Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Jan Burak

Multi-objective Genetic Algorithm for Engineering Design Optimization

Master's thesis in Computer Science
Supervisor: Ole Jakob Mengshoel
June 2020

kees torn https://www.flickr.com/people/68359921@N08

NTNU
Norwegian University of
Science and Technology

Jan Burak

Multi-objective Genetic Algorithm for Engineering Design Optimization

Master's thesis, Spring 2020

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering
Supervisor: Ole Jakob Mengshoel

Abstract

This report explores how evolutionary computation methods are applied in the field of structural design optimization with a focus on jacket optimization with a genetic algorithm. Jackets are steel towers supporting offshore installations such as oil platforms and wind turbines. Due to high costs associated with material, construction, and installation, there is an interest in decreasing the number, as well as the sizes, of elements in jacket designs. To this end, company Kværner has developed a system employing a genetic algorithm to optimize designs. This report explains the underlying concepts relating to the problem, and explores previous related studies. The project extends the optimization system by employing the Non-Dominated Sorting Genetic Algorithm (NSGA-II), and comparing it to the existing implementation of a genetic algorithm. The new approach managed to show design improvement, but further work is needed to produce designs conforming to engineering rules. It has also been found that the constraint handling method of NSGA-II navigates poorly in the infeasible space. Finally, suggestions for future work are discussed.

Sammendrag

Denne rapporten utforsker hvordan evolusjonær databehandling metoder blir brukt for strukturell optimalisering, med et fokus på bruk av genetiske algoritmer til design av jacketer. Jacketer er fagverksplattformer bygget i stål, som brukes som understøtte for oljeplattformer og vindmøller. Høye kostander av materialet, bygging, og installering, motiverer forsøk på å redusere størrelsene og antallet av elementer i konstruksjonene. Bedriften Kværner har utviklet et system som anvender en genetisk algoritme til å optimalisere designer. Denne rapporten forklarer teorien bak problemet og gir et sammendrag av relaterte prosjekter. Prosjektet utvider optimaliseringssystemet ved å anvende Non-Dominated Sorting Genetic Algorithm (NSGA-II), og sammenligner den med eksisterende implementasjonen av den genetiske algoritmen. Den nye strategien klarte å forbedre designer, men videre arbeid trenges til å produsere konstruksjoner som samsvarer med designkravene. Det ble også oppdaget at begrensningshåndtering mekanismen til NSGA-II fungerer dårlig med ugyldige løsninger. Til slutt presenteres forslag til forbedringer av systemet.

Preface

This project was conducted during the spring of 2020 at the Norwegian University of Science and Technology (NTNU) in Trondheim. It was done as the final project of the five year Compute Science program, with Professor Ole Jakob Mengshoel as the supervisor, and the company Kværner as the sponsor.

Jan Burak
Trondheim, June 9, 2020

Contents

List of Figures

List of Tables

Acronyms

EA - Evolutionary algorithm
EC - Evolutionary computation
FLS - Fatigue limit state
GA - Genetic algorithm
MOEA - Multi-objective evolutionary algorithm
NSGA - Non-dominated sorting genetic algorithm
OWT - Offshore wind turbine
PAES - Pareto archive evolution strategy
PBIL - Population-based incremental learning
PSO - Particle swarm optimization
SPEA - Strength Pareto evolutionary algorithm
ULS - Ultimate limit state

Chapter 1

Introduction

This chapter introduces the project by giving an overview of the underlying topics. The method and the problem domain are presented. The project has been conducted in cooperation with the company Kvaerner, and their system, which was the basis for the project, is introduced. Finally, the structure of the report is presented.

1.1 Background

Evolutionary computation is a family of optimization algorithms inspired by biological processes. The most popular of these is the genetic algorithm that is based on principles underlying evolution. It has been applied to a variety of domains, including structural engineering. Given the wide spread and millennia of history of land structures, much research has been done regarding the optimization of their designs. On the other hand, offshore structures have not been given as much attention, especially being less palatable due to deployment sites that are far away from population centers. These two domains vary greatly in their underlying design challenges. Offshore structures are exposed to waves, wind, corrosion, and seabed conditions. These issues and more require a great deal of domain knowledge to ensure the reliability of a structure, whilst at the same time there is the incentive of reducing the cost of sizeable projects. Thus, a variety of support structures has been devised to conform to a site's con-

ditions in the best way possible. One such type of a structure is a steel truss support called a jacket, that resembles a transmission line support tower. An example of a jacket can be seen in Figure 1.1. Jackets are used as support structures for topsides such as oil and gas platforms, as well as offshore wind turbines. The dimensions of jackets vary with sea depth and the size of the topside, with jacket heights varying from tens to hundreds of meters. Due to the massive amounts of steel, their weight is measured in thousands of tonnes. By optimizing a jacket design, the amount of steel can be reduced whilst still ensuring the design requirements given for an expected structure lifetime. To this end, the company Kvaerner has developed a system employing a genetic algorithm to optimize jacket designs.

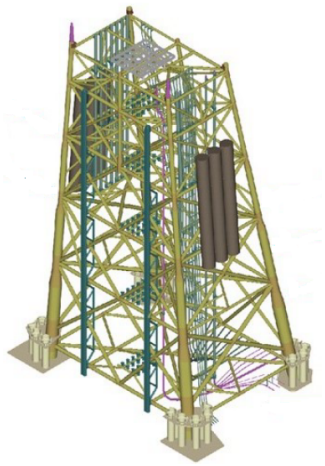


Figure 1.1: Visualization of a jacket. Adapted from Kling et al.[1].

Kvaerner is a Norwegian company providing engineering, production and construction (EPC) of projects in the energy sector and heavy industry. Over the years, Kvaerner has amassed extensive knowledge regarding jacket design throughout a multitude of projects [1]. With recent technological advancements within IT, Kvaerner prioritizes digitalization and innovation to continue improving their services. Among those efforts,

[1]Kvaerner Jackets Brochure `https://www.kvaerner.com/wp-content/uploads/2019/04/Jackets_lowres-1.pdf`

an internal research project of automated jacket optimization has been founded[1]. Code-named L-Alpha, the system parses a jacket design to a format that can be used by a genetic algorithm to produce variations of the original design. By automatically using existing analysis software used by jacket engineers, the new designs are evaluated and put back into the algorithm for further adjustments based on the analysis results. As the amount of practical requirements and the knowledge of jacket engineers is vast, there is still work to be done on ensuring that the generated designs will conform to the established EPC practices. The novel approach to this complex problem presents an intriguing avenue for further research.

In this project, a multi-objective alternative to the genetic algorithm, Non-Dominated Sorting Genetic Algorithm (NSGA-II), was implemented to compare to the existing system. Chapter 2 presents the theory behind the genetic algorithm and NSGA, as well as basics of jacket design. Chapter 3 describes how the current system implements a genetic algorithm for the purpose of jacket design optimization. Chapter 4 summarizes a selection of works relating to structural optimization using evolutionary computation. Chapter 5 presents implementation details and test results. Finally, Chapter 6 discusses the results and draws conclusions in relation to future work.

Chapter 2

Background and Theory

This chapter explains the theory related to this project. It starts with presenting the genetic algorithm in Section 2.1. It is followed up by defining multi-objective problems in Section 2.2 and presenting a multi-objective extension of the genetic algorithm, NSGA, in Section 2.2.1. Finally, basic information relating to jacket design is given in Section 2.3.

2.1 Genetic Algorithm

The genetic algorithm is a method from the evolutionary computation family, that is used for optimization problems. It was introduced by John Holland in 1960s and extended by David Goldberg in 1989 [2]. The algorithm is based on Darwin's theory of evolution. In the theory, genes of the most fit individuals spread throughout generations of a species, and this mechanism is the basis of the genetic algorithm. Thus, in the genetic algorithm, solutions to computational problems are encoded using parameters representing genes. These parameters can be of any type present in programming, such as integers and strings, but the two most common types used are bits and real numbers. It is common that a solution representation suitable for a genetic algorithm is different from the one used when solving a problem. Thus, a transformation is needed between the two forms, which are respectively called the genotype and the phenotype.

When working with a genetic algorithm, various operations may be

used to focus on either exploration or exploitation. Exploration prioritizes trying out diverse individuals in the hopes of finding the global optimum or an interesting area of the search space. Meanwhile, exploitation focuses on finding good solutions quickly, but this may lead to being stuck in local optima.

By having a set of candidate solutions, representing a population, the most fit individuals can be found by comparing them based on how well they solve a specific problem. Thus, the genetic algorithm can only be used on problems where it is possible to gauge either the quality of a solution, or how close the solution is to fulfilling certain criteria. The most fit individuals are then allowed to take part in recombination. Recombination exchanges the genes of two individuals to produce one or multiple new solutions. In that case, the individuals being used for crossover are labeled as parents, and the resulting ones as offspring. Additionally, mutation can be used to adjust the genes of an individual in a random or guided fashion. After application of the crossover and mutation operators, the parent and offspring sets are combined. Usually the maximum size of a population is restricted, so that individuals have to compete with each other to survive. This is done by sorting the population based on a value called fitness, representing how well a given individual solves the problem. By ensuring that a given number of solutions with the highest fitness is brought further to the next generation, the process is guided towards exploring solutions within a promising region of the search space. Putting together the mentioned processes, a pseudocode for a genetic algorithm can be written as follows.

Algorithm 1: Genetic algorithm pseudocode

Initialize the population;
Evaluate all members of the population;
while ***Termination*** *condition not reached* **do**
  Select individuals in the population to be parents;
  Create new individuals by applying **recombination** and **mutation** operators to the copies of parents;
  Evaluate new individuals;
  Replace some/all of the individuals in the current population with the new individuals;
Return the most fit individual

The main parts of the algorithm, marked by bold text in the pseudocode, are described below.

Initialization

The starting point of a genetic algorithm is the initial population. It can be created by assigning random values to the genes, or by using one or multiple promising candidate solutions. If the number of the input individuals is lower than the required size of the population, it can be filled up with new individuals. These can be created through the crossover and mutation operators, or by adjusting the gene values of an input individual through a probability distribution.

Evaluation

Every individual is evaluated by applying it to the problem to be solved. If working with a genotype different from the phenotype, the transformation has to take place first. Evaluation tells us how well an individual solves the problem. For example, when finding the shortest path in a graph, we are interested in the length of a proposed path. This information is represented by a fitness value, that is used in the following step.

Selection

Selection is the process of picking individuals from the population that will be used for producing offspring. Choosing the parents in a reasonable way is important for convergence of a genetic algorithm. There are many ways in which selection can be done. Thierens and Goldberg (1994) [3] analyze four such schemes, two of which will be summarized here to provide an example. Proportionate selection, also commonly called roulette wheel, picks parents with a probability proportionate to their fitness. The probability P_i of picking individual i with fitness f_i and population size N can be written as:

$$P_i = \frac{f_i}{\sum_{n=1}^{N} f_n}. \tag{2.1}$$

An often used alternative is tournament selection. Given a tournament size K, K individuals are chosen, and the one with the best fitness becomes a parent. This is done with replacement until a desired number of parents is picked.

After selecting a sufficient number of parents, they are used to produce offspring using the recombination and mutation operators. The implementation of these varies greatly depending on the modality of the genotype, and how much domain knowledge can be exploited.

Recombination

Recombination takes in two parents and produces offspring that have a combination of genes of the parents. Eiben (2003) [4] presents multiple possible implementations of the operator grouped by modality. A simple example for binary genes is one-point crossover. It operates on a string of bits, by choosing a random point around which bits will be exchanged across parents. The first part of the string is taken from one parent, and the second part taken from the other parent. Another combination is produced by switching which parent the first part of the string comes from, as shown in Figure 2.1. By using multiple crossover points, we can implement the N-point crossover. An example of recombination working on floating-point numbers is interpolation, where a child is produced by setting some or all gene values to the average of the gene values of the parents. Domain knowledge can be applied to recombination operators by

exploiting the local structure of the problem. For example, when finding the shortest path in a graph, a promising subpath can be copied to another solution in an attempt to find a better global solution.

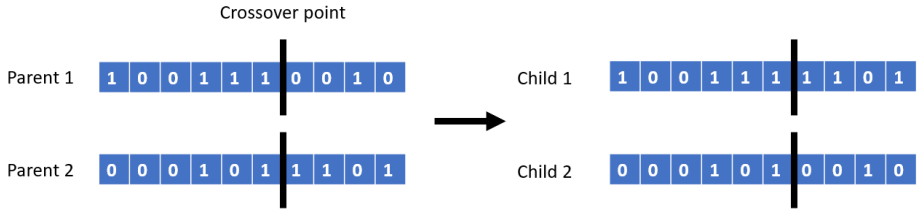


Figure 2.1: Visualization of the binary one point crossover.

Mutation

Mutation operates on a single individual by stochastically adjusting gene values. As opposed to recombination, which mostly serves to propagate existing features throughout generations, mutation increases diversity by introducing new features. For binary genes, the simplest mutation is the bit flip that changes one value from 0 to 1 or vice versa. This can be done for any number of bits. A large number of bits flipped increases diversity, but at the same time may likely produce offspring that are way worse due to diverging from previously found valuable substructures. For floating-point numbers, mutation may either assign a random value within a given range to a gene or adjust it by a specified increment. Like recombination, mutation may be guided by domain knowledge. In the path finding example, one may replace an edge with one that has lower cost.

Replacement

Population replacement, also called reinsertion, is the process of putting together the next generation of the population, which is to be used as input to selection in the next iteration. Eiben [4] categorizes replacement schemes as either age-based or fitness-based. Age-based schemes prioritize survival of the offspring, whilst fitness-based prioritize the best individuals from the combined set of offspring and parents. Whilst fitness-based schemes make the genetic algorithm converge faster, we run into the risk

of plateauing too soon. A combined scheme may be used to ensure the survival of just one or more fittest individual as a compromise.

Termination

A genetic algorithm may be run for a given number generations, until a solution of a given quality is produced, or until no improvement is observed. Given the stochasticity of the algorithm, there is no guarantee that it will converge. Usually it is sufficient to return the fittest individual found during the run, as the fitness value should clearly represent how good a solution is. In many cases this may not be easily achievable, due to the complexity of the problem and conflicting objectives.

2.2 Multi-objective Optimization (MOO)

Many real life optimization problems are not straightforward enough to be able to compare their solutions based on a single value. The evolutionary computation family includes various algorithms that take multiple objectives into consideration. Multi-objective problems are defined in Definition 2.2.1. Definitions in this section are taken from Zavala et al. (2014) [5]. In the definition, $\mathbf{x}^*$ denotes an encoding of a solution, and $\mathbf{f}(\mathbf{x})$ the objectives. It is assumed, without loss of generality, that all the objective functions are to be minimized.

Definition 2.2.1. (Multi-objective Problem) Find a vector $\mathbf{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]$ which satisfies the m inequality constraints $g_i(\mathbf{x}) \geq 0$, $i = 1, 2, \ldots, m$, the p equality constraints $h_i(\mathbf{x}) = 0$, $i = 1, 2, \ldots, p$, and minimizes the vector function $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})]^T$, where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables.

When working with a genetic algorithm, one may implement weighted-sum fitness, where each objective is multiplied by a specified weight and summed together to give a single fitness value. Whilst seemingly simple, this approach introduces the need of finding weights that sufficiently represent the trade-offs between the objectives. Additionally, the algorithm will approach a single solution representing the best know combination of objectives for the given weights, which may not be satisfactory when working with competing objectives.

An alternative approach is to find a set of solutions, where each solution has a combination of objective values that is strictly better than that of any solution not in the set. This concept is called Pareto optimality, as defined in Definition 2.2.2. The feasible region Ω includes all solutions that satisfy the constraints mentioned in Definition 2.2.1.

Definition 2.2.2. (Pareto Optimality) Given a feasible region Ω, a point $\mathbf{x}^* \in \Omega$ is Pareto Optimal if for every $\mathbf{x} \in \Omega$ and $I = \{1, 2, \ldots, k\}$ either $\forall_{i \in I}(f_i(\mathbf{x}) = f_i(\mathbf{x}^*))$ or there is at least one $i \in I$ such that $f_i(\mathbf{x}) > f_i(\mathbf{x}^*)$.

Following that, we can define the operator $\preccurlyeq$, that is used to check for dominance between two solutions. Domination means that one solution has a strictly better combination of objective values than the other solution. This means that the dominating solution has at least one objective value that is better, and the rest of the values is better or equal. This is defined in Definition 2.2.3.

Definition 2.2.3. (Pareto Dominance) A vector $\mathbf{u} = (u_1, \ldots, u_k)$ is said to dominate $\mathbf{v} = (v_i, \ldots, v_k)$ (denoted bu $\mathbf{u} \preccurlyeq \mathbf{v}$) if and only if $\mathbf{u}$ is partially less than $\mathbf{v}$, i.e., $\forall i \in \{1, \ldots, k\}$: $u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i < v_i$.

To give an example of dominance, let's take the data from Table 2.1 for the cost and travel time of three flights. When comparing flights A and B, we can see that the cost of B is better, whilst the flight time of A is better, which means that they do not dominate each other. This is also true for A and C, where C is cheaper, and A is shorter. When comparing B and C, we can see that their flight times are the same, but C is cheaper. Thus, C dominates B, as when comparing only the cost and time properties, C is strictly better.

Flight	Cost	Time
A	1000$	1:00h
B	900$	2:00h
C	500$	2:00h

Table 2.1: Data for a simple multi-objective problem.

When picking a flight amongst these three options, we can readily disregard B. This leaves us with two non-dominated options: A and C. Together, they constitute the Pareto optimal set, which is defined as the set including all non-dominated feasible solutions as per Definition 2.2.4. Furthermore, the objective values of Pareto optimal solutions make up the Pareto front, as per Definition 2.2.5.

Definition 2.2.4. (Pareto Optimal Set) For a given MOP $\mathbf{f}(\mathbf{x})$, the Pareto optimal set is defined as $\mathcal{P}^* = \{\mathbf{x} \in \Omega | \neg\exists \mathbf{x}' \in \Omega, \mathbf{f}(\mathbf{x}') \preccurlyeq \mathbf{f}(\mathbf{x})\}$.

Definition 2.2.5. (Pareto Front) For a given MOP $\mathbf{f}(\mathbf{x})$ and its Pareto optimal set $\mathcal{P}^*$, the Pareto front is defined as $\mathcal{PF}^* = \{\mathbf{f}(\mathbf{x}) | \mathbf{x} \in \mathcal{P}^*\}$.

2.2.1 NSGA-II

One of the most popular multi-objective algorithms from the evolutionary computation family is the non-dominated sorting genetic algorithm (NSGA) introduced by Srinivas and Deb in 1995[6]. Because it is based on the genetic algorithm, NSGA is relatively straightforward to implement when working with a preexisting genetic algorithm. NSGA concerns itself mostly with the reinsertion step, whilst also the selection and evaluation need to be adjusted to take multiple objectives into consideration. In this section, NSGA-II, an improved version of the algorithm, will be presented based on Deb et al. (2002) [7]. The algorithms and figures presented in this section are adapted from that article. The advantage of using NSGA-II comes from the lack of need of a sharing parameter found in other MOO algorithms, and its $O(MN^2)$ running time, where M is the number of objectives and N the number of individuals.

The NSGA-II reinsertion procedure is show in Algorithm 2. It combines the parent and offspring population into one set, that is then divided into consecutive non-dominated fronts by the fast non-dominated sort shown in Algorithm 3. Then, the next population is filled up with the fronts until the next front would not fit in full. At that point, the last front to be added into the population is sorted using the crowded comparison operator $\prec_n$. The operator compares two individuals based on their ranks, which denote which front they are in, and the crowding distance computed by Algorithm 4. The operator is formalized in Definition 2.2.6.

Algorithm 2: NSGA-II Reinsertion

```
// combine parent and offspring population
R_t = P_t ∪ Q_t;
// F = (F_1, F_2, ...) all non-dominated fronts of R_t
F = fast-non-dominated-sort(R_t);
P_{t+1} = ∅;
i = 1;
// while the parent population is not filled
while |P_{t+1}| + |F_i ≤ N do
    crowding-distance-assignment(F_i);
    // include the ith non-dominated front
    // in the parent population
    P_{t+1} = P_{t+1} ∪ F_i;
    // check the next front for inclusion
    i = i + 1;
// sort in descending order using ≺_n
Sort(F_i, ≺_n);
// choose the first (N − |P_{t+1}|) elements of F_i
P_{t+1} = P_{t+1} ∪ F_i[1 : (N − |P_{t+1}|)];
```

Definition 2.2.6. (Partial order) Given individuals i and j, $i \prec_n j$ if $i_{rank} < j_{rank} \vee (i_{rank} = j_{rank} \wedge i_{distance} > j_{distance})$.

The partial order is similar to the Pareto dominance operator $\preccurlyeq$ from Definition 2.2.3, but it also includes crowding distance, which ensures an even spread of individuals across the last front with regards to every objective. The domination operator itself is used in the fast non-dominated sort shown below in Algorithm 3.

Algorithm 3: Fast non-dominated sort on population P

```
forall p ∈ P do
    S_p = ∅;
    n_p = 0;
    forall q ∈ P do
        // If p dominates q
        if p ≼ q then
            // Add q to the set of solutions
            // dominated by p
            S_p = S_p ∪ {q};
        else if q ≼ p then
            // Increment the domination counter of p
            n_p = n_p + 1;
    if n_p = 0 then
        // p belongs to the first front
        p_rank = 1;
        F_1 = F_1 ∪ {p};
// Initialize the front counter
i = 1;
while F_i ≠ ∅ do
    // Q is used to store members of the next front
    Q = ∅;
    forall p ∈ F_i do
        forall q ∈ S_p do
            n_q = n_q − 1;
            if n_q = 0 then
                // q belongs to the next front
                q_rank = i + 1;
                Q = Q ∪ {q};
    i = i + 1;
    F_i = Q;
```

The fast non-dominated sort splits the combined parent and offspring population P into a set of non-dominated fronts $\mathcal{F}$. It does so by first finding all the non-dominated solutions by doubly iterating the total population P and adding them to the first front $\mathcal{F}_i$, whilst the dominated

solutions are added to each individual's domination set S_p and their domination counters n_p are incremented. Then the fronts are consecutively iterated to reduce the domination counter of individuals not yet added to the next front and add them if the counter reaches zero. This is the most computationally expensive part of the algorithm, which caps the overall complexity at $O(MN^2)$. This was an improvement over the original NSGA with a complexity of $O(MN^3)$.

Algorithm 4: Crowding distance assignment on a non-dominated set $\mathcal{I}$

```
// number of solutions in I
l = |I|;
// initialize distance
forall i ∈ I do
    i_distance = 0;
forall objectives m do
    // sort using each objective value
    I = sort(I, m);
    // make boundary points always selected
    I[1]_distance = I[l]_distance = ∞;
    // for all other points
    for i = 2 to (l − 1) do
        I[i]_distance =
          I[i]_distance + (I[i + 1].m − I[i − 1].m)/(f_m^max − f_m^min);
```

In addition to fast sorting, NSGA-II offers diversity preservation through crowding distance computed by Algorithm 4. Crowding distance aims to present a measure that can be used by the partial order operator to pick out individuals which are diverse, on the assumption that a large difference in objective values corresponds to a significant difference in the genetic makeup between individuals. It is computed by sorting a front on each objective. After each sort, the distance of individuals at the edges is set to infinity. For the rest, the normalized difference of M neighboring individuals' objective values is added to the crowding distance. The distance corresponds to the sum edge length of a cuboid delimited by the M neighbors as shown in Figure 2.2 for two objectives.

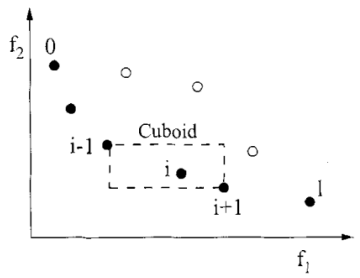


Figure 2.2: Cuboid representing the crowding distance found for an individual i, where f_1 and f_2 are objective functions.

The overall NSGA-II reinsertion process can be simply presented as in Figure 2.3.

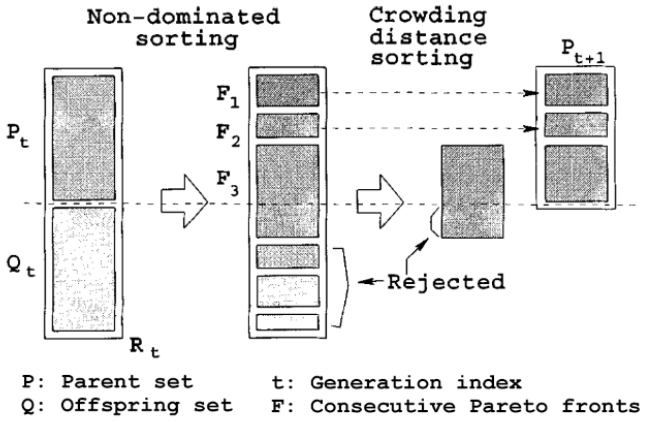


Figure 2.3: Reinsertion process of NSGA-II. When adding individuals from $\mathcal{F}_3$, the ones with the highest crowding distance survive.

The article also proposes a constraint handling scheme. This is done by replacing domination with constrained-domination, as shown in Definition 2.2.7.

Definition 2.2.7. (Constrained-domination) A solution i is said to constrained-dominate a solution j, if any of the following is true.

1. Solution i is feasible and solution j is not.
2. Solutions i and j are both infeasible, but solution i has a smaller overall constraint violation.
3. Solutions i and j are feasible and solution i dominates solution j.

In addition to reinsertion, a different selection mechanism is needed to take multiple objectives into consideration. In the NSGA-II article, an adjusted tournament selection operator is used. It works the same way as presented in Section 2.1, except fitness comparison is replaced with the partial order operator from Definition 2.2.6.

2.3 Jacket design

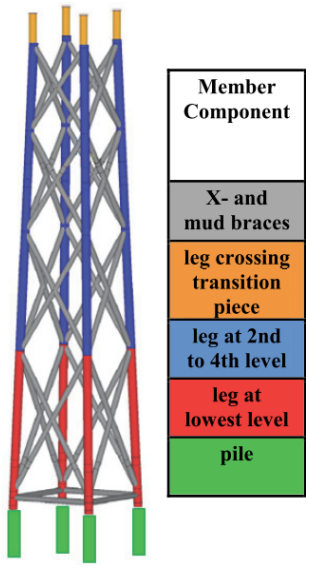


Figure 2.4: Drawing of the UpWind reference jacket with element labels. Adapted from Chew et al. (2013)[8].

Jackets are steel structures used as support for offshore oil and gas installations, as well as wind turbines. They are similar to truss towers

used for land constructions, such as transmission line supports. Jackets are made up of tubular beams connected in a pattern that is able to support a topside. For oil and gas installations, examples of topsides include drilling platforms, production platforms, living quarters, and heliports[9]. A public reference jacket, called UpWind, that has been used in multiple studies to analyze the optimization potential of jackets, is shown in Figure 2.4 [10]. The jacket is designed to support a wind turbine, and the jacket's height is 70 meters from the seabed to the bottom of the wind turbine.

The main parameter driving jacket design is the number of legs. Some wind turbine supports can be constructed with only three legs, whilst more demanding topsides may require four or more legs. Beams that constitute a leg are called chords, meaning that they are outer members of the structure, deciding its shape, and receiving in other components. The incoming beams are called braces. The elements are connected by joints reinforced by cans as shown in Figure 2.5. The endpoints of a brace going into a can are called stubs. Jackets are fastened to the seabed by piles that extend deep below the seabed.

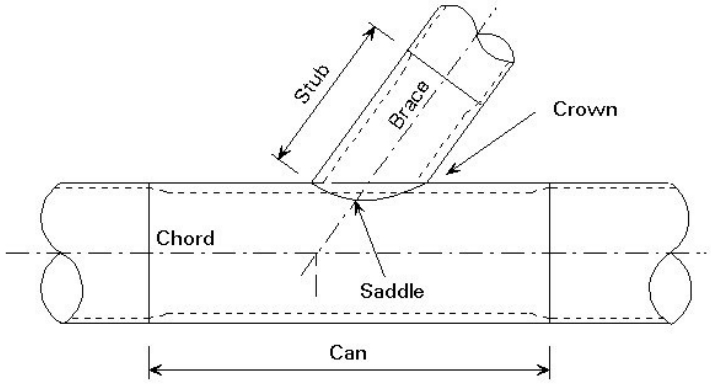


Figure 2.5: A diagram showing a can and its proximate elements [11].

Figure 2.6 shows the different jacket design topics and exemplifies how jacket elements are grouped together. Elevations are horizontal groups and the number of elevations is a main design driving parameter together with the number of legs. Rows are vertical groups, and as shown in the lower-right corner of Figure 2.6, there can be multiple rows.

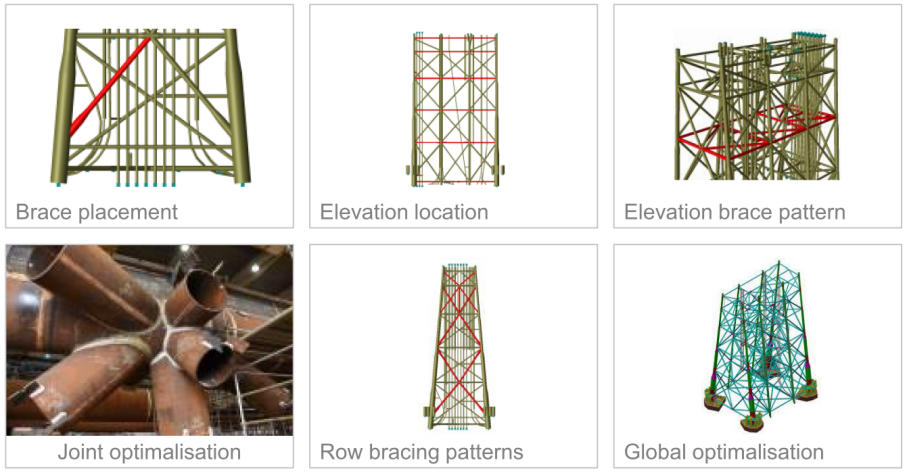


Figure 2.6: Jacket design topics [12].

Jacket design is a subfield of structural design, which concerns itself with the strength, rigidity, and stability of structures. Kicinger et al. (2005) [13] outline three domains of structural optimization, which are topology, shaping, and sizing. These are exemplified in Figure 2.7. When simplifying a structure to a graph, topology optimization decides the number of nodes and connections between them, corresponding to joints and beams in a jacket. Shaping decides the angles between elements and their lengths, and sizing adjusts the diameters and thicknesses of elements. Jackets are mostly constructed from tubular steel beams akin to pipes. Figure 2.8 presents a cross section of such an element and shows a difference between the outer and inner diameter. The majority of a jacket's construction cost comes from the cost of steel. There is also a significant cost dependent on welding, which is influenced by the angles at which the elements are connected together[14].

Jackets need to endure environmental conditions present at the deployment site, as well as fulfill structural design requirements. Thus, the design process is influenced by factors such as wind, waves, seabed conditions, and seismic activity. Multiple analyses are performed to ensure that a design fulfills requirements, including static, fatigue, installation, and transportation.

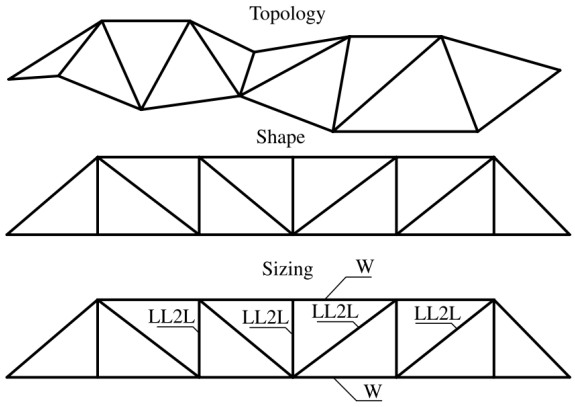


Figure 2.7: Discrete topology, shape, and sizing optimization domains [13].

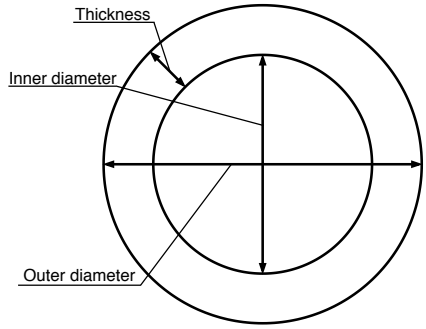


Figure 2.8: Cross section of a pipe with parameter labels. Two parameters are sufficient to fully describe the cross section.

2.3.1 Limit State Design

This subsection has been reused from the specialization project preceding this thesis [15].

Limit state design is a structural engineering method where a design is tested against a set of potential loads to check whether a structure fulfills the design criteria, such as structural integrity and durability[16][17]. Loads are categorized into dead loads, which are constant over time, such as originating from the weight of the structure, and dynamic loads that include waves and winds. In addition to operational extremes, offshore structures need estimated loads for transit and installation. A limit state is a condition where a criterion is no longer fulfilled, and thus expresses a possibility of failure. Safety margins between highest likely loads and weakest resistances have to be ensured so that they are large enough to tolerate fatigue damage. Two kinds of limit states are particularly relevant: ultimate (ULS) and fatigue (FLS). ULS design limits the stress that materials experience to conform to strength and stability demands. FLS design is concerned with simulating aerodynamic and hydrodynamic loads and extrapolating for the required lifetime of a structure. The analyses can be used to find the utilization of elements, value of which is below 1 if a particular element is expected to endure the predicted loads. Additionally, structures may be designed while optimizing for stiffness, which describes the rigidity of a structure. Stiffness corresponds to the ability of resisting deformation and deflection when force is applied. Alternatively, it may be represented as compliance, which is the inverse of stiffness.

Chapter 3

Current system

The company Kvaerner has developed a system that uses a genetic algorithm to optimize jacket designs. It was written in C# and the .Net framework. A diagram of the modules and flow is shown in Figure 3.1. Following that is a description of the relevant parts of the system.

3.1 Initial design and representation

The input to the system is a jacket design produced by the engineers. The design is a set of JavaScript files describing the geometry of the jacket and the environmental conditions of the deployment site. The files are in the format used by DNV GL's Sesam software package[1]. The software is used by both the engineers and the system to evaluate the designs.

The geometry of a design is described by a few main numerical parameters like elevation heights, and foot and head dimensions that decide the overall shape of a jacket. After objects representing legs are created, horizontal braces are added at the elevation heights. Subsequently a bracing pattern is formed by adding members in positions relative to the elements already created. The design files include a list of possible pipe sizes to choose from, and every member is given a specific pipe size. The system

[1]DNV GL Sesam for fixed structures `https://www.dnvgl.com/services/offshore-and-marine-structural-engineering-sesam-for-fixed-structures-1096`

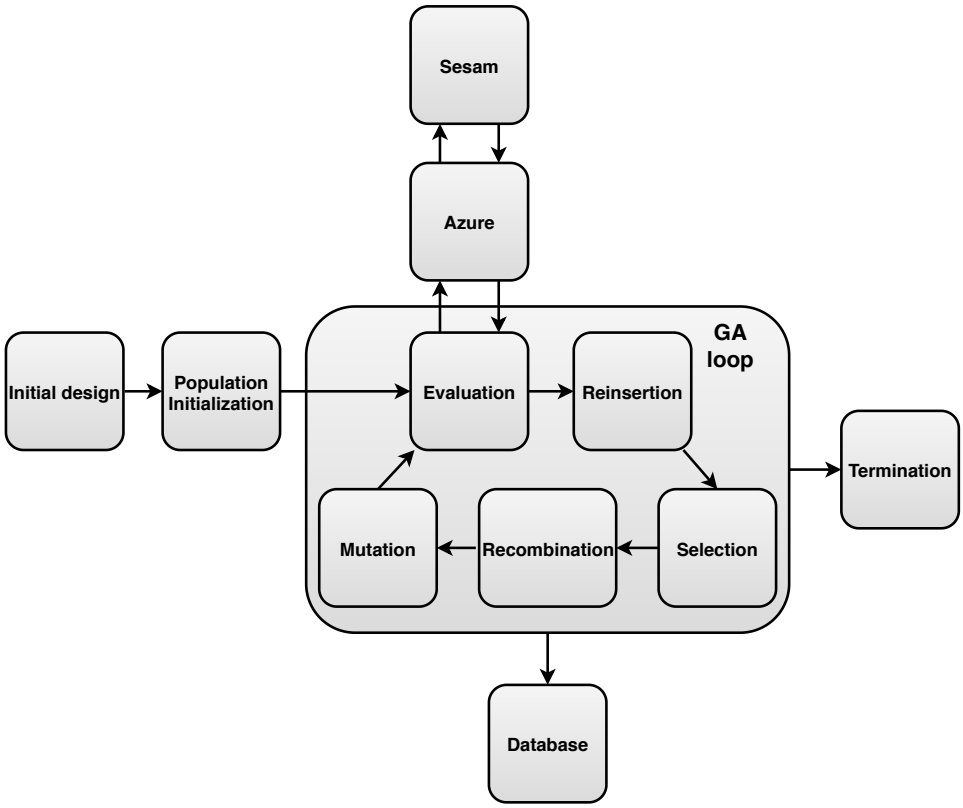


Figure 3.1: A diagram of Kvaerner's system.

parses the geometry of the initial design to create a list of elements and a graph describing its connections. Given that a 3D model of a jacket is created procedurally, it is not possible to easily adjust the positions of individual members. A list of genes specifies how the genetic algorithm should adjust the main numerical parameters within given ranges, as shown in Listing 3.1. Additionally, the list of elements can be used to create genes that adjust the pipe sizes of each element. These genes are restricted by gene profiles, which decide which properties can be adjusted within a given range and precision. An example of a gene profile is shown in Listing 3.2.

```
GeneDef;     GeneUnits;  GeneType;    GeneRange;   Accuracy;
Width_B_X;   m;          Uniform;     16 37;       1;
H10;         m;          Uniform;     8 12;        0.5;
Width_B_Y;   m;          Uniform;     24 40;       1;
H15;         m;          Uniform;     -17 -12;     0.5;
```

Listing 3.1: An example gene list in a csv format for two elevation heights and two base widths.

```
new GeneProfile{
        Ceiling = 1.5,
        Floor = 0.5,
        Precision = 0.01,
        SetType = "Beam",
        NamePrefix = "Diameter"}
```

Listing 3.2: An example gene profile for the diameter property of a beam element with given range and precision.

Thus, the genes of an individual consist of the main parameters specified by a gene list, as well as diameters and thicknesses of individual members. As shown in Listings 3.1 and 3.2, a discrete real-valued representation is used. Each gene has an ID that shows whether two genes from two different individuals relate to the same structural element, which is needed for the computation of diversity and recombination.

3.2 Population initialization

The initial population is created by cloning the individual created from the input design and adjusting every gene by a normal distribution. New individuals are created in this way until the population is filled to a given maximum size. As shown in the system diagram in Figure 3.1, the initial population goes straight to the evaluation, so that individuals are given meaningful fitness values before going to selection. The reinsertion used on the initial population does not affect it, as at that point the number of individuals is equal to the maximum population size.

3.3 Genetic algorithm loop

Once the population is initialized, the algorithm enters a loop that may be run for a given number of generations or until timeout. The loop proceeds in the same way as for the classic genetic algorithm in Algorithm 1 in Section 2.1. After every iteration, the individuals and their evaluation reports are saved in a database for later retrieval.

3.4 Selection

The system implements multiple selection schemes, including tournament, roulette wheel, and ranked selection. Ranked selection is implemented by first adding the individual with the best fitness to the parents list. Then, individuals are added consecutively to the parents list without replacement based on a diversity measure. The diversity measure returns an individual with the highest score, that is computed by dividing a diversity score by fitness. The diversity score is computed by summing the absolute difference between each gene value of an individual and the respective gene value of the individual with the best fitness. Computation of diversity score is exemplified in Table 3.1. Ranked selection was used to produce the results presented in Chapter 5. The operator is based on AlSukker et al. (2010) [18].

Gene ID	Best Individual	Candidate Individual	Absolute difference
1	1.2	0.5	0.7
2	0.7	0.8	0.1
3	1.3	1.4	0.1
4	0.9	1.1	0.2
		Diversity score:	1.1

Table 3.1: Example of how diversity score is computed in ranked selection.

3.5 Recombination

Selection returns a list of parents that is used by recombination to produce offspring. Parents are taken in order in pairs from the list to exchange

their genes and produce two offspring. The scheme that was used in this project is called crossover uniform interpolation. For each pair of genes across two individuals relating to the same structural element, the scheme can perform two operations with given probabilities. The first operation switches around the gene values. The second one interpolates the values by assigning new values to the genes of both individuals taken from a uniform distribution with the two original values as endpoints. This is exemplified in Listing 3.3. The values are automatically brought to the precision determined by the gene profile.

```
maxValue = Max(gene1.Value, gene2.Value);
minValue = Min(gene1.Value, gene2.Value);
gene1.Value = minValue +
    (maxValue - minValue) * Random.Double();
gene2.Value = minValue +
    (maxValue - minValue) * Random.Double();
```

Listing 3.3: A function assigning random gene values from a uniform distribution. Random.Double() returns a real number between 0 and 1.

3.6 Mutation

The system implements multiple mutation schemes, and the ones used in this project were create beam, dispose beam, and uniform adaptive mutation. The create beam mutation adds new elements to the structure by either utilizing existing joints or creating new joints for the beam to connect to. The genes of the new beam are randomized. Conversely, the dispose beam mutation removes an existing element at random. These two operations carry out topology optimization. Uniform adaptive mutation adjusts gene values by a given step percentage and with a given probability. The step percentage gives a step value based on the gene profile for the type of the element being adjusted. Additionally, the probability of performing that mutation is adjusted in every iteration of the genetic algorithm loop. If no improvement in the best individual has been observed in the previous generation, the probability of mutation is increased by a given increase rate. If improvement has been observed, the probability is lowered. Maximum and minimum possible values of probability

are given to limit the range within which it can be adjusted. Given that this mutator adjusts the genes relating to the diameters and thicknesses of elements, as well as the main parameters, it constitutes shaping and sizing optimization.

3.7 Evaluation

Once offspring has been created, it needs to be transformed to its phenotype form to be used as input to evaluation. As mentioned in Section 3.1, the initial design is in a format used by the Sesam software package that evaluates designs. To evaluate individuals created by the genetic algorithm, the gene values are applied to a copy of the initial design to adjust the element sizes and main parameters. Additionally, the structure list is scanned to apply the creation and disposal of beams. Once the design files are prepared, they are uploaded to virtual machines in the Azure[2] cloud that are running Sesam software. This is done due to the fact that evaluation is the most computationally demanding part of the system, and employing virtual machines enables running evaluations in parallel.

The current system has the capacity to perform ULS and FLS analyses, briefly described in Section 2.3.1, based on the load cases present in the design files. Only ULS has been used in this project. The analyses produce a report for each individual that includes the inherent physical properties of each element, their utilizations, and angles between the elements. Since the genotype includes parameters influencing multiple elements at the same time, the information obtained from the evaluation is not readily available in the genotype. The report is used to compute the objectives that go into the fitness function presented in Listing 3.4.

```
1 double fitness = TotalWeight * JacketWeightCost
2 + TotalUtilizationFactor
3 + UtilizationTargetDeviationSum
4 + WeldWeight * WeldingWeightCost
5 + FitnessLambda * (WajacPenalty
6   + AngleViolations * AngleDeviationSum
7   + UtilizationViolations * UtilizationDeviationSum
```

[2] Azure cloud `https://azure.microsoft.com/en-us/overview/what-is-azure/`

```
8   + PileViolations * PileDeviationSum);
```

Listing 3.4: Fitness computation

Each objective present in Listing 3.4 is briefly explained in Table 3.2. Some objectives are multiplied by a deviation sum, which expresses how much the resulting values differ from desired ranges. The terms in the parentheses are penalties, and they are multiplied by a fitness lambda, that expresses how heavily the penalties should be weighted in relation to the other factors. What has been omitted from Listing 3.4 is that every summand is multiplied by its respective lambda, representing the weight of each objective. Thus it is an example of a weighted-sum approach of combining multiple objectives.

Variable	Description
Total Weight	Sum weight of each individual element. It is multiplied by a cost that approximates the cost of steel.
Total Utilization Factor	Sum of each element's utilization.
Utilization Target Deviation Sum	Sum of each element's absolute difference from a specified target utilization. Target is usually between $0.85 - 0.95$ as a safety margin.
Weld Weight	A measure describing the complexity and the cost of welding elements together. It is computed based on the area between connected elements, and then multiplied by an estimated cost.
Wajac[3]Penalty	Equal to 1 if Wajac failed. Wajac consists of hydrostatic, hydrodynamic, and wave fatigue analyses.
Angle Violations	Number of angles between elements that are not within required ranges.
Utilization Violations	Number of elements with utilization above 1.
Pile Violations	Number of piles that experience forces outside of a required range.

Table 3.2: An overview of the variables used for fitness computation.

[3]Sesam Wajac analysis `https://www.dnvgl.com/services/hydrostatic-and-`

3.8 Reinsertion

The fitness computed in the evaluation stage is used during reinsertion to decide which individuals of the combined parent and offspring set are taken into the next generation. The system employs a simple reinsertion scheme that sorts the individuals by ascending fitness values and lets the first half survive. This scheme is completely elitist and there is no duplicate removal, so the population may lose diversity over time. Once reinsertion is complete, the genetic algorithm loop continues with the next generation starting with the selection stage, unless the termination criterion is fulfilled.

`hydrodynamic-analysis-wajac-2244`

Chapter 4

Related Systems

This chapter has been reused from the specialization project preceding this thesis [15].

This chapter presents some of the work related to jacket optimization, as well as on similar applications of evolutionary computation for structural design. Table 4.1 outlines the systems that are described below.

4.1 Jacket optimization

Multiple studies have been performed on the topic of jacket optimization for offshore wind turbines (OWT). Some work has been done on evolutionary computation methods, as summarized here, but much of the literature is based on gradient methods. There is no consensus on which methodology is preferred. The EC methods offer navigation of a poorly understood search space, but recent efforts have made gradient methods more feasible, whilst offering reduced computational time. Due to the apparent lack of literature relating to EC jacket optimization for oil platforms, the focus in this section will be on OWT supports.

System	Algorithm	Objective	Domains	Structure
Pasamontes 2014	GA	Mass	Size, shape	Jacket
Schafhirt 2014	GA	Mass	Size	
Martens 2015	GA	Cost (material, painting, welding)	Size, topology	
Häfele 2016	PSO (ALPSO)	Cost (material, production, coating, transition piece, transport and installation)	Size, shape, topology	
Kunakote and Bureerat 2011	PAES, PBIL, NSGA-II, SPEA2, MPSO	Mass, eigenfrequency, structural compliance	Topology	Emergent
Noilublao and Bureerat 2011	PBIL, SPEA2, MOSA	Mass, compliance, frequency analysis parameters	Size, shape, topology	Truss tower

Table 4.1: Overview of the systems described in this chapter.

4.1.1 Genetic algorithm for shape and sizing optimization

Pasamontes et al. (2014) used a genetic algorithm to optimize the mass of the OC4 jacket, with 16 binary genes for the thicknesses and diameters of the elements, as well as 3 genes for the bay heights[19]. This resulted in diameters ranges of 800-2048mm and 400-2448mm, and thicknesses of 40-104mm and 10-74mm, for legs and braces respectively. Similarly, the heights varied by ±8192mm from the positions of the original design. The optimization of bay heights was reported as belonging to the topology domain, yet according to Kicinger's classification it fits the shaping category, due to not changing the number of joints or beams[13]. The effect of varying the bay heights can be seen in figure 4.1.

The initial population was generated randomly by setting the gene values within the given ranges using a uniform distribution, as well as being subject to validity constraints. The system was tested with population sizes of 15 and 30. ULS and FLS analyses were performed to calculate whether the reliability constraints were fulfilled, and the designs that passed had their scaled fitness calculated based on mass. Individuals were chosen for reproduction by roulette wheel selection, and the next

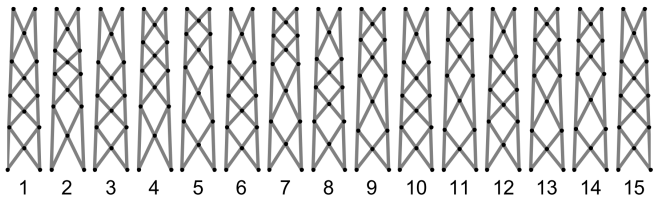


Figure 4.1: Jacket shaping with varying bay heights. Taken from Pasamontes et al. (2014)[19].

generation was picked through elitism by picking the fittest individuals from both the parents and the children. Every 10th generation, immigration was performed by introducing new randomly generated designs to increase the diversity of the population. The operators used were multipoint crossover and bit flipping mutator. The mutator had a 0.05 probability, that was increased by 0.01 if no change in best fitness was observed for 20 generations. The termination condition was reaching 300 generations. For the single load case used, the results amounted to around 30% reduction in mass, regardless of whether shaping optimization was used. A particular problem relating to the binary representation was pointed out, where thresholds for gene values emerged that made navigating the search space difficult.

4.1.2 Extending the genetic algorithm

Schafhirt et al. (2014) extended the work of Pasamontes, but only for the thickness and diameter parameters [20]. The number of iterations until convergence was reduced to one third through multiple improvements. Reanalysis was introduced by using mutators on promising designs without repeating the time-domain analysis, and instead approximating the performance based on parent's results. Additionally, fitness precalculation was employed to check whether the fitness of an offspring was higher than the lowest fitness in the current population. Individuals that did not pass the check, were not analyzed and instead discarded to save time. Furthermore, similarity checks were performed bit by bit to discard overly similar offspring. Regarding further work, the authors propose that instead of, or in addition to, the mass, the fitness calculation may be based on stiffness,

eigenfrequencies, or damage capacity. It was also discussed that designs optimized based only on static loads may express superfluous redundancy due to large safety margins.

4.1.3 Topology optimization based on a ground structure

Martens et al. (2015) used a genetic algorithm to optimize both the topology and sizing of an OWT jacket [21][22]. The starting point of the process was a ground structure with fully connected and symmetric faces, hollow middle, and a fixed number of joints. The genes consisted of diameters and thicknesses of members. Shaping optimization was not performed, as the relative positions of joints were not adjusted. This can be especially seen in the fact that as opposed to the OC4 jacket, the legs of the tested design were not at an incline, and the system had no way of arriving at such a solution.

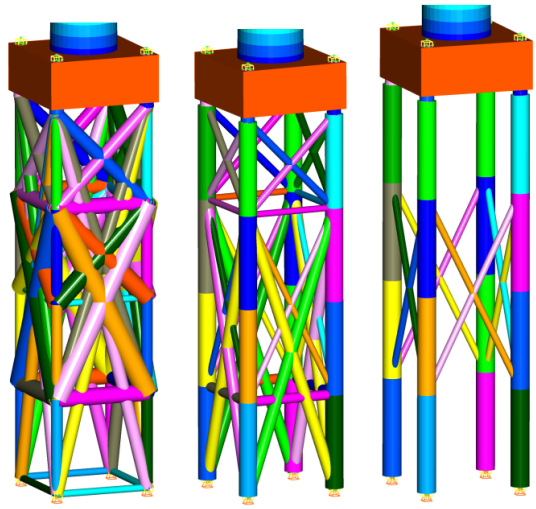


Figure 4.2: Example progress of the optimization process of Martens' system. Taken from Martens et al. (2015)[21].

The initial population was generated randomly based on user-specified gene constraints, as well as a probability to remove members from the ground structure. Fitness was calculated with the material and instal-

lation cost, approximating the efforts of cutting, welding, and painting. Furthermore, the genetic algorithm operators were implemented similarly to those of Pasamontes' system. An adaptive mutation rate was employed by measuring the diversity of the population based on the number of unequal genes between the best and the worst individuals. A test was run for 100 generations that took 24 hours to complete. Some designs resulting from the test can be seen in figure 4.2. Manual optimization was performed on a fixed topology for comparison, and it resulted in slightly better fitness of the final design.

4.1.4 Modelling the jacket search space and particle swarm optimization

Häfele et al. (2016) performed extensive modelling of the jacket design search space and employed a particle swarm optimization algorithm. The focus was on representing the problem in a way that did not restrict how the resulting solutions may be formed. Thus, the parameters consisted of lengths, diameters, thicknesses, foot and head radii, number of legs, number of bays, and whether mud braces are used. An example of a resulting jacket can be seen in figure 4.3.

The faces seem to be restricted to an X-brace per bay, and thus the system did not allow other nodes than X-joints between the legs, unlike the system of Martens. The fitness consisted of the sum costs of material, production, coating, transition piece, transport and installation. Additionally, there was a constraint on whether the lifetime of all joints is above the design lifetime, as well as whether the utilization ratios are below 1. Thus, the problem was formulated as single objective with constraints, and a variant of PSO called Augmented Lagrangian Particle Swarm Optimization was employed to be able to handle the constraints. A test was run for the OC4 reference jacket. The computation took 47 days, and the resulting three-legged jacket can be seen in figure 4.4. The cost was calculated to be 2.36 million €, compared to the supposed 2.91 million € cost of the best four-legged design.

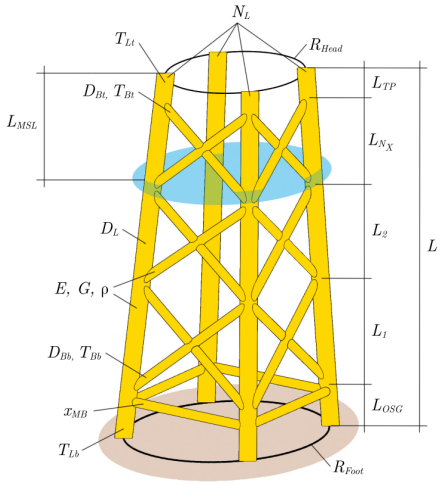


Figure 4.3: An example jacket from Häfele's system together with the design driving parameters. Taken from Häfele et al. (2016)[23].

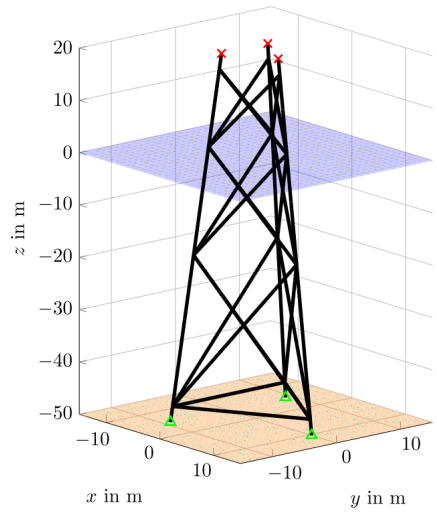


Figure 4.4: Häfele's test result jacket. Taken from Häfele et al. (2016)[23].

4.2 Structural optimization

Evolutionary computation methods are popular in structural design, and thus there are multiple works describing their applications. Kicinger et al. (2005) have summarized the most prominent systems of the field that have been developed since the 1970s[13]. The majority of the systems employ genetic algorithms, whilst some use other EAs such as genetic programming or evolution strategy. Additionally, the comparison includes optimization domains, researched structure, representation, type of fitness calculation and objectives, as well as constraints. The study also discusses the potential of structure generation by EC methods, and issues regarding representation and constraint handling. Furthermore, the multi-objective nature of structural design problems is described, whilst noting the lack of sufficient research regarding it. Addressing this in a more recent paper, Zavala et al. (2014) have performed a survey of various works applying multi-objective optimization to structural design problems[5]. The survey

collected over 50 references and classified them based on multiple criteria, such as algorithm used, kind of structure, and optimization domain. Methods from the evolutionary algorithm family were pinpointed as the most popular, with NSGA-II being studied most often. At the same time a wide variety of related solvers was mentioned. The study also discusses the issue of floating-point versus binary encoding and proposes additional research on the use of differential evolution for the former. Some of the relevant systems applying EC for structural design are presented below.

4.2.1 Evaluation of multi-objective EC methods for a grid-based structure

Kunakote and Bureerat (2011) performed an evaluation of four multi-objective evolutionary algorithms with regards to topology optimization, with structural compliance, natural frequency, and mass as objectives[24]. These algorithms were: Pareto archive evolution strategy (PAES), population-based incremental learning (PBIL), non-dominated sorting genetic algorithm (NSGA-II), strength Pareto evolutionary algorithm (SPEA2), and multi-objective particle swarm optimization (MPSO). Furthermore, NSGA and SPEA were tested for various parameters for both the binary and real value encodings. PAES and PBIL used binary, and MPSO used real valued encodings. Instead of optimizing for a specific structure, four environments were modelled based on a grid with specific criteria, and thus the search space could allow for an arbitrary structure. The evaluation was based on hyper-volumes and generational distances of the generated Pareto fronts. Hyper-volume is an indicator of a front's advancement and extent through the calculation of the volume between each individual and a reference point. A generational distance of a front is measured in relation to an approximate true Pareto front based on the minimal distances between individuals across the two fronts. These indicators describe the optimality of the resulting front, as well as its diversity. Overall, PBIL scored highest based on these measures, with PAES in second. The resulting structures were compared to ones produced by a gradient-based method, and a significant inferiority of the former was noticed. Regardless, the use of EC methods was said to be advantageous due to their robustness when met with search spaces of structural design problems. The study also reported that parameter configurations with higher muta-

tion probability, rather than crossover, performed better for NSGA and SPEA.

4.2.2 Multi-objective optimization of a truss tower

Noilublao and Bureerat (2011) compared the performance of PBIL, SPEA2, and archived multi-objective simulated annealing (AMOSA) when optimizing the topology, shape, and sizing of a truss tower with regards to five objectives[25]. The tower was 50 meters high with 10 to 20 levels, similar to jacket bays, that had four identical faces per level. Varying the number of levels constituted topology optimization. Ten design variables controlled the shape of the tower regardless of the number of levels, by manipulating the heights and widths of the levels. Lastly, each level consisted of 16 elements, cross-sections of which were adjusted by six parameters per level. The cross-sections could have five different values, where one of the values corresponded to removing the elements. Examples of the resulting towers are shown in figure 4.5. The optimization of the three domains

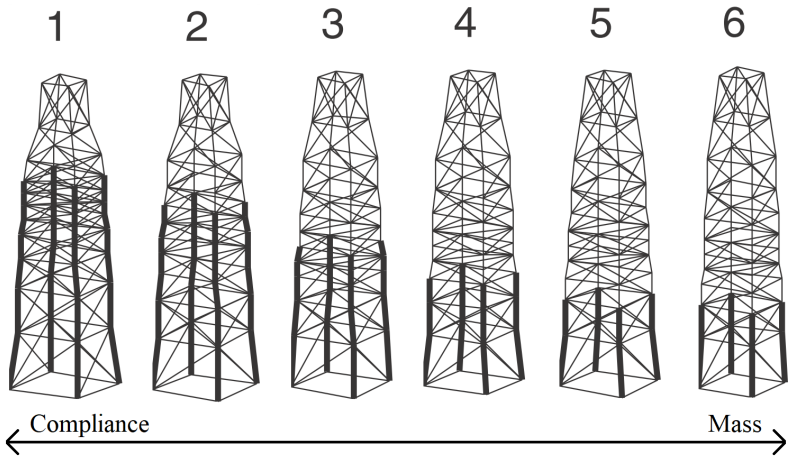


Figure 4.5: Six towers from a final Pareto front. Towers further to the left are optimized for compliance, whilst the ones further to the right are optimized for mass. Adapted from Noilublao and Bureerat (2011)[25].

was performed simultaneously, and it is argued that this produces better results than sequential optimization. The evaluation was split into four

bi-objective problems, with mass being present in every problem. The first problem included compliance, while the three others included objectives related to frequency analysis. The evaluation metrics consisted of averages and standard deviations of hyper-volumes and generational distances computed over 10 runs per problem. Based on these metrics, PBIL turned out to be the most effective for the compliance problem. For the three others, PBIL produced more extended fronts, whilst the fronts of SPEA2 advanced more. The runs were performed with a population size of 200 and lasted for 300 generations. The computation took between 30 to 360 minutes per run based on the problem, with AMOSA having strictly lowest computation time.

Chapter 5

Implementation and results

This project extended the current system presented in Chapter 3 with the possibility of employing NSGA-II (Section 2.2.1) for the jacket design problem. Section 5.1 explains how NSGA was implemented along with other changes. Results from the original GA and the newly implemented NSGA are presented in Section 5.2.

5.1 Implementation

As described in Section 2.2.1, NSGA shares most of its functionality with the genetic algorithm presented in Algorithm 1. The mutation and recombination operators that were already present were reused. Evaluation was adjusted to utilize multiple objectives. A multi-objective tournament was implemented as selection. Reinsertion was implemented according to Algorithm 2, including the constrained-domination operator from Definition 2.2.7.

5.1.1 Evaluation

Two objectives were chosen to be optimized by NSGA. The first one was the sum of total weight and weld weight, which were grouped together due to being closely correlated. The second objective was the total utilization factor. The two objectives are competing, as lower utilization is

obtained by using more steel, thus giving a larger safety margin. This in turn increases the total weight of the design. The two objectives were to be minimized. Angle, utilization and pile violations, as well as the Wajac penalty were used as constraints. Their sum was used as constraint violation per Definition 2.2.7, and if the sum was equal to 0, a solution was deemed feasible.

Additionally, the Azure cloud solution was replaced with OneCompute[1], that was developed by DNV GL, the company responsible for Sesam. Close integration of the evaluation software with the cloud solution allowed for more reliable execution. The switch improved the analysis time, allowed running more evaluations in parallel, and diminished the number of failed evaluations to an insignificant amount.

5.1.2 Selection

The pseudocode for the implemented multi-objective tournament selection is shown in Algorithm 5. The tournament size was set to 3. The value of the tournament size is a trade-off between exploration is exploitation. With lower values, it is more likely for worse solutions to reproduce. With higher values, the better solutions are prioritized, but the number of possible unique pairings is lower. The operator takes in the previously generated population and picks a number of competitors equal to the tournament size. The competitors are compared against each other using the partial order operator. This picks the individual with the best rank, and if ranks are equal, it picks the one with the largest crowding distance. The best individual amongst competitors is added to the set of parents. Given that the recombination operator that comes after selection picks parents in pairs from the set, pairs of duplicates are avoided by temporarily removing the last tournament winner from the set of candidates for the next iteration. Other than that, picking of parents is done with replacement, unlike the selection mechanism described in Section 3.4 of the current system.

[1] OneCompute About `https://devpeuwwa01platonecomputedocumentation.azurewebsites.net/docs/v3.0/`

Algorithm 5: Multi-objective tournament selection

```
candidates = population;
tournament_size = 3;
parents = {};
i = 0;
while ||parents|| < ||population|| do
    competitors = pick tournament_size individuals at random
     from candidates;
    forall c1 in competitors do
        c1.wins = 0;
        forall c2 in competitors do
            if c1 ≼ c2 then
                c1.wins = c1.wins + 1;
    add competitor with the most wins to parents;
    if i mod 2 = 0 then
        remove winner from candidates;
    else
        add previously removed winner back to candidates;
    i = i + 1;
return parents;
```

5.1.3 Reinsertion

Reinsertion was implemented by closely following Algorithm 2 of the NSGA-II reinsertion scheme. Early tests have shown that some duplicate solutions end up in the population. Reinsertion was adjusted to remove individuals that had combinations of objective values that were equal to those of other individuals. Thus after every reinsertion, a population was obtained where every individual had a unique combination of objective values.

5.2 Results

This section summarizes the tests that have been done to evaluate the performance of NSGA-II for the jacket design problem, as well as compare it to the performance of the original genetic algorithm.

5.2.1 Test setup

The starting point of the tests was a design of the Valhall Flank West[2] jacket. A non-final version of the design was used, as to leave room for the algorithm to find optimization potential. The initially used version did not fulfill all the constraints mentioned in Section 3.7. Early tests of NSGA-II have shown that the constraint handling method struggles to navigate in the infeasible space. To mitigate this, the original genetic algorithm was used to produce a feasible design that was used in the tests presented in Section 5.2.2. A diagram of the Valhall Flank West jacket is shown in Figure 5.1.

Parameter	Value 1	Value 2
Mutation probability per element	0.005	0.005
Step percentage	0.1	0.6
Probability ceiling	0.5	0.5
Probability floor	0.005	0.0005
Probability increase rate	0.1	0.05
Probability decrease rate	0.2	0.1

Table 5.1: Parameter for the two uniform adaptive mutations.

Six test runs were performed: three for GA and three for NSGA-II. Each test was run with 50 individuals and for 150 generations. The mutation and recombination operators described in Chapter 3 were used. Two versions of uniform adaptive mutation were used concurrently during each test run, and their parameter values are summarized in Table 5.1. The most significant difference in parameter values is that one of the

[2]Valhall Flank West press release `https://www.akerbp.com/en/valhall-flank-west-successfully-installed/`

mutations adjusts the gene values by a step percentage of 0.1, whilst the other one by 0.6.

Operator	Probability for	Probability	Per which object
Crossover	Application	1.0	Individual
Crossover	Gene switch	0.5	Element
Crossover	Gene interpolation	0.1	Element
Dispose beam mutation	Beam disposal	0.025	Individual
Create beam mutation	Beam creation	0.0025	Individual
Create beam mutation	Joint creation	0.8	Individual

Table 5.2: Probabilities for the operators used, except for uniform adaptive mutation.

The probabilities for the rest of the operators is summarized in Table 5.2. The crossover probability was 1, which meant that every offspring was produced by crossover. The per-individual probability of beam creation and disposal means that every child has the given chance of being adjusted once by the operator. For example, an individual has a single chance of 0.025 to have one of its elements removed at random. The probability for joint creation comes into play only after an individual has been picked for the beam creation operation.

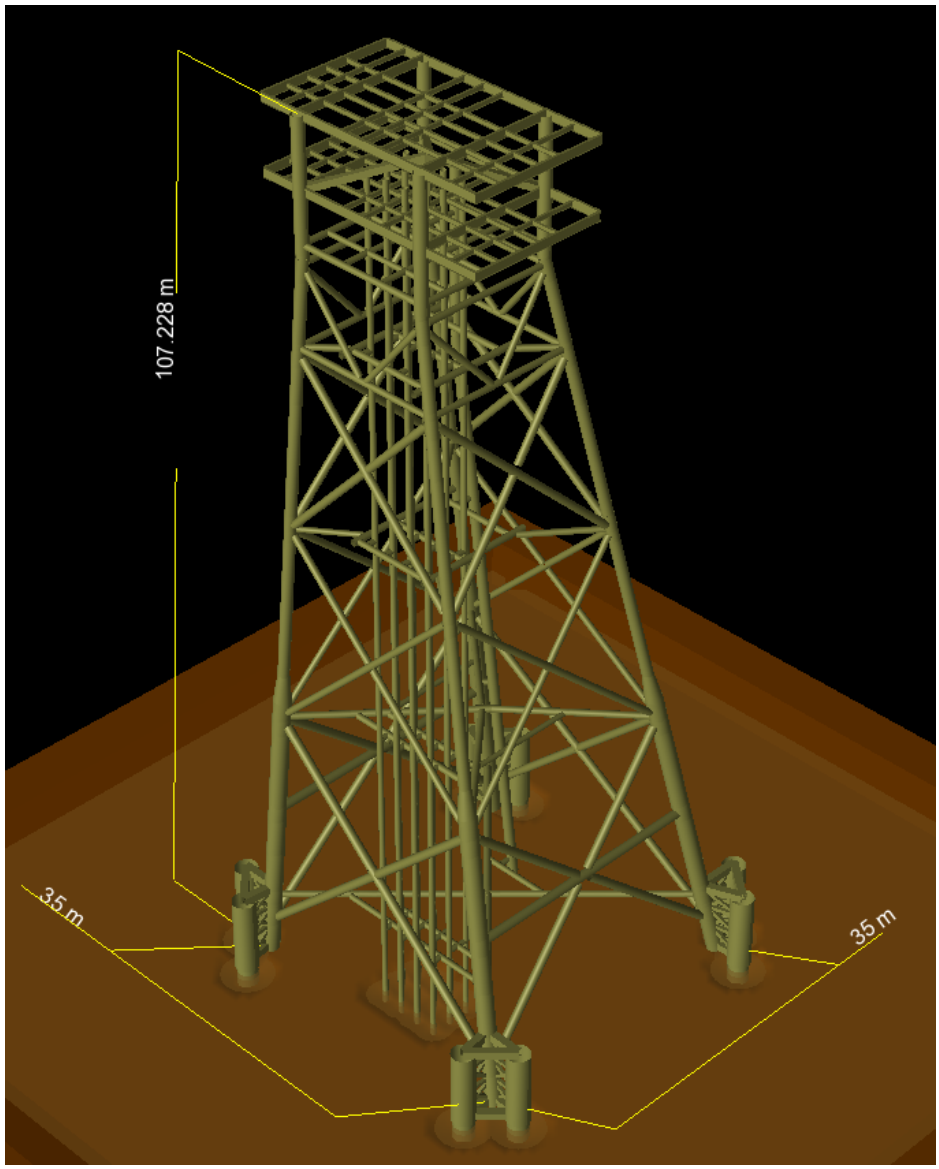


Figure 5.1: A visualization of a design of the Valhall Flank West jacket generated using the Sesam software.

5.2.2 Tests

This section presents the data obtained from the tests. Conclusions based on the results will be presented later in Section 6.1. All the tests were run with the same initial conditions, and the differences between the behaviour comes from the stochasticity of the algorithms. Table 5.3 lists all the presented test runs together with their computation time and information about individuals with best fitness. The GA tests produces individuals with better objectives values, but they were deemed infeasible due to utilization violations. The NSGA solution were feasible, but with significantly worse objective values. The values given in the tables in this section are rounded to the closest integer.

Test	Time (hh:mm)	Best fitness	Total + weld weight	Total utilization	Generation produced
GA1	12:20	3948	3173	765	140
GA2	11:40	3920	3164	745	138
GA3	11:00	3860	3119	734	137
NSGA1	12:40	4257	3502	754	148
NSGA2	13:00	4176	3440	735	134
NSGA3	12:00	4248	3484	763	129

Table 5.3: Overview of the tests performed, together with the individual with best fitness from each run.

Name	Fitness	Total + weld weight	Total utilization	Generation produced
NSGA1 W	4258	3502	756	147
NSGA1 U	4315	3567	747	109
NSGA2 W	4175	3440	735	134
NSGA2 U	4250	3524	725	147
NSGA3 W	4249	3484	765	97
NSGA3 U	4285	3529	756	139

Table 5.4: Individuals from the endpoints of the final fronts of NSGA tests with either minimized weight (W) or utilization (U).

Table 5.4 shows the individuals from the final fronts of the NSGA tests. This means that they have the lowest values of the respective objectives. When comparing with Table 5.3, it can be seen that NSGA2-W has the same values as the one with the best fitness from the NSGA2 test.

Figure 5.3 shows every individual in the objective space for every test with outliers removed for large values of both objectives. The outliers were removed at the third percentile of largest weight values and fifth percentile of largest utilization values. The color scheme presented in Figure 5.2 was used to color-code individuals based on which generation they entered the population. The individuals were plotted in such a way that if an individual from a later generation occupies the same space as another individual, the color of the earlier generation is shown on top. Crosses represent infeasible individuals, whilst dots represent the feasible ones. It can be seen that due to the strict constraint handling method of NSGA, infeasible individuals are only found in the early generations. Every test has shown a large spread of solutions in the early generations. GA can be seen converging more strongly towards single solutions, whilst NSGA shows a larger spread in the final generations compared to GA.

Figure 5.4 shows the last generation of every test. It shows that the GA1 and GA3 tests converged to a single solution, whilst GA2 ended up with two solutions. Meanwhile, as duplicate removal was implemented for NSGA, NSGA1 and NSGA2 show a single final front, whilst NSGA3 shows three fronts.

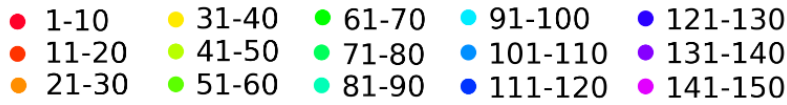
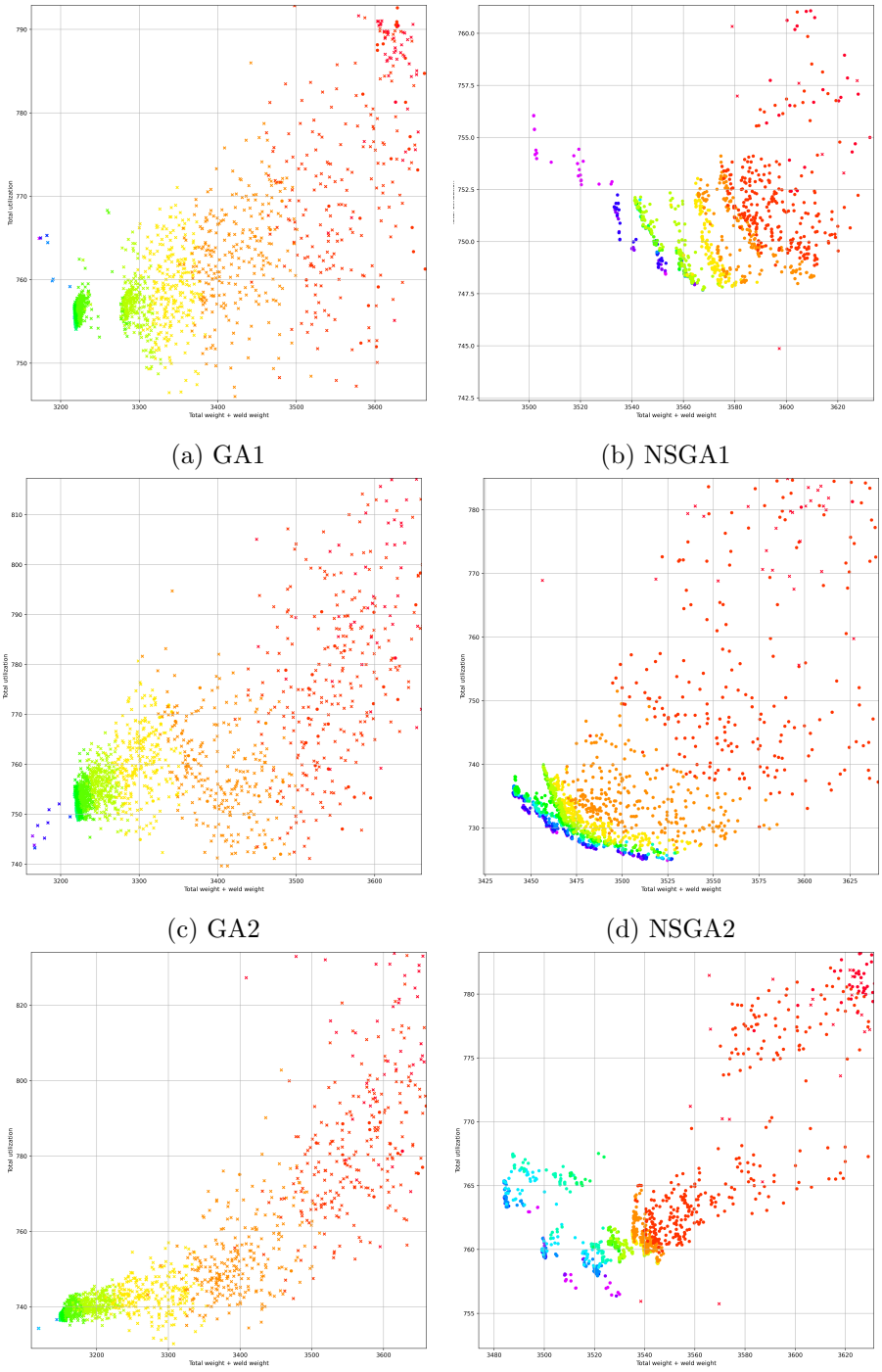


Figure 5.2: Color scheme used for groups of generations in Figure 5.3.



(a) GA1

(b) NSGA1

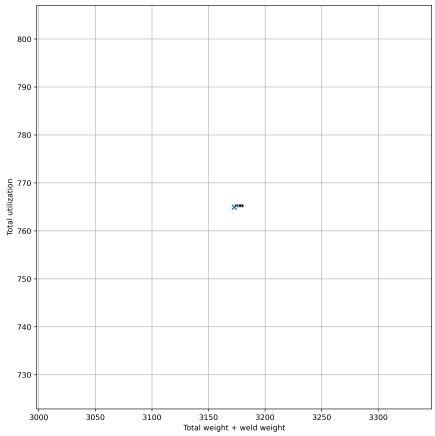
(c) GA2

(d) NSGA2

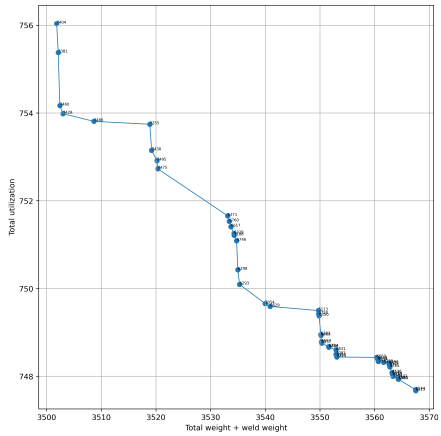
(e) GA3

(f) NSGA3

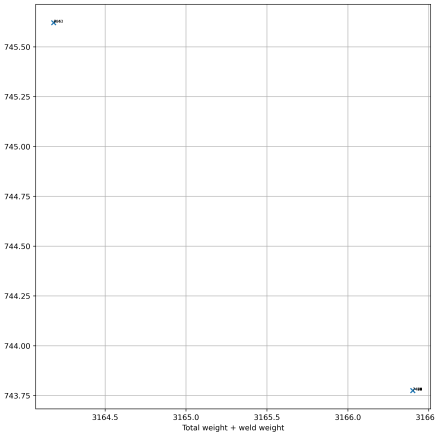
Figure 5.3: Graphs in the objective space over every individual from all generations for every test except outliers.



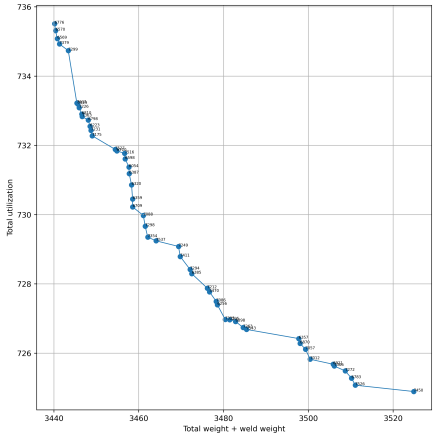
(a) GA1



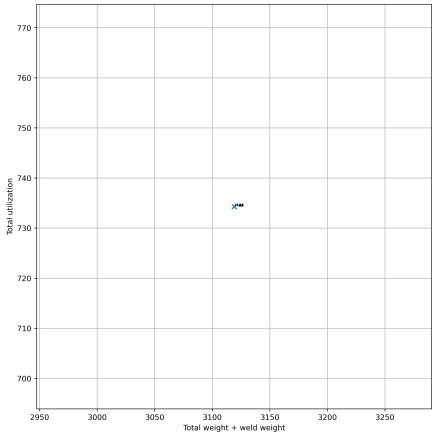
(b) NSGA1



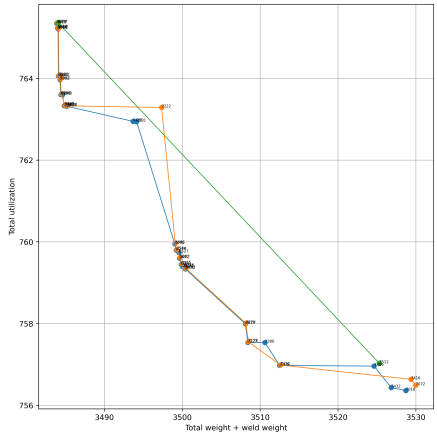
(c) GA2



(d) NSGA2



(e) GA3



(f) NSGA3

Figure 5.4: Graphs in the objective space over the last generation from every test.

Chapter 6

Conclusion

6.1 Discussion

The tests have shown drastic differences between the behaviour of GA and NSGA due to multiple reasons. The main factor was constraint handling. GA added penalty values for the broken constraints, which were low enough to deem infeasible solutions with lower weights to be better than feasible solutions. NSGA used the constraint handling method presented in Definition 2.2.7, which always prefers feasible solutions. As mentioned in Section 5.2.1, this scheme made it near impossible to navigate in the infeasible space. The explanation for that is that the constraint handling scheme approximates a weighted sum fitness approach with weights equal to 1. Another significant difference was that NSGA was adjusted by implementing duplicate removal, and thus the population always contained 50 individuals that differed at least slightly. Meanwhile, GA ended up with one or two unique individuals in their final populations. On the one hand this increases exploitation by producing more offspring of the best individual, but diversity suffers.

Preliminary smaller tests have been done throughout the project to gauge the behavior of the implemented NSGA. Originally, the two objectives used were total weight and weld weight, whilst utilization was barely used as a constraint, but this was deemed to produce little variation in individuals. A test was also performed with every term from the fitness

function in Listing 3.4 as an objective, but it produced too many infeasible individuals for any meaningful advancement. Utilization was once used as an objective to be maximized whilst keeping the per-element utilization below 1, as it is a common guideline during the jacket design process, but this also led to poor front advancement.

The test-improve approach could have been continued to produce better results. A meeting held late during the project with a jacket engineer, that had previously used the system, led to a conclusion that the system requires fundamental changes to make its use practical. Despite working closely with the analysis results from the Sesam software, and implementing multiple design constraints, it has been found that there are many more rules used during the design process that decide the feasibility of a jacket. As such, the focus has shifted towards other changes to the system, as described in the section below.

6.2 Future work

The most important issue to work on is the consideration of constraints that influence the design process. Whilst many rules are already implemented in the system, the list is vast and requires complete support in the system to produce useful designs. As the topic of jacket design is complex and requirements differ between projects, it is difficult to map out all the rules. A few examples of design constraints include [12]:

- Leg can diameter must equal diameter of leg above.
- Inside diameter for cans on braces must match inside diameter of braces.
- Bottom leg sections must widen to increase buoyancy.
- Legs should be splayed
- Braces coming into a leg should be separated by 100 mm.
- Brace diameter must be less than 0.95 * leg diameter.

An alternative approach would be to shift the focus of the system from optimizing complete designs to adapting it for use during the early stages

of projects. At that point, simplified designs are studied to decide the rough topology of a jacket. This would mean that significantly fewer constraints would have to be considered.

Another shortfall is that the system implements only a subset of analyses performed during jacket design. Examples of the missing analyses include transportation, launch, and accidental limit state. It is important to perform every analysis after making changes to a design, as adjustments aimed at improving one aspect of a design might decrease its performance with regards to other measures. Adding more analyses would increase the running time of tests considerably, but there is still room for improvement with regards to how evaluation is done. The OneCompute cloud solution employed during this project fared better than the previous Azure one, and it might be possible to optimize its use to reduce the current running time by up to 50%. This would make it more feasible to perform multiple analyses.

One way to possibly improve the behavior of NSGA would be to implement a hybrid approach which could use the GA to find feasible solutions that feed into the NSGA. For example, by having two populations, one for GA and one for NSGA, their efforts could be combined by having a common reinsertion step that lets the individuals with best fitness into the GA population, and feasible individuals with good objective values into NSGA. As it might be possible that minor constraint violations are not significantly detrimental to a design, the constraint handling method of NSGA could be made less strict by marking individuals with constraint violation below certain level as feasible.

Besides the changes mentioned above, a possible improvement would be to adjust the genotype to extend the search space. Currently, as mentioned in Chapter 3, the topology and shaping optimization relies on designs having parameters that influence these domains. An example of such a parameter is the base width which determines the angles of legs, and consequently the lengths of horizontal beams. This is a tight coupling between the genotype and phenotype representation. Removing this coupling would require a method that could produce a design based on the genotype, instead of appending overrides to existing design files. Additionally, dependencies between structural elements would have to be modelled appropriately in the system, so that adjustment of any gene would pro-

duce a valid structure. These mechanisms would make it possible to allow for designs that differ to a larger degree from the original design. On the other hand, it would put more pressure on ensuring that a solution adheres to design constraints.

In general, the system could benefit from employing more domain knowledge. Parameters are currently adjusted in a random fashion. By analyzing the evaluation results in depth by element, it might be possible use mutators in a guided way to adjust parameters towards beneficial values. An example would be to increase the size of elements with utilization violations. Correctness of such procedures would have to be ensured by expert knowledge.

Bibliography

[1] Eli Yecheskiel Kling, Bianca Ferri, Brede Bjørhovd, Frode Strand, Sigmund Mongstad Hope, et al. Automated jacket design. In *The 29th International Ocean and Polar Engineering Conference*. International Society of Offshore and Polar Engineers, 2019.

[2] David Goldberg et al. Genetic algorithms in search. *Optimization and Machine Learning, Reading, Massachusetts*, 1989.

[3] Dirk Thierens and David Goldberg. Convergence models of genetic algorithm selection schemes. In *International Conference on Parallel Problem Solving from Nature*, pages 119–129. Springer, 1994.

[4] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[5] Gustavo R Zavala, Antonio J Nebro, Francisco Luna, and Carlos A Coello Coello. A survey of multi-objective metaheuristics applied to structural optimization. *Structural and Multidisciplinary Optimization*, 49(4):537–558, 2014.

[6] N Srinivas and K Deb. Multiobjective optimization using nsga. *Evolutionary Computing*, 2(3), 1995.

[7] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[8] Kok Hon Chew, E.Y.K Ng, Kang Tai, Michael Muskulus, Daniel Zwick, et al. Structural optimization and parametric study of offshore wind turbine jacket substructure. In *The Twenty-third International Offshore and Polar Engineering Conference*. International Society of Offshore and Polar Engineers, 2013.

[9] Mohamed A El-Reedy. *Offshore structures: design, construction and maintenance*. Gulf Professional Publishing, 2019.

[10] Fabian Vorpahl, Wojciech Popko, and Daniel Kaufer. Description of a basic model of the" upwind reference jacket" for code comparison in the oc4 project under iea wind annex xxx. *Fraunhofer Institute for Wind Energy and Energy System Technology (IWES), Germany*, 2011.

[11] Great Britain. Dept. of Energy. *Offshore installations: Guidance on design, construction and certification*. HMSO, 1990.

[12] Frode Strand, Kvaerner. Genetic algorithms for jacket design optimalisation, lecture notes in it3708 bio-inspired artificial intelligence, February 2020.

[13] Rafal Kicinger, Tomasz Arciszewski, and Kenneth De Jong. Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers & structures*, 83(23-24):1943–1978, 2005.

[14] Wybren De Vries, NK Vemula, P Passon, T Fischer, D Kaufer, D Matha, B Schmidt, and F Vorpahl. Support structure concepts for deep water sites. *Delft University of Technology, Delft, The Netherlands, Technical Report No. UpWind Final Report WP4*, 2011.

[15] Jan Burak. Jacket optimization using a genetic algorithm. Project report in TDT4501, Department of Computer Science, NTNU – Norwegian University of Science and Technology, Dec. 2019.

[16] Yong Bai. *Marine structural design*. Elsevier, 2003.

[17] Dennis Lam, Thien Cheong Ang, and Sing-Ping Chiew. *Structural steelwork: design to limit state theory*. Crc Press, 2014.

[18] Akram AlSukker, Rami N Khushaba, and Ahmed Al-Ani. Enhancing the diversity of genetic algorithm for improved feature selection. In *2010 IEEE International Conference on Systems, Man and Cybernetics*, pages 1325–1331. IEEE, 2010.

[19] Lucía Bárcena Pasamontes, Fernando Gómez Torres, Daniel Zwick, Sebastian Schafhirt, and Michael Muskulus. Support structure optimization for offshore wind turbines with a genetic algorithm. In *ASME 2014 33rd International Conference on Ocean, Offshore and Arctic Engineering*. Citeseer, 2014.

[20] Sebastian Schafhirt, Daniel Zwick, Michael Muskulus, et al. Reanalysis of jacket support structure for computer-aided optimization of offshore wind turbines with a genetic algorithm. In *The Twenty-fourth International Ocean and Polar Engineering Conference*. International Society of Offshore and Polar Engineers, 2014.

[21] Johan Henrik Martens, Daniel Zwick, and Michael Muskulus. Topology optimization of a jacket structure for an offshore wind turbine with a genetic algorithm. In *11th World Congress on structural and multidisciplinary optimization. Sydney*, 2015.

[22] Johan Henrik Martens. Topology optimization of a jacket for an offshore wind turbine: by utilization of genetic algorithm. Master's thesis, NTNU, Institutt for bygg, anlegg og transport, 2014.

[23] Jan Häafele, Raimund Rolfes, et al. Approaching the ideal design of jacket substructures for offshore wind turbines with a particle swarm optimization algorithm. In *The 26th International Ocean and Polar Engineering Conference*. International Society of Offshore and Polar Engineers, 2016.

[24] Tawatchai Kunakote and Sujin Bureerat. Multi-objective topology optimization using evolutionary algorithms. *Engineering Optimization*, 43(5):541–557, 2011.

[25] Norapat Noilublao and Sujin Bureerat. Simultaneous topology, shape and sizing optimisation of a three-dimensional slender truss tower using multiobjective evolutionary algorithms. *Computers & Structures*, 89(23-24):2531–2538, 2011.

NTNU

Norwegian University of
Science and Technology