.2, and summarized in a table together with the logged data of a contractor using the ScaffCalcs web application.

5.2.1 Visual Comparison O1, Constructor Vs Algorithm

The first visual comparison was constructed and executed according to the instructions given in Section 3.2.2, and "Object" $O1$ in 3.4. In Table 5.1 the data retrieved from a constructor modeling a scaffold around test case $O1$ are presented.

Table 5.1: *Amount of scaffolding compartments used for each wall within the contractors project with time taken for the project modeling of O1.*

First wall	Second wall	Third Wall	Fourth Wall	Time (s)
5	5	4	5	779

Based on the results presented for the small test case in Section 5.1.1, the number of generations was set to 250 for the simulation, as this would result in a generational time of less than 2 seconds, as well as a strongly converged outcome based on Figures 5.1 and 5.2. In Table 5.2, 10 independent runs with a population of 30 are presented.

Table 5.2: *Amount of scaffolding compartments used for each wall from 10 independent runs performed with 250 generations and 30 in population on the reconstructed test object O1.*

Run #	First wall	Second wall	Third Wall	Fourth Wall	Time (s)
1	4	4	4	4	1.89
2	4	4	4	4	1.72
3	4	4	4	4	1.79
4	4	4	4	4	1.81
5	4	4	4	4	1.74
6	4	4	4	4	1.72
7	4	4	4	4	1.78
8	4	4	4	4	1.89
9	4	4	4	4	1.72
10	4	4	4	4	1.69
Avg	**4**	**4**	**4**	**4**	**1.78**

The algorithm output was consistent and outperformed the scaffold produced by the contractor with respect to both material and time. Due to the relatively short wall metrics of the test object $O1$, the algorithm proposed the same optimized solution each time, visualized and compared to that of the contractors in Figures 5.11, and 5.10.

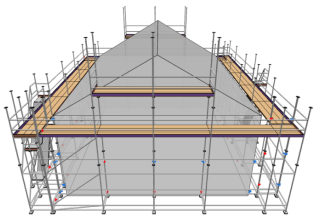


Figure 5.10: *front view of wall 3 built by the constructor on test case O1*

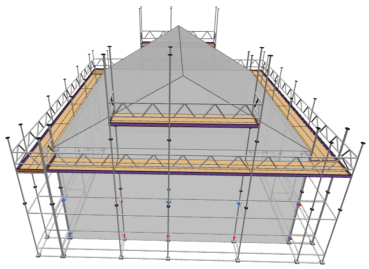


Figure 5.11: *front view of wall 3 generated by the algorithm on test case O1.*

Based on a visual comparison in Figures 5.10 and 5.11, both the constructor and the algorithm have successfully built the scaffold in terms of height and length, as no major parts of the scaffold are extended beyond the edges of the roof, nor are there any need for consoles.

5.2.2 Visual Comparison O2, Constructor Vs Algorithm

Data from the building contractor creating a scaffold around the test object $O2$ are summarized in Table 5.3.

Table 5.3: *Amount of scaffolding compartments used for each wall within the contractors project with time taken for the project modeling of test object O2.*

First wall	Second wall	Third Wall	Fourth Wall	Time (s)
7	12	7	13	3399

The metric for test object $O2$ is summarized in Section 3.2.2, and with a gable length of 20 meters, the results collected from the medium test case, presented in Table 5.4 acted as a good measurement for the choice of 240 generations. The algorithm was run for 10 independent iterations each with 30 in population, and is presented in Table 5.4.

Table 5.4: *Algorithmic results from 10 independent runs performed with 240 generations and 30 in population on the reconstructed test object O2.*

Run #	First wall	Second wall	Third Wall	Fourth Wall	Time (s)
1	8	12	8	12	1.80
2	8	12	8	12	1.96
3	8	12	8	12	1.81
4	8	12	8	12	1.73
5	8	12	8	12	1.93
6	8	12	8	12	1.77
7	9	12	9	12	1.90
8	8	12	8	12	1.70
9	8	12	8	12	1.84
10	8	12	8	12	1.88
AVG	**8.1**	**12**	**8.1**	**12**	**1.83**

As seen for test object $O1$ in Section 5.2.1 each run of the algorithm is sub 2 seconds, which compared to the contractors 3399 seconds demonstrates a vast improvement. Furthermore, the contractor outperforms the algorithm with regard to wall 1, where one less compartment is used. Due to the symmetric nature of walls 1 and 3 the algorithm merely mirrors the first wall resulting in a poorer use of material along wall 3 as well. However, the algorithm consistently generates 12 materials for the fourth wall, which is 1 less than that produced by the contractor. A visual representation of the contractors and an algorithmic solution will be presented in Figures 5.12 and 5.13.

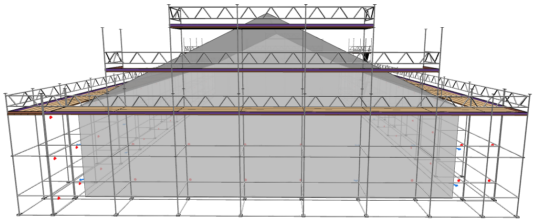


Figure 5.12: *Visualization of the contractors scaffold construction for test object O2, viewed from wall 1.*

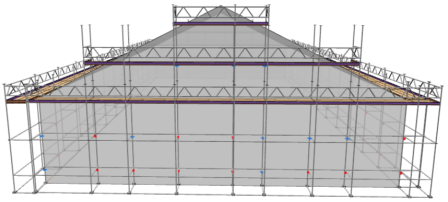


Figure 5.13: *Visualization of the algorithmic scaffold generation for test object O2, viewed from wall 1.*

It can be seen in Figure 5.12 that the contractor uses two 3.05 compartments on the upper floor, allowing the contractor to use only seven compartments, at the expense of potential roof fitness.

5.3 Quantitative Comparison of Scaffolding Generation

The results generated from each test case presented in Section 3.2.3 will be summarized in this section. Unlike in Section (5.2) quantitative measurements comprised of all three objective functions will be introduced with respect to both the algorithm output and the manual users.

5.3.1 Test Case B1

The first test case, B1, had a wall length similar to that introduced for the small test case in Section 5.1.1 with the requirement of an additional scaffold story due to a gable height of 5 meters. To compensate for the additional story, a suitable choice of 240 generations and 30 in population was selected, supported by the data presented in Figures 5.1 and 5.2. The best overall solution constructed by the users as well as generated by the algorithm together with the overall average for the manual users and 100 algorithmic runs is presented in Table 5.5, where the best solutions are chosen according to the filtering process in Section 4.4.

Table 5.5: *Best and average objective metrics for multiple user modeling and algorithm of B1 building for each wall number in parenthesis.*

Metric	User		Algorithm	
	Best	Avg	Best	Avg
Wall fit (1)	0.35	0.38	0.15	0.15
Wall fit (2)	0.28	0.28	0.28	0.28
Wall fit (3)	0.5	0.44	0.15	0.15
Wall fit (4)	0.12	0.06	0	0
Materials (1)	5	5.86	6	6
Materials (2)	4	4	4	4
Materials (3)	5	5.29	6	6
Materials (4)	5	4.14	4	4
Roof fit (1)	0	0.87	0	0
Roof fit (3)	0.03	0.63	0	0
Time taken (s)	555	265	≤ 2	≤ 2

The algorithmic output produced 100 consistent solutions for all 4 walls, where the wall discrepancy was in the range 0.1 - 0.3 meters, 6 and 4 compartments, and a roof distance of 0 meters, for wall 1 and 3. A graphical representation of the best users' solution as well as the algorithmic solution is illustrated in Figures 5.14, and 5.15.

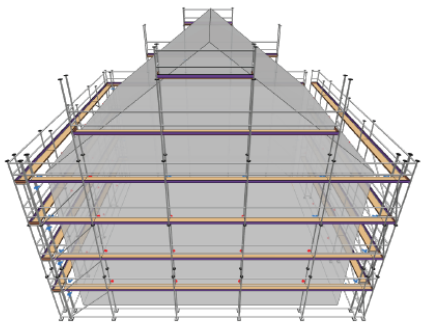


Figure 5.14: *Visualization of best user made modeling of scaffolding for building B1 viewed from front.*

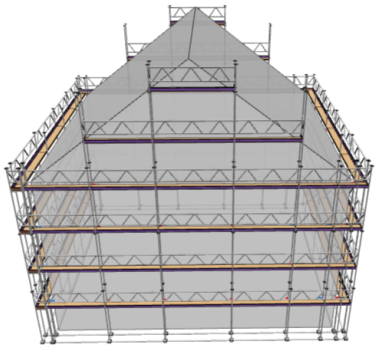


Figure 5.15: *Visualization of the algorithmic scaffold generation for test object B1, viewed from wall 1.*

5.3.2 Test Case B2

The same users who provided models and data for test case B1 in the previous section (5.3.1) were instructed to construct scaffolds around test case B2, with metrics according to Table 3.5. The overall best solution, along with the overall average for users and 100 algorithmic runs, is summarized in Table 5.6, with the best solutions

being chosen according the the filtering process in 4.4.

Table 5.6: *Best and average objective metrics for multiple user modeling and algorithm of B2 building for each wall number in parentheses.*

Metric	User		Algorithm	
	Best	Avg	Best	Avg
Wall fit (1)	0.27	0.35	0.27	0.192
Wall fit (2)	0.17	0.35	0.17	0.17
Wall fit (3)	0.5	0.42	0.27	0.192
Wall fit (4)	0.28	0.24	0.15	0.15
Materials (1)	7	7.9	7	8.69
Materials (2)	7	7.4	7	7
Materials (3)	8	8.2	7	8.69
Materials (4)	7	7.4	7	7
Roof fit (1)	1.55	2.88	1.55	1.32
Roof fit (3)	1.91	3.19	1.55	1.32
Time taken (s)	463.71	391.77	≤ 2	≤ 2

Given the wall metrics for test case B2, the results generated from the large test case in Section 5.1.3 supported the choice of 230 generations, with 30 as population size. The results of 100 independent runs with these metrics were carried out, and the results of the stochastic algorithm are presented in Figure 5.16.

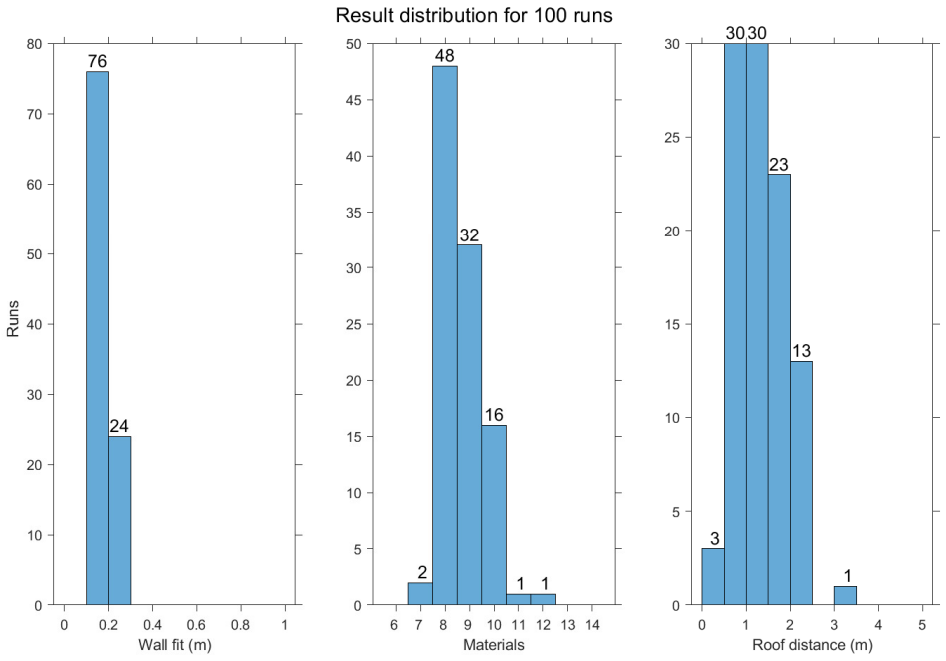


Figure 5.16: *Objective values distribution for the selected solution from 100 run for B2 building dimensions using the genetic algorithm for wall 1 and 3.*

Compared to the results generated for test case B1, the spread for all three objectives

is larger, with wall fitness ranging from 0.1 - 0.3 meters, materials 7 - 12, with a dense cluster around 8-9, and the roof distance mainly grouped between 0.5 - 2.5 meters. All solutions were below 2 seconds, implying a successful choice of generations and population. Further, the best solution created by the user was similar to what the algorithm managed to generate, with the main difference being wall-fit for walls 3 and 4, material for wall 3, and roof fitness on wall 3. A visual representation of the best solution for the user and the algorithm is shown in Figures 5.17 and 5.18.

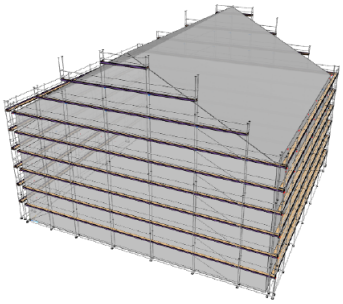


Figure 5.17: *Front-side view of best user modeled scaffolding around building B2.*

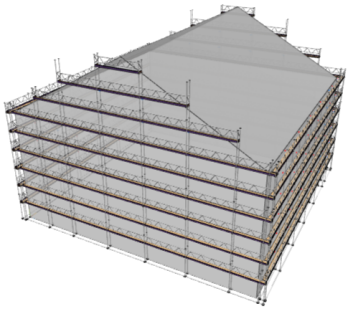


Figure 5.18: *Front-side view of best algorithm generated scaffolding around building B2.*

5.4 Inventory

For the unconstrained test case, all materials were available according to Table 5.7, and for the constrained test, the inventory was specified according to Table 5.8.

Table 5.7: *Table showing the unlimited amount of each ledge length that was available.*

Length	Amount
0.45	∞
0.73	∞
1.09	∞
1.40	∞
1.57	∞
2.07	∞
2.57	∞
3.07	∞
4.14	∞

Table 5.8: *Table displaying the limited amount of each ledge length that is available before the run.*

Length	Amount
0.45	150
0.73	50
1.09	100
1.40	1000
1.57	300
2.07	100
2.57	100
3.07	100
4.14	56

The first results reflect the solution of the test case constrained by time and inventory, which can be seen in Figure 5.19.

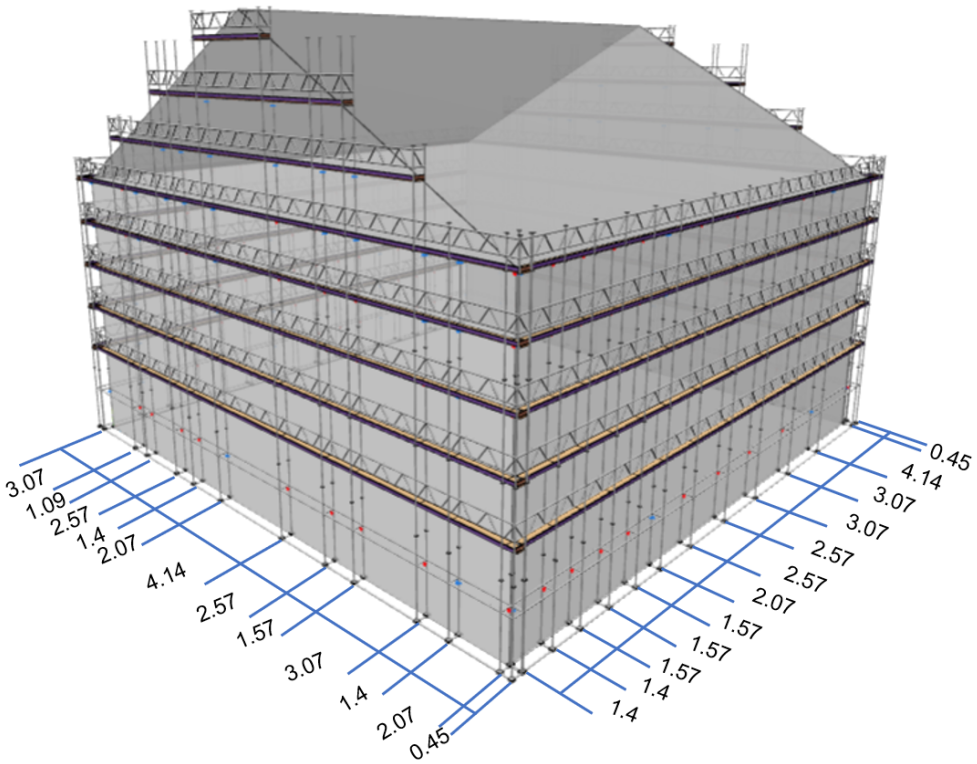


Figure 5.19: *Solution from an constrained inventory run on test case B2.*

The materials available after the run was complete are presented in Table 5.9, where it can be seen that the algorithm has prioritized the longer lengths.

Table 5.9: *Table of remaining amount of ledge lengths after generated scaffold on constrained inventory.*

Length (m)	**Amount**
0.45	108
0.73	50
1.09	68
1.40	894
1.57	26
2.07	8
2.57	4
3.07	12
4.14	2

Furthermore, it can be seen in Figure 5.19 that the majority of longer lengths have been placed along wall 2, with more emphasis on shorter lengths across wall 1. The algorithm has successfully managed to utilize most of the longer lengths [1.57 - 4.14], with a smaller change to the lengths below 1.57. The outputs from the constrained and unconstrained run are summarized in Table 5.10.

Table 5.10: *Summation of the generated solutions with a constrained and unconstrained inventory for test case B2.*

Example	Wall fit[W_1,..,W_4] (m)	Materials [W_1,..,W_4] (m)	Roof fit [W_1, W_3] (m)
Unconstrained	[0.27, 0.17, 0.27, 0.15]	[8,7,8,7]	[1.48, 1.48]
Constrained	[0.15, 0, 0.15, 0]	[11, 11, 11, 16]	[0.64, 0.64]

Here, the results generated from the test case without constraints reflect the average performance of the algorithm similar to those presented in Table 5.6. Moreover, the roof-fit and wall-fit are better optimized with a constrained inventory; however, the materials used are notably larger, with 9 more materials used for wall 4, which is solved last by the algorithm.

5.5 Time-unconstrained Optimization

The results generated from the time-unconstrained test case presented in Section 3.2.4 will be presented in this section. In contrast to the other results shown, this section will include algorithmic results when the algorithms exceed the 2 second runtime to compare objective performance with the results mentioned in Section 5.3.

5.5.1 Test Case B1

The results of 100 time-unconstrained optimization runs using the MOGA are shown in Figure 5.20.

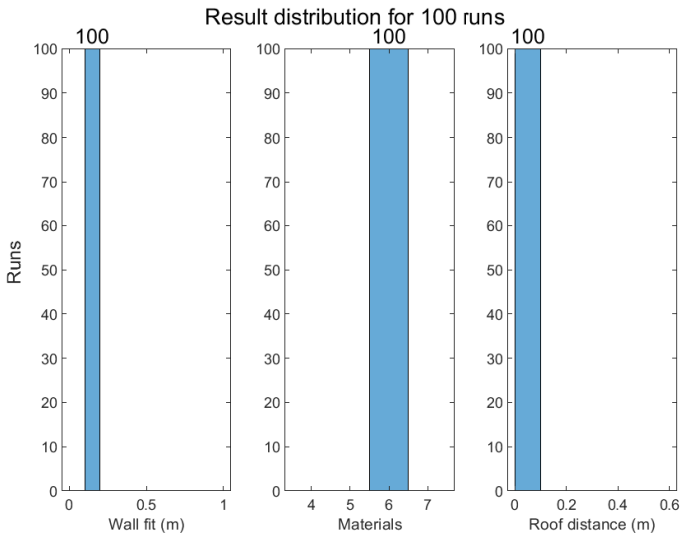


Figure 5.20: *Objective values distribution from 100 runs using non-time related termination criteria for building B1 regarding wall 1 and 3.*

These results imply that the same solution is found within each run and is the optimal solution found with respect to the termination criteria. The average number of generations the algorithm went through from these 100 runs was 1140 with a minimum of 1100 generations and a maximum of 1600 generations. On average, the optimal solution was found at generation 140 as the following 1000 generations did not lead to any improvement on the Pareto front.

Table 5.11: *Metrics for the average objectives generated by 100 runs from the algorithm for time unconstrained test case B1.*

	Wall fit[W_1,..,W_4] (m)	Materials [W_1,..,W_4] (m)	Roof fit [W_1, W_3] (m)
Avg	[0.15, 0.282, 0.15, 0]	[6,4,6,4]	[0, 0]

5.5.2 Test Case B2

The resulting objective values for test case B2 using an unconstrained time termination criterion are shown in Figure 5.21.

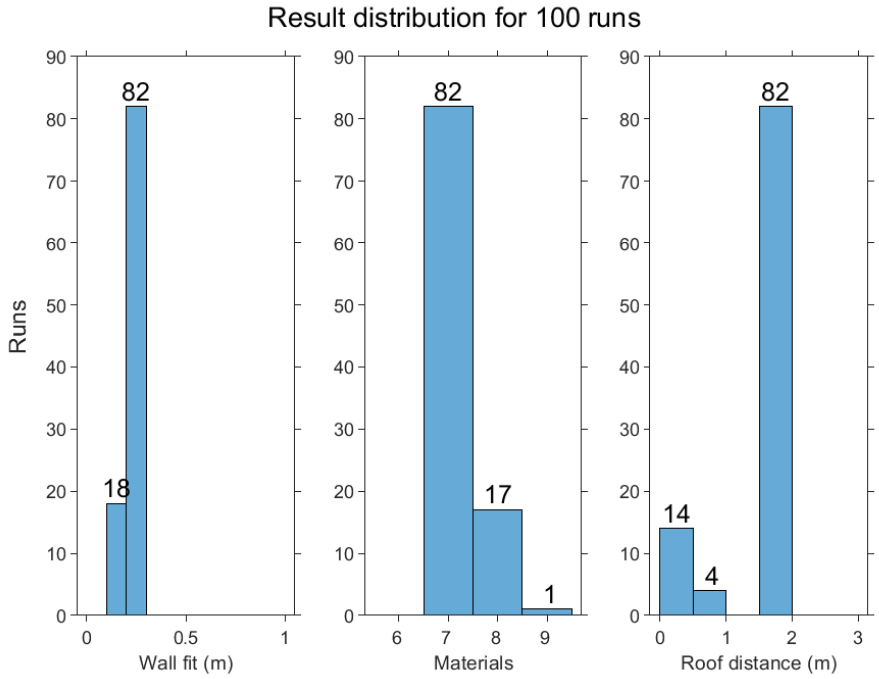


Figure 5.21: *Objective values distribution from 100 runs using non-time related termination criteria for building B2 regarding wall 1 and 3.*

The 100 runs went on for an average of 5475 generations, with a minimum of 1900 and a maximum of 12500 before achieving the termination criteria. This means that the algorithm has converged on an optimal solution with respect to the termination criteria at around 4475 generations, which is equivalent to approximately 40 seconds when using linear interpolation on the results demonstrated in the large test case in Figure 5.7. Comparably, the distribution of solutions varied less than for the time-constrained case presented in Section 5.3.2, with 82% of the runs converging to the solution with the least amount of material, deemed the optimal solution in the time-constrained test.

Table 5.12: *Metrics for the average objectives generated by 100 runs from the algorithm for time unconstrained test case B2.*

	Wall fit[W_1,..,W_4] (m)	Materials [W_1,..,W_4] (m)	Roof fit [W_1, W_3] (m)
Avg	[0.25, 0.17, 0.25, 0.15]	[7.19, 7, 7.19, 7]	[1.34, 1.34]

6
Discussion

In this section the most important results are summarized and discussed along with the methods developed and used to generate said results. The aim of this project was to construct an algorithm consisting of a MOGA and LP to solve scaffolding optimization problems. The proposed research questions were to investigate whether the algorithm could propose more optimized solutions to scaffolding problems than produced manually by a user with a time constraint of 2 seconds. And how large a performance increase with respect to the generation of optimal solutions the algorithm would experience if the time constraint were lifted.

6.1 Algorithmic Evaluation

All mentions of an optimal solution have not been theoretically proved nor derived but rather gathered by allowing the unconstrained algorithm to run and utilize the filtering process presented in Section 4.4 to localize a reference for an optimal solution.

In Section 5.3.1 it is clear that the results produced support the usage of the algorithm to solve smaller problems, similar to that in size of test case B1. Here, the algorithmic robustness is truly enhanced by the generation of consecutive outputs, which holds in comparison to those manually produced by Scaffcalcs users. Comparably, the best manual result is relatively close to that of the algorithms' generated output, with similar roof fitness and slightly more optimized usage in material. However, when the scaffold was manually created, as seen in both tests B1 and B2, the use of mirroring walls 1 and 3 appeared to be often neglected, as the wall fit often varied within each constructed solution. By adapting this mirroring feature, the MOGA only needs to solve wall 1 whilst the LP part solves wall 2 and 4, resulting in roughly halving in time and prevention of user mismatching. Although, given a relative closeness in objective performance, it is hard to ignore the large discrepancy in time when comparing the user's solutions for B1 against the algorithm's solutions. With a factor of approximately 230 in favor of the algorithm, the user experience would be enhanced by adopting the algorithm for smaller constructions. A statement further supported by the convergence graph in Figure (5.3) presented in Section 5.1.1, demonstrating how each objective function on average was successfully optimized within the time limit of 2 seconds, providing a robust

and consecutive solution for smaller building metrics. Furthermore, by lifting the time constraint, the generated solutions remained unchanged, which would indicate that the algorithm had fully converged towards an optimal solution under 2 seconds.

With a positive implementation of the algorithm on smaller time-constrained problems, the results presented in Section 5.3.2 for test case B2 show a wider distribution of the solutions generated for a larger building. This is mainly due to the proposed time constraint of 2 seconds, which inhibits the algorithm from properly exploring the search space in search for an optimal solution. However, despite a larger distribution in suggested solutions, the average performance holds in comparison to the average results generated manually by ScaffCalc users. On average, the algorithm outperforms users in both wall-fit and roof-fit, while producing poorer results for material. It is common for the generated solution to perform better in one or two objectives but rarely in all three. This is mainly due to the pure nature of most multi-objective scaffolding problems, where an improvement in one objective function, such as material, consequently leads to a poorer fit along the walls and roof, caused by the loss of flexibility. The results in Section 5.2.2, further supports this correlation between the objective functions, since the scaffold generated for test case $O2$ by the contractor is clearly outperformed by the algorithm with regard to both wall- and roof-fit. However, the contractor uses one less material on wall 1 and 3. Furthermore, the solution that was considered the most optimized with respect to the material for test case B2 with 7 materials for each wall, 0.27 meters in wall-fit and 1.55 meters in roof-fit was generated only in 2% of all iterations and the optimal solution with respect to wall and roof-fit with 8 material, 0.15 meters wall-fit, and 0.27 meters in roof-fit was generated in none of the iterations, a result that indicates that larger buildings require a larger number of generations and therefore more time to converge towards an optimal solution.

The results obtained from the visual and quantitative tests in Sections 5.2-5.3, support the use of MOGA and LP to improve a manual user solution for smaller buildings while being constrained to a generation time of 2 seconds, shown by the pure robustness of the proposed solutions. However, for larger buildings, the algorithm is able to find optimal solutions but cannot guarantee any consistency with a time constraint of 2 seconds. Therefore, on average the algorithm performs equivalent to that of the user when compared for objective performance on larger building, with an increasing risk of proposing a non-optimal solution correlating with the increase in building size. Consequently, a method was constructed and tested to further investigate the time-unconstrained convergence of the algorithm for larger problems with the results displayed in Section 5.5.2, where on average a generational time of approximately 40 seconds would be required to allow the algorithm to fully converge for test case B2. However, the method to find the number of genes and thus the generational time required the entire Pareto front to remain unchanged for 1000 consecutive runs while being checked every 100^{th}. Therefore, an optimal solution according to our filtering processes could have been located in the early iterations but kept running due to changes in the Pareto front, and a generational time of 40

seconds should be seen as an average upper ceiling for the time required on buildings similar in size to test case B2.

Although the results displayed for the unconstrained test case B2 were more promising than those for the constrained, the algorithm was not consistent in its proposed solutions, and for 4% of the runs, a sub-optimal solution was proposed that was objectively worse than the other solutions. This is most likely the result of premature convergence, where the algorithm is trapped in a local minima which can happen in the case of loss of genetic material within the population. In NSGA-II the usage of crowding distance is applied to increase diversity and therefore spread the genetic material within the population, however, if the population size is too small, the algorithm still runs a risk of converging too fast. Therefore, a higher choice in population size could have produced a more consistent result for the time-unconstrained test case. Furthermore, with the exception of one sub-optimal solution, it could be argued that a generational time of approximately 40 seconds is an acceptable time for the user to wait for a proposed well-optimized solution, especially when the average scaffolding construction time for test case B2 was 390 seconds. Additionally, most of the scaffolding projects constructed on ScaffCalc are smaller than in test case B2, implying that a shorter generational time should be expected for most problems. In summary, the time-constrained algorithm generated an optimal solution with respect to all objective functions in 2% of all its runs, while the time-unconstrained algorithm successfully generated an optimal solution in 82% of the runs with respect to material and another 14% with respect to wall and roof-fit. The results of the time-unconstrained algorithm thus successfully generated an optimal solution in 96% of its runs and, consequently, were 94 percentage units better for larger buildings than when constrained to a 2-second generation time. By introducing a suitable weight factor, the time-unconstrained algorithm could be tuned so that the generation of optimal solutions would only be suggested based on a predetermined importance of the objective functions. By tuning this weight factor to optimize material the algorithm would hopefully converge towards an optimal solution faster as all suggested solutions not focusing on optimizing material would receive a poor evaluation and not be proposed as an optimal solution. This feature would bring more complexity, but also flexibility, to the algorithm and would be worthwhile to investigate in future work.

6.2 Effect of a Constrained Inventory

In Section 5.4 it is clear that when adapting a constrained inventory, the amount of material presented in the solution is larger than that seen by the test case with an unconstrained inventory. However, when the algorithm operates with a limited inventory, it prioritizes the longer compartment widths, indicating that the objective function regarding material usage is operational and works successfully when applied to problems with a constrained inventory. Furthermore, an inventory restricted to a majority of shorter compartments widths will experience greater flexibility, resulting in a better optimized wall and roof fit, as seen in Table 5.10. And vice versa, for an inventory with the majority of longer compartment widths available, flexibility

is lost, resulting in low material usage at the expense of poorer optimized wall and roof-fit. The motivation for a larger material usage on wall 4 for the constrained solution is grounded on the wall prioritization done by the algorithm, where the MOGA proposes a solution for wall 1 that is mirrored onto wall 3, reserving material for both walls. This is then followed by the LP solving wall 2, and finally wall 4. Although a constrained inventory was never compared in terms of wall and roof-fit with a user test case, it could still be argued that it would serve an important role, since the difficulty of backtracking the inventory scales in complexity with the size of the building, which inevitably would result in hard manageable objectives.

6.3 Comparison with Existing Software

Reiterating back to Section 1.1 where today's most commonly used computer-aided tools for scaffolding generation were introduced, the algorithmic approach focused heavily on material optimization and was optimized only in regards to wall-fit to a certain degree [4], [5]. Furthermore, there seemed to be no software that incorporated a third objective along the height of the gables but merely generated scaffolds up to a set height. Comparably, we have successfully developed an algorithm that builds upon these optimization tools, by adding a third objective function to integrate roof fit into the optimization part, and develop a more advanced method for optimizing wall-fit.

6.4 Limitations

The data collected and processed for the qualitative Section 5.3 are largely constructed from the results of the ScaffCalc personal, who are well versed within the scaffolding industry, but not titled scaffolding contractors. Therefore, the results presented should be used as an indication of how the algorithmic performance would compete with manual users. Additionally, due to the lack of feedback from professional contractors, the data collected were also limited in amount, which implied that no statistical certainty could be established as this would require a larger collection of data.

Although the algorithmic performance has shown very positive results, it is vital to recognize that no form of strength or FEM calculations has been adopted in the solutions produced. If the algorithm were to incorporate such calculations, it would not be able to meet the time constraint of 2 seconds, resulting in infeasible solutions.

6.5 Future Research

In order to build on the work presented in this thesis, it would be highly interesting and beneficial to further investigate hyperparameter tuning, as most of our work has focused on algorithmic construction and logic, while the choice of parameters

has been based on external sources, due to time limitations.

Furthermore, since the exclusion of FEM-calculations was adopted in the early stages of the project, the choice to treat the evaluation of a roof-fit value between 0 and 0.5 meters as equal was based mainly on external guidance. By extending upon our findings and including FEM-calculations, this interval as well as the roof-fit in (4.5) would most likely vary with respect to external wind forces. If such a relation could be located, the complexity and adaptability of the algorithm would increase.

The time constraint of 2 seconds was provided by ScaffCalc and was mainly based on their intuition regarding user experience. It would serve a great purpose to investigate this time constraint more thoroughly within the scaffolding industry by interviewing contractors and users of similar platforms. As discussed in this thesis, a user could potentially consider sacrificing time in exchange for a converged and more optimized scaffolding proposal.

7
Conclusion

This thesis aimed to develop an algorithm that would incorporate a multi-objective genetic algorithm and linear programming to solve constrained optimization problems in regards to the scaffolding industry. After a thorough literature and market review, it was apparent that most available software was used for visualization purposes, with some implementing a simpler form of optimization with respect to material and scaffold length, but none, to the best of our knowledge, in the vertical plane.

Results generated from testing our methods indicated consecutive robustness in the proposed solutions when applied to smaller buildings while being time-constrained. These findings would suggest that the algorithm is well versed for smaller constructions, as it guaranteed a solution comparable to that of a users' in performance, whilst doing so approximately 270 times faster. Furthermore, the algorithmic output was not as consistent when applied to larger buildings while time-constrained. However, on average, it produced equivalent results compared to the users', with a few outliers that would suggest that a time constraint of 2 seconds is too strict to guarantee an optimal solution in every run. When the algorithm was allowed to fully converge without the presence of any time constraint, it generated an optimal solution in 96% of the runs for the large building, with an average run time of 40 seconds, which in turn is a 94 percentage unit increase in optimal generated solutions compared to when time constraint.

It can therefore be concluded that the algorithm is well suited for smaller buildings, as it produced solutions with an objective performance equivalent to that of a manual user, while being substantially faster. However, the algorithm cannot guarantee an improved solution for larger buildings if time-constrained to 2 seconds. Although, with an average objective performance for larger buildings slightly better than that of the users', the algorithm is still sufficient under a time constraint in relation to a time-unconstrained run.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY

Bibliography

[1] Arbetsmiljöverket. (2016) Olycksstatistik 1979-2016. Accessed 20.01.22. [Online]. Available: https://www.av.se/produktion-industri-och-logistik/stallningar/arbetsskadestatistik-om-stallningar/arbetsskadestatistik-om-stallningar-1979-2016/

[2] ——. (2016) Scaffolding (AFS 2013:4). Accessed 22.04.22. [Online]. Available: https://www.av.se/en/work-environment-work-and-inspections/publications/foreskrifter/stallningar-afs-20134-provisions/

[3] Swedish Standards Institute. Swedish Standard SS-EN 12811-1:2004 'Temporary works equipment - Part 1: Scaffolds - Performance requirements and general design'. Accessed 21.04.22. [Online]. Available: https://www.sis.se

[4] SMARTScaffolder. Scaffolding design and estimating software. Accessed 29.03.22. [Online]. Available: https://smartscaffolder.com/

[5] Layher. LayPLAN Classic / LayPlan CAD. Accessed 19.04.22. [Online]. Available: https://www.layher.se/teknisk-support/layplan/

[6] Scia. Engineering Software for the Scaffolding and Rack Systems industry. Accessed 19.04.22. [Online]. Available: https://www.scia.net/en/industries/scaffolding-and-rack-systems

[7] Avontus. Avontus Software - Technology Solution for Scaffolding. Accessed 19.04.22. [Online]. Available: https://www.avontus.com/

[8] CerTus. Scaffolding design software. Accessed 19.04.22. [Online]. Available: https://www.accasoftware.com/en/scaffold-design-software

[9] Scafom-rux. ScaffMax, Scaffolding planning software. Accessed 19.04.22. [Online]. Available: https://www.scafom-rux.de/en/products/software/scaffmax

[10] ScaffCalc. Scaffolding calculation software. Accessed 19.04.22. [Online]. Available: https://sv.scaffcalc.com/

[11] K. Kim and J. Teizer, "Automatic design and planning of scaffolding systems using building information modeling," *Advanced Engineering Informatics*, vol. 28, no. 1, pp. 66–80, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474034613000979

[12] M. Wahde, *Biologically inspired optimization methods: an introduction*. UK: WIT Press, 2008.

[13] G. B. Dantzig and M. N. Thapa, *Linear programming 1: introduction*. Springer Science & Business Media, 2006.

[14] L. A. Hannah, "Stochastic optimization," *International Encyclopedia of the Social & Behavioral Sciences*, vol. 2, pp. 473–481, 2015.

[15] J. C. Spall, "Stochastic optimization," in *Handbook of computational statistics*. Springer, 2012, pp. 173–201.

[16] ——, *Introduction to stochastic search and optimization: estimation, simulation, and control*. John Wiley & Sons, 2005, vol. 65.

[17] D. W. Corne and M. A. Lones, "Evolutionary algorithms," *arXiv preprint arXiv:1805.11014*, 2018.

[18] N. M. Razali, J. Geraghty *et al.*, "Genetic algorithm performance with different selection strategies in solving TSP," in *Proceedings of the world congress on engineering*, vol. 2, no. 1. International Association of Engineers Hong Kong, China, 2011, pp. 1–6.

[19] S. L. Yadav and A. Sohal, "Study of the various selection techniques in genetic algorithms," *International Journal of Engineering, Science and Mathematics*, vol. 6, no. 3, pp. 198–204, 2017.

[20] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems." 1975. [Online]. Available: https://hdl.handle.net/2027.42/4507

[21] O. Abdoun, J. Abouchabaka, and C. Tajani, "Analyzing the performance of mutation operators to solve the travelling salesman problem," *arXiv preprint arXiv:1203.3099*, 2012.

[22] L. Jiao, R. Shang, F. Liu, and W. Zhang, *Chapter 3 - Theoretical basis of natural computation*. Elsevier, 2020, pp. 88–89. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128197950000037

[23] C. R. Raquel and P. C. Naval Jr, "An effective use of crowding distance in multiobjective particle swarm optimization," in *Proceedings of the 7th Annual conference on Genetic and Evolutionary Computation*, 2005, pp. 257–264.

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[25] E. K. Burke, E. K. Burke, G. Kendall, and G. Kendall, *Search methodologies: introductory tutorials in optimization and decision support techniques*. Springer, 2014.

[26] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Inter-*

national conference on parallel problem solving from nature. Springer, 2000, pp. 849–858.

[27] J. S. Arora, “Chapter 14 - practical applications of optimization,” in *Introduction to Optimum Design (Fourth Edition)*, fourth edition ed., J. S. Arora, Ed. Boston: Academic Press, 2017, pp. 666–667. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128008065000147

[28] Pymoo. Multi-objective Optimization in Python. Accessed 24.05.22. [Online]. Available: https://pymoo.org/

[29] PuLP. Optimization with PuLP. Accessed 24.05.22. [Online]. Available: https://coin-or.github.io/pulp/index.html