ADAPTIVE REPAIR METHOD FOR CONSTRAINT HANDLING IN MULTI-OBJECTIVE OPTIMIZATION BASED ON CONSTRAINT-VARIABLE RELATION

by

FAEZEH SAMANIPOUR

BSc, Sharif University of Technology, 2017

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE
in
THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Mechanical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

February 2020

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Adaptive Repair Method for Constraint Handling in Multi-objective Optimization Based on Constraint-Variable relation

submitted by	Faezeh Samanipour	in partial fulfilment of the requirements
for the degree of	Master of Applied Science	
in	Mechanical Engineering	

Examining Committee:

Dr. Jasmin Jelovica

Supervisor

Dr. Ryozo Nagamune

Supervisory Committee Member

Dr. Omar Swei

Supervisory Committee Member

Abstract

Evolutionary algorithms are popular tools for optimization of both theoretical and real-world problems due to their ability to perform global search and to deal with non-convex, multi-objective problems. Handling the constraints is a major concern in optimization that can prolong the search or prevent the algorithm from convergence. Common approaches for constraint handling usually discard or devalue infeasible solutions, losing the valuable information they carry. Alternatively, common repair methods for constraint handling are limited to specific problem types. This study focuses on the development of a repair method for constraint handling in multi-objective optimization.

A generic approach is proposed for improving the constraint handling. The method identifies infeasible solutions with high-quality objective values or small constraint violations. These solutions are modified to make them feasible while preserving their good position in the objective space. The repair is performed based on the relationship between constraints and variables in the problem. Variables causing infeasibility are replaced with values from other solutions. The number of repaired solutions varies during optimization. The remaining part of the solution set is created by usual operators to preserve the diversity and normal procedure of the algorithm.

The proposed repair method is applied to NSGA-II as one of the most commonly used multi-objective algorithms. The algorithm is tested on an optimization benchmark test case and an engineering optimization problem involving the structural design of a product tanker. The performance of the proposed approach is compared to the original algorithm and a few other constraint handling methods. Also, a competitive evolutionary algorithm, MOEA/D, is used for validation of the results. The proposed method showed faster convergence to the Pareto frontier and better diversity, covering the highly constrained regions of the design space. Additionally, the proposed algorithm was successful in reaching feasible solutions much faster, which is important in the case of computationally expensive problems, a common situation in engineering.

Lay summary

Optimization is finding the best possible solution of a problem considering the available resources and different limitations and restrictions. Optimization of real-world problems faces many challenges and the performance of the optimization algorithms needs to be improved for dealing with these complex problems. An approach is proposed in this thesis to improve the performance of one of the most famous optimization algorithms for obtaining high-quality solutions with less computational effort. The method is tested on a benchmark problem and a tanker hull structural optimization problem. The results demonstrate improved performance of the algorithm throughout the optimization process.

Preface

Dr. Jasmin Jelovica performed the definition and design of this research project.

For all chapters, the research work, numerical simulations and analysis is done by me.

Part of the results in Chapter 4 which examined the effect of repair method on optimization of the tanker hull structure was published as a conference paper at the 7th International Conference on Marine Structures held in Dubrovnik, Croatia in May 2019:

- Faezeh Samanipour, Jasmin Jelovica, “Improving structural optimization using a novel constraint-handling method in multi-objective genetic algorithm”, MARSTRUCT, 7th International Conference on Marine Structures, May 6-8, 2019, Dubrovnik, Croatia.

Further, a journal paper is published in the journal of Applied Soft Computing, based on the results in chapter 4:

- Faezeh Samanipour, Jasmin Jelovica, “Adaptive repair method for constraint handling in multi-objective genetic algorithm based on relationship between constraints and variables”, Applied Soft Computing, Vol. 90, 2020.

Table of contents

List of tables

List of figures

List of abbreviations

CB	Coupled Beam
FEM	Finite Element Method
FR	Feasibility Ratio
GA	Genetic Algorithm
Gen	Generations
HV	Hypervolume
MOEA/D	Multi-Objective Evolutionary Algorithm based on Decomposition
NSGA-II	Non-dominated Sorting Genetic Algorithm II
PSO	Particle Swarm Optimization
SA	Simulated Annealing
SBX	Simulated Binary Crossover

List of symbols

Bi	Neighboring subproblems
c_1, c_2	Acceleration coefficient
CV_i	Solution's constraint violation
CV_{mean}	Population's average constraint violation
d^i	Step size in coordinate i
D	Set of donor solutions
$f(x)$	Vector of objective functions
F_i	Non-dominated front
$F(x)$	Fitness function
g^{te}	Tchebycheff fitness value
$g(x)$	Vector of constraint functions
H(x)	Hessian matrix, second partial derivatives of function f
K	Boltzmann constant
l	Number of constraint functions
m	Number of objective functions
n	Number of variables
N	Population size
N_R	Maximum repair set size for Repair2
N_1	Repair set size for Repair1a
N_2	Repair set size for Repair1b
p_c	Probability of crossover
p_m	Probability of mutation
P	Parent population
$P(x)$	Penalty function
q	Number of the violated constraints
Q	Children population
r_1, r_2	Random number
R	Repair candidates for Repair2
R_1	Repair candidates for Repair1a

R_2	Repair candidates for Repair1b
T	Neighborhood size
T	Cooling control factor
w	Inertia weight
x	Vector of variables of a solution
$x_{\min,i}$	Lower variable bound
$x_{\max,i}$	Upper variable bound
x*	Solutions of the Pareto frontier
X	Design space
Y	Objective space
z	Ideal point
$\wp$	Variable permutations
Ω	Feasible space
Γ	Infeasible space
$\widehat{\Omega}$	Pareto frontier
Δ_k	Pattern direction
μ	Step size coefficient
d_k	Search direction
∇f	Gradient of function f
v	Particle velocity
Ƥ	Probability measure
ΔE	difference in the objective or energy
ω_j^i	Non-dominated front j in generation i
η_c	Crossover polynomial distribution index
η_m	Mutation polynomial distribution index
λ	Weight vector
τ	Allowable violation threshold
a	Infeasible solutions share
γ_j^i	Infeasible non-dominated front j in generation i
α^i	Infeasible solutions dominating ω_1^i

Acknowledgments

Firstly, I want to express my deepest gratitude to my parents and my sister for their love and continuing support throughout my study and research work far from home.

Major thanks to my supervisor Dr. Jasmin Jelovica for his constant guidance and encouragement. Without his support and guidance, I would not have been able to complete this research work.

Many thanks to my friends, research group members and UBC NAME group for their support and valuable feedback.

Lastly, this work was supported by Natural Sciences and Engineering Research Council of Canada [grant number 514711–17] and the start-up grant from The University of British Columbia which are highly appreciated.

1 Introduction

1.1 Optimization formulations and definitions

Optimization means discovering the best possible solution of a problem due to the available resources, limitations and restrictions of the problem. An optimization problem is mathematically formulated as maximizing or minimizing one or more functions called the objective function(s) by selecting a vector of variables. While finding the optimum value for the objective function, the variables may also be required to satisfy some restricting functions called constraints. The optimization problem can be mathematically formulated as:

$$\begin{aligned} &\text{Minimize} && \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \dots, f_m(\boldsymbol{x}))^T \\ &\text{Subject to} && \boldsymbol{g}(\boldsymbol{x}) = (g_1(\boldsymbol{x}), \dots, g_l(\boldsymbol{x}))^T \geq 0 \\ &\text{while} && \boldsymbol{x} = \left\{(x_i, \dots, x_n)^T \middle| x_{\min,i} \leq x_i \leq x_{\max,i}\right\} \end{aligned} \quad (1)$$

Where $\boldsymbol{f}(\boldsymbol{x})$ is the set of *m* objective functions and $\boldsymbol{g}(\boldsymbol{x})$ is the set of *l* constraint functions, and $\boldsymbol{x}$ is the vector of variables. Maximization of an objective function can be performed using the same approach if the objective values are multiplied by -1. Each variable x_i can be bounded in a lower and upper limit $x_{\min,i}$, $x_{\max,i}$. An optimization problem can have multiple local optima with relatively extreme objective values in the objective space. The global optimum is the absolute best solution of the problem. Variables can be continuous or discrete depending on the nature of the problem. A vector of variables often called a solution or a design is acceptable if it is feasible which means it satisfies all the constraint functions. Feasible solutions are members of the feasible set $\boldsymbol{\Omega}$.

$$\boldsymbol{\Omega} = \{\boldsymbol{x} \in \mathbf{X} | \boldsymbol{g}(\boldsymbol{x}) \geq \mathbf{0}\} \quad (2)$$

$\boldsymbol{X}$ is the set of all possible solution alternatives formed on variable permutations $\wp$ between their lower $x_{\min}$ and upper $x_{\max}$ bounds:

$$\mathbf{X} = \{\wp(x_1, \dots, x_n)^T | x_{\min} \leq x \leq x_{\max}\}. \quad (3)$$

Infeasible solutions are solutions from design space **X** that do not satisfy all the constraints. Infeasible solutions are members of infeasible set **Γ**:

$$\boldsymbol{\Gamma} = \{\boldsymbol{x} \in \mathbf{X} | \exists\ g_j(\boldsymbol{x}) < 0\} \tag{4}$$

In most multi-objective optimization problems, objective functions are in conflict which means that there does not exist a single solution with optimum value but a set of feasible solutions with trade-off values for the objectives. The set of the optimum feasible solutions of the problem is called the Pareto frontier or Pareto optima $\widehat{\boldsymbol{\Omega}}$. Pareto frontier $\widehat{\boldsymbol{\Omega}}$ is the set of non-dominated solutions x^* in the entire feasible space. In a set of solutions like **Ω**, the non-dominated set $\widehat{\boldsymbol{\Omega}}$ is a set of solutions which none of its members are dominated by any of the other members in **Ω.** Solution x_1 dominates solution x_2 if both conditions below are true [1]:

1. x_1 is no worse than x_2 in all of the objectives.

2. x_1 is strictly better than x_2 in at least one objective

The Pareto frontier (with minimization definition for all the objectives) can be formulated as:

$$\widehat{\boldsymbol{\Omega}} = \{\boldsymbol{x}^* \in \boldsymbol{\Omega} | \nexists \boldsymbol{x}^k, f_j(\boldsymbol{x}^k) < f_j(\boldsymbol{x}), \forall j \in [1, m], \forall \boldsymbol{x}^k \in \boldsymbol{\Omega}, \forall \boldsymbol{x} \in \boldsymbol{\Omega} \backslash \boldsymbol{x}^*\}, \tag{5}$$

1.1.1 Optimization problem classifications

Optimization problems can be classified into different types based on the nature of the problem and its functions. Some of the common optimization problem types based on different aspects of the problem are explained below:

- Discrete and continuous optimization problems: In discrete problems, variables can only be assigned specific values. The value of the discrete variables can be integer, binary (0 or 1) or any specific fixed amount. For instance, whether or not a traveler is going to pass a destination or number of bars for supporting a structure. On the other hand, variables in a continuous problem may have any real value in the defined range of the variable [2].
- Linear and non-linear optimization problems: If all the objectives and constraints are linear functions of the variables, the problem is known as a linear optimization problem. The problem is called non-linear if even one of the objectives or constraints are nonlinear function of the variables [3].

- Multi-objective and single-objective optimization problems: In cases where there is only one objective function, the problem is a single-objective optimization problem. However, practical problems involve several conflicting objectives, e.g. cost and safety. Problems with more than one objective function are called multi-objective optimization problems [4].
- Constrained and unconstrained optimization problems: A problem that its variable values cannot be freely selected is called a constrained problem. If there are no constraints restricting the variable values the problem is called an unconstrained optimization problem.

1.2 Optimization methods

Similar to optimization problems that can be classified into different types, different optimization methods exist for solving optimization problems. In order to determine which algorithm is the most appropriate for solving a given problem, it is necessary to understand the paradigm of different optimization methods and their capabilities. The main classes of optimization methods are described in the following subchapters.

1.2.1 Direct search methods

Direct search methods are often easy to implement and applicable to many non-linear programs [5]. Unlike derivative-based methods that require information about the gradient and curvature of the functions, direct search methods only deal with the function values. Pattern Search Method which is one of the earliest and still most common direct search methods uses a series of patterns and exploratory moves for exploration and improvement in the objective values [6]. In a minimization problem with objective function f(**x**) as a function of a vector of variables **x** = (x_1, x_2…, x_n), the Pattern Search Method is programmed to improve the objective value from one iterative to the next by pattern moves. A move is called successful if a reduction in the objective value is observed [6]. The exploratory move is described as:

$$x_{k+1}^{i} = x_k^i + d_k^i \qquad (6)$$

Where k indicates the iteration, x^i is the i^{th} variable and d^i is the step size in i^{th} coordinate. If the move is successful the variable value is kept, otherwise, it would keep the previous iteration value. The procedure is performed in all variable coordinates and the final vector of variables is called

the base point. The vector difference between current and previous base point indicates the direction of pattern move [6].

$$\mathbf{x}' = \mathbf{x}_k + \Delta_k \quad (7)$$

$$\Delta_k = \mathbf{x}_k - \mathbf{x}_{k-1} \quad (8)$$

A new vector of variables $\mathbf{x}'$ is created based on the pattern direction Δ_k, and the base point $\mathbf{x}_k$. If improvement in the objective function is observed, the new point is stored as the new base point. If the pattern move fails, the algorithm restarts the exploratory moves. The procedure continues until no further improvement is achieved even with small step sizes [6]. Direct search methods are easy to implement but they are not applicable to problems with large discontinuities in the design space. Moreover, they are computationally very expensive for large scale problems and not intended for multi-objective optimization.

1.2.2 Derivative-based methods

Derivative-based methods are iterative algorithms moving from an initial point toward the optimum based on the derivatives of the optimization function [3].

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \mu_k \mathbf{d_k} \quad (9)$$

Where the vector of variables in iteration k+1 is improved by moving toward direction $\mathbf{d_k}$ (based on the derivatives of the function) in a step size coefficient μ_k. Different procedures are proposed in the literature to compute step size μ_k and search direction $\mathbf{d_k}$ such as Steepest Descent Method and Newton's Method [2] for unconstrained and sequential quadratic programming [7] for constrained problems. These methods require the gradient and Hessian matrices which are respectively the first and second order partial derivatives of a function. The gradient of function f(x), $\nabla f(x)$ is defined as:

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_1}(x) \;\; \frac{\partial f}{\partial x_2}(x) \;\; \ldots \;\; \frac{\partial f}{\partial x_n}(x)\right] \quad (10)$$

And the matrix of second partial derivatives of function f(x), Hessian matrix H(x) is:

$$H(x)=\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \quad (11)$$

A solution x* is optimum if it satisfies necessary and sufficient conditions. The necessary condition for a point to be optimum is that the gradient of the function at that point is equal to zero; $\nabla f(x^*) = 0$. Also, in a minimization problem, the sufficient condition for optimality of point x* is Hessian matrix to be positive definite at that point [3].

Derivative-based methods are relatively robust and fast when dealing with smooth differentiable functions but they usually converge to local optima of the problem and are dependable on the choice of the starting point [8].

1.2.3 Evolutionary/nature-inspired methods

Nature-inspired methods only require function values and not their gradients so continuity or differentiability of the functions are not a concern in these methods. This feature allows these methods to be suitable for different problems with continuous, integer or binary variables in convex or non-convex objective spaces. Due to the randomness in their nature, they are more successful in finding global optima [9].

The problem of nature-inspired methods is that they are relatively slow in convergence compared to gradient-based methods. Evolutionary algorithms usually work with a population of solutions instead of a single solution as in most classical optimization methods. This provides a better potential for global search and also makes these algorithms suitable for multi-objective problems.

Particle swarm optimization algorithm

Particle swarm optimization (PSO) algorithm is based on swarm theory inspired by the social behavior of animals like bird flocking and fish schooling [10]. In nature, each individual in the swarm depends on both its own and swarm intelligence for finding food. If an individual finds out that another member in the swarm has a better path towards food, it will change its direction to follow the better path. In PSO, particles (individuals) start with random initial velocity vectors.

The particles change their location based on their own record and overall swarm experience. Each particle preserves the memory of its best position, *pBest* and the best solution found by the entire swarm, *gBest*, until the current iteration. Supplying the required memory is an important concern in the PSO algorithm. The algorithm should provide a balance between the local and global information to allow itself exploring different regions in the space while keeping the tendency toward the successfully found regions. The velocity of a particle in iteration *t+1* is calculated as:

$$v(t+1) = w(t) \times v(t) + c_1 \times r_1(gBest - x(t)) + c_2 \times r_2(pBest - x(t)) \quad (12)$$

And the position is updated to:

$$x(t+1) = x(t) + v(t+1) \quad (13)$$

Where w , $c_1$and c_2 are control parameters and r_1 and r_2 are random numbers [11].

Simulated annealing

Simulated annealing (SA) algorithm was first proposed for optimization by Kirkpatrick et al. (1983) [12]. The algorithm is inspired by the statistical mechanics where a material is heated and then gradually cooled down in order to obtain a crystalline structure with high strength. The method proposed an analogy between optimization concept and annealing process. The goal in annealing is to reach the lowest energy state with minimum entropy production. SA starts with a random initial solution (state). In each iteration, a new random solution (change in the state) is created by an adopted strategy. The current and new solutions are compared. If the new solution is better, it will be accepted and it will replace the current solution. If the new solution is not better in the objective value, the algorithm allows the solution to be preserved by a probability measure Ƥ (for minimization problem):

$$\text{Ƥ} = exp\left(\frac{-\Delta E}{KT}\right) \quad (14)$$

Where ΔE is the change in the objective function (change in energy), *K* is a constant parameter (Boltzmann constant) and *T* is a controlling factor in a given iteration (temperature) defined with a cooling schedule. Allowing the non-improving solutions to stay in the process by some probability helps the algorithm to escape local optima. SA is a memory-less algorithm that only works on current information of the solution and it is a single-solution method. SA is easy to implement, however, it requires fine parameter tuning. It is suitable for problems with a great

number of local optima but in many problems, it can prolong the search process and even fail in reaching the optimum [11].

Genetic algorithm

Genetic algorithm (GA) initially proposed by Holland (1975), is inspired by Darwin's theory of natural selection. GA is a popular global search algorithm that has been extensively used in different domains of science and engineering [11]. GA starts with an initial set of solutions called population, often randomly located in the design space. Chromosomes and genes are common terms used in GAs referring to the solutions and variables. GA algorithms often represent chromosomes as strings of binary genes but real-coded GAs are also developed and widely used in the literature. The algorithm is intended to make progress toward optimum by updating the population in each iteration. It creates a set of children solutions by combining the current solutions as their parents. As the theory of natural selection suggests, individuals with better characteristics have more chance to survive and reproduce. The main operators in GAs which are responsible for this process are selection, crossover and mutation. These operators are aimed to preserve better solutions in the population, combining parents to create new children and impose random changes in the children for providing diversity and better exploration in the problem space. Genetic algorithms are popular optimization tools due to their efficiency, global search, ease of use and applicability, and they have been widely used in different areas such as science, engineering and economy [1]. Two well-known GA algorithms for multi-objective problems NSGA-II and MOEA/D are further explained in Chapter 2. A recent literature review article [13] indicated that GA is still the most widely used optimization algorithm in engineering.

1.3 Constraint handling methods

Evolutionary optimization algorithms are originally defined for unconstrained problems. Constraint-handling is one of the main challenges and an active research topic in optimization [14]. In a constrained problem, in addition to optimizing the objective values, the algorithm has to focus on searching for solutions which satisfy the constraints. Constraints can cause disconnected regions and infeasibility holes in the design space, prolong the optimization process and causing premature convergence to local optima. Different approaches proposed in the literature to efficiently deal with constrained problems can be mainly classified into [14]–[16]:

Penalty functions

Penalty functions are the most common approach for handling the constraints. They convert a constrained problem to an unconstrained one by adding a penalty to the objective value based on the constraint violation. Hence the problem in (1), with the same variable bound condition would be switched to:

$$\text{Minimize} \quad \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{P}(\boldsymbol{x}) \qquad (15)$$

Where *F(x)* is called the fitness function that should be minimized and *P(x)* is the penalty function and has zero value when the solution is feasible. Different penalty functions are proposed in the literature that can be classified into Death penalty, static penalty, dynamic and adaptive penalty functions. Death penalty is the most simple method in which infeasible solutions with any amount of constraint violation are assigned the worst possible fitness and rejected from the population [14]. This will impose great computational cost for evaluating discarded solutions. Static penalty function uses constant penalty parameters to penalize the objective value of an infeasible solution [15]. Dynamic and adaptive penalty functions use time (generation), degree of constraint violation or infeasible to feasible solutions ratio as flexible controlling factors for penalty parameters [16]–[18]. Penalty approaches are relatively easy to implement but major issue for most penalty methods is that they require fine parameter tuning. Also, relations obtained for dynamic and static penalties are problem-dependent and hard to be generalized [14]. Penalty functions are not very practical for multi-objective problems since each objective should be properly penalized [19]. A high penalty would push the algorithm deeper in the feasible region and the algorithm may converge to a feasible solution which is far from the optimum. Conversely, a small insufficient penalty may result in failure in finding any feasible solutions [16].

Special operators for producing feasible solutions

For solving constrained optimization problems, specific operators may be designed to preserve the feasibility of solutions or moving in a specific part of the design space e.g. the constraints' boundary [20], [21]. GENOCOP [20] is one of the most well-known methods of this class that only deals with linear constraints and requires at least one initial feasible solution. Decoders are

the other type that maps the feasible search space to an easier-to-explore space for the optimization algorithm [14], [22], [23]. These special operators have shown competitive results in some cases, however, they are limited to very few applications and often require a set of initial feasible solutions which are difficult to obtain in problems with challenging constraints [14].

Separation of objectives and constraints

Unlike penalty functions that combine the constraint violation and objective values, this method handles constraint violation and objectives separately. This can be formed as coevolution, using two separate populations for optimization and feasibility which interact [24], or use of multi-objective concepts and considering the constraint violations as the new objectives of the problem to be minimized [25]. Although these methods are parameterless, they increase the complexities and computational costs of the optimization process. Another popular method in this class, first proposed by Deb [26], adds three feasibility criteria to the tournament selection of the solutions:

1. Comparing two feasible solutions, the one with better objective(s) is selected.
2. Comparing a feasible and infeasible solution, the feasible solution will always be preferred to the infeasible one.
3. Comparing two infeasible solutions, the one with lower sum of constraint violation is preferred.

The method is parameterless and robust for many problems but has shown premature convergence for problems where optimum lies on constraint boundaries due to the fact that it will not allow infeasible solutions to occupy part of the population [14].

Repair

When an infeasible solution is created, repair algorithm is aimed to direct it to feasibility. However, repair algorithms are mainly problem dependent and require comprehensive knowledge of the problem space [16]. Repair algorithms add an additional computational procedure to the algorithm. This extra computation should be considered for choosing a suitable repair method based on the application. Repair algorithms are usually problem dependent and mostly proposed in combinatorial optimization problems [27].

1.4 Optimization in engineering problems

Optimization is a common area in engineering for finding the best possible solution/design based on the available resources and limitations. Engineering often involves complex systems and large-scale problems. The functions are often nonlinear and the problems should satisfy different constraints. In many engineering problems, variables can only adopt discrete values due to market restrictions [28]. These would result in inconsistencies and gaps in the design space. Evaluation of the functions may not be possible analytically and in many cases, software simulations like finite element method and computational fluid dynamics should be linked to the optimizer [28]. Engineering problems are usually multi-objective since in decision making and design process different criteria have to be considered. Traditional optimization methods are not applicable in most cases for dealing with engineering and real-world problems. Derivative-based methods can get stuck in local optima and need derivative information of the functions. This means that the design space and functions should be continuous and differentiable which is not a common case in engineering. Due to the computational effort required for engineering problems, it is common to use parallel computing systems for optimization, however, this is not practical in most traditional methods that have to proceed point-by-point [28].

Evolutionary algorithms are popular for engineering problems since they can overcome the mentioned issues. Some examples from numerous cases of using the optimization algorithms for solving engineering problems can be found in [29]–[33]. GAs are popular optimization algorithms among engineers due to their capability of handling integer and continuous variables, possibility of parallel computing, handling multi-objective, constrained and highly nonlinear problems [34], [35]. Discovering the Pareto front of the problem typically needs global optimization methods to come close to the front and then using local search methods to find the exact set of Pareto solutions. This thesis focuses on improving global optimization algorithms and not on finding the true Pareto frontier thus global search is not followed by local. It is common that global optimization finds solutions very close to the true Pareto front.

1.5 Aim of the thesis

The main objective of this thesis is to develop a novel constraint handling technique which is not problem dependent and does not require an exact priori knowledge for transforming an infeasible solution to a feasible one. Challenging constraints in real-world problems increase the computational costs since the algorithm needs to spend many iterations to find feasible solutions. This can be a major issue in engineering problems which are already computationally expensive and also require using population-based algorithms. The aim is to improve the algorithm performance so that it can find high quality and acceptable solutions with less effort and reduce computational cost. The common constraint handling method which discards infeasible solutions ignores their valuable information. The proposed method is aimed to utilize the best infeasible solutions and modifying them slightly instead of totally rejecting them. The improvement in the efficiency of evolutionary algorithms is desired in two aspects:

- Faster convergence to the Pareto front
- Larger spread of the Pareto front

Faster convergence is important for computational costs and saving the time as a critical factor in the industry. Larger spread provides higher quality trade-offs for better decision making. Better convergence to the Pareto front provides more accuracy to the true optima of the problem. The proposed method is examined on a challenging structural engineering problem.

1.6 Thesis outline

The thesis is organized into five chapters. The present chapter is dedicated to a brief introduction to optimization and important topics of this area. Chapter 2 introduces original multi-objective evolutionary algorithms used in this study, their major concepts and parameters. Further, constraint handling methods used for validation and a performance metric used for comparison are presented in this chapter. Chapter 3 describes the proposed repair method for constraint handling and its implementation to the original optimization algorithm. Chapter 4 defines optimization case studies and their results obtained by the modified proposed algorithm are compared with the original algorithm and validated by other algorithms. Further, the chapter presents the influence of the

repair method on the solutions and the effect of the reduced feasible region. Finally, conclusion and future work are presented in Chapter 5.

2 Elitist multi-objective genetic algorithm

2.1 General

One of the most frequently used multi-objective optimization algorithms in engineering is NSGA-II, as shown fairly recently in a review publication [34]. It is popular in many research fields, even beyond engineering due to its robustness. The algorithm preserves the elite (best) solutions found throughout the search process while maintaining the diversity of the solutions. The mechanism and control parameters of the algorithm are described below.

2.2 Control parameters

The performance of the algorithm depends on the control parameters. These are population size, number of generations, mutation and crossover probabilities. An initial population can be manually provided by the user beforehand or it can be generated randomly using the uniform distribution of the variables between their upper and lower bounds. A random initial population may be completely infeasible which is potentially a challenging starting point for the algorithm but suitable for evaluating the capabilities of the algorithm.

Population size

Population is a set of different solutions that would lead the algorithm toward the optimum. Size of the population refers to the number of alternative solutions in the population. As the complexity of the problem increases due to the increase in the number of the variables, objectives and constraints or difficulties in the search space, the population size should be increased. Greater population size will provide a better possibility for exploring various parts of the space and lower risk in premature convergence to local optima. Also, a higher number of solutions allows the algorithm to better converge to the Pareto-frontier and maintain the diversity of the final solutions [1]. The cost to a greater population size would be increased computational effort.

Number of generations

The algorithm does iterative updates on the population by using the GA operators (Crossover, mutation and selection). These iterations repeat for a specific number called the number of

generations. After that, the algorithm stops and returns the last population as the final solution of the problem or saves the final population and re-runs with a new initial population. Setting the algorithm to run multiple times means that the algorithm does the optimization process for the number of runs and saves the final front of each. Commonly, the best solutions out of all the final solutions through all runs will be considered as the final Pareto front.

2.3 Non-domination sorting

The algorithm improves the set of solutions iteratively. In every generation, the algorithm performs the non-domination sorting and divides the population into multiple non-dominated fronts based on the principle in Eq. (5). Once one non-dominated front with the best objective values is found, the solutions in this set are removed from the population and the rest of the solutions go through the non-domination sorting to find the next non-dominated front. This procedure will continue until all the solutions in the populations are assigned to a certain non-dominated front. $\boldsymbol{\omega}_{(j)}^{(i)}$ in Figure 1 indicates the feasible solution of the non-dominated front *j* in generation *i*.

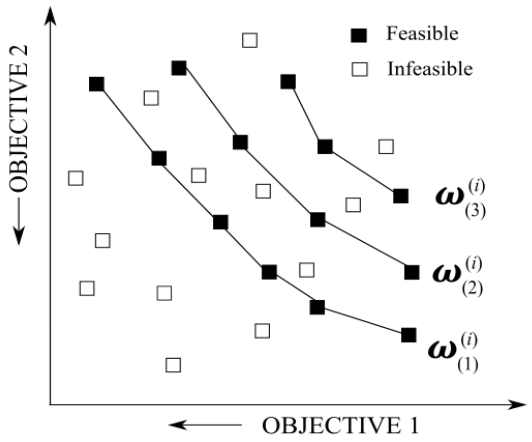

Figure 1. Non-dominated fronts of feasible solutions in the objective space.

Infeasible solutions are ranked based on the normalized constraint violation after the feasible non-dominated fronts. Solutions in each front are further sorted based on the crowding distance. The crowding distance metric is measured based on the density of the solutions around each particular

solution in the population. This quantity takes the average distance of a solution to the nearest neighboring solutions on either side of the front. As shown in Figure 2, the average side length of the largest cuboid enclosing solution i without containing other solutions is estimated as the crowding distance of the solution i.

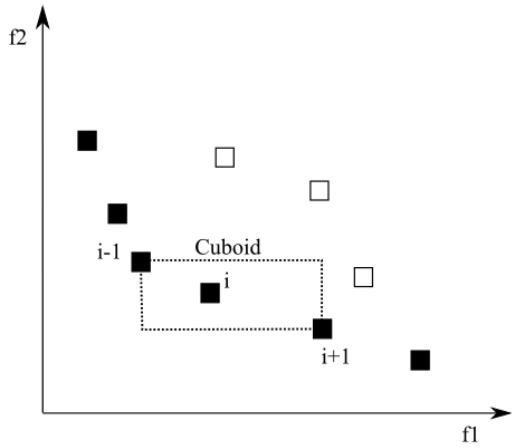

Figure 2. Crowding distance calculation for solution *i*.

The crowding distance of the solutions at the extreme points of the front is assigned as infinity. Solutions with larger crowding distance are favored since they lay on the less populated parts of the front and can help the algorithm to find a better spread of the final front and maintain the diversity of the solutions [36].

2.4 Genetic operators

Selection

For every generation, population first goes through the selection process. The selection operator chooses better solution among the two selected. Tournament selection is used here as reported to be better in convergence and has lower computational complexity [37]. Two solutions are randomly picked and the better one is chosen based on the non-domination rank. This will continue until the number of the selected solutions reaches the population size. The selection operator does not create any new solutions and only decides on the set of parents for the current generation. Creation of children is performed with crossover and mutation operators.

Crossover

Different crossover and mutation operators are presented in the optimization literature for both binary and real-parameter encodings. The crossover operator randomly picks two solutions from the population as the parent solutions to create two new children solutions by combining them. In this study, single-point and simulated binary crossover (SBX) methods [1] are used as crossover operators for binary and real-parameter problems, respectively. In the single-point crossover, the parents mutually exchange part of their strings to create new solutions. A random bit is selected as the crossing point and all the bits from the right side of that point are exchange between the two parent strings. SBX simulates the single-point crossover principles for real encoding. In this operator, the interval schemata between parents are preserved for children. The children are created from a polynomial combination of the two parents based on a distribution index η_c. Additional details and corresponding formulations can be found in Refs. [1], [38]. A new child created by the crossover operator is not necessarily better than its parents but the chance of creating a better solution is higher than random especially if the combining parent solutions are of good quality [1]. The probability of crossover, p_c, indicates the share of the population that are randomly chosen for going through the crossover operator.

Mutation

After selection and crossover operators, mutation operator is applied to the population. Mutation is intended to create small random changes after combination (crossover) in the population of solutions. Mutation provides better exploration in the objective space and diversity in the population. In binary-coded problems, solutions are strings of bits with values of either zero or one. The mutation operator flips the value of some of the bits based on the probability of mutation p_m. Similar to SBX operator, for real-coded problems, polynomial mutation is used which defines a polynomial distribution index, $\eta_{m,}$ in the variable bounds and creates a child from the parent by slightly changing it in the allowable variable space [1].

A high mutation rate expands the population cloud in the design space and provides a more comprehensive search. However, scattered solutions may distract the algorithm from moving toward the optima. On the other hand, low mutation shrinks the population cloud and results in solutions too close to each other which slows than the algorithm in searching the design space and

can result in premature convergence. Therefore, improper mutation rate wastes computational resources and prolongs the optimization process. Mutation rate is problem dependent, however, studies have shown that the general rule appropriate for most problems is 1/(number of bits) for binary and 1/(number of variables) for real coded problems which indicates altering the value of one variable in a solution[36].

2.5 Optimization procedure

After defining the problem and control parameters, the optimization process can be started. The initial population is created and assigned as the first parent population. For each generation t, the parent population P_t is evaluated in terms of objective values, constraint violation, non-domination rank and crowding distance. The population undergoes the GA operators (selection, crossover and mutation) for creating the children population Q_t. The children population is evaluated. NSGA-II is an elitist algorithm which means that the new children would only replace the parents if they are better. Therefore, the parent and children populations are combined to create an intermediate population of size 2N and sorted to a sequence of non-dominated fronts F_1, F_2,... The population of size N for the next generation is created from solutions of the intermediate population with the best front ranks. A front may need to be split due to the restricted size of the population. In this case, solutions with higher crowding distances are preferred. The new population will be considered as the parent population of the next generation and undergoes the same procedure as described. This routine is repeated for the number of generations. Figure 3 shows the schematic procedure of NSGA-II.

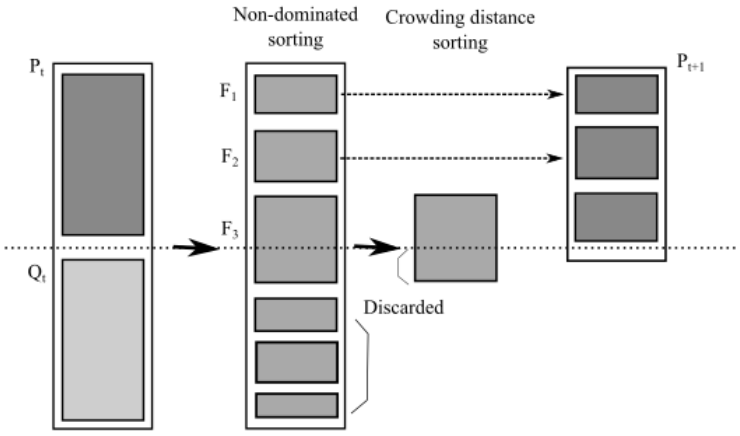

Figure 3. NSGA-II procedure.

2.6 Multi-objective evolutionary algorithm based on decomposition

The performance of NSGA-II is compared with multi-objective evolutionary algorithm based on decomposition MOEA/D [39]. It has been shown that MOEA/D outperforms NSGA-II in many case studies [39]–[41]. The decomposition-based algorithm evaluates solutions by converting the multi-objective problem into a number of single-objective problems. The Tchebycheff approach is used for decomposition as in Ref. [42]. The decomposition technique sets the population as N subproblems and assigns uniformly distributed weight vector (λ_i) to each subproblem. The Euclidean distances among the weight vectors are used for indicating T closest neighboring subproblems B*i* to subproblem *i*. Offspring reproduction and selection procedure of each subproblem is done regarding its neighborhood.

The scalar optimization in the Tchebycheff approach is defined as:

$$g_i^{te}(x|\lambda, z^*) = \min\{\lambda_j(\left|f_j(x) - z_j\right|)\} \quad (16)$$

Where g_i^{te} is the fitness value of subproblem *i*. z is the ideal point i.e. $z_j = \min\{f_j(x)|x\epsilon\boldsymbol{\Omega}\}$ for each j=1,..,*m*.

Similar to all evolutionary algorithms, MOEA/D originally is introduced for unconstrained problems. The adaptive constraint handling approach described in 2.7.1, has shown successful

results in dealing with constrained problems. MOEA/D is used as a validation for the results obtained by NSGA-II.

2.7 Constraint handling methods from literature

2.7.1 Adaptive threshold approach

This method for constraint handling in multi-objective evolutionary algorithms was introduced in [42] and validated in [43]. The method defines a threshold for constraint violation, where infeasible solutions with violation less than the threshold are allowed to remain in the population. The threshold is parameterless and adaptive. It is calculated and updated based on the solutions in the population. The allowable violation τ is calculated as:

$$CV_i = q * \sum_{j=1}^{l} \left|\min\left(g_j, 0\right)\right| \ , \tag{17}$$

$$CVmean = \frac{1}{N}\sum_{i=1}^{N} CV_i, \tag{18}$$

$$\text{Feasibility ratio (FR)} = \frac{Number\ of\ the\ feasible\ solutions}{Population\ size} \tag{19}$$

$$\text{Allowable violation threshold } (\tau) = CVmean \cdot FR. \tag{20}$$

Where q is the number of the violated constraints in the i^{th} solution. Solutions with violation less than the threshold τ would be treated as feasible and compared based on their objective values. Therefore, infeasible solutions with high-quality objective values and relatively small constraint violations are maintained in the population and passed to the next generation.

2.7.2 Deterministic infeasible sorting

As proposed in [32] for NSGA-II, a fixed share a of the elitist population in each generation is occupied with high-rank infeasible solutions. In each generation, after the population is evaluated, the infeasible solutions are sorted based on the non-domination principle while their constraint violation is ignored and only their objective values are considered. a of the infeasible solutions with best rank are combined with N-a best feasible solutions to create the final elitist population of the generation. The selection operator still favors the feasible solutions to the infeasible ones. If the size of the infeasible set is smaller than a, all the infeasible solutions are copied to the elitist populations.

2.8 Performance metric (hypervolume)

In multi-objective optimization, there are two main goals: convergence to the true optima and diversity of the solutions [28]. Convergence indicates the accuracy and closeness of the obtained solutions to the optimum front of the problem. Diversity indicates the distribution of the solutions along the front. Higher diversity of the solutions provides better trade-off in the objective space. For assessing the performance of optimization algorithms, these two aspects should be considered at the same time. The hypervolume performance metric, *HV*, is used in this study since it is based on both convergence and diversity of the solutions. Originally defined as "size of the space covered" in [44], it calculates the area (or volume) of the objective space which is enclosed by the obtained non-dominated solutions of the front and a reference point. As the quality of the solutions is improved, the enclosed area increases and so does the hypervolume metric. Figure 4 shows the hypervolume enclosed by the non-dominated solutions obtained in a minimization problem. The reference point is determined based on the worst value found for each objective among all the feasible solutions.

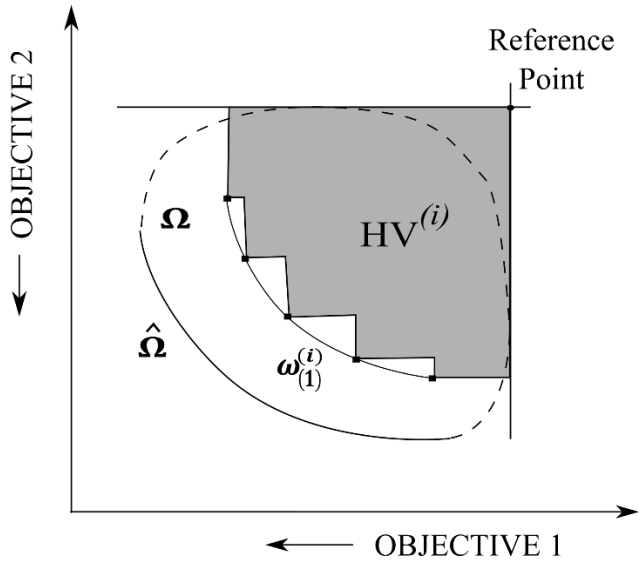

Figure 4. The hypervolume enclosed by the set of non-dominated solutions in the generation *i*

The approach is commonly used in the literature for performance evaluation of multi-objective optimization algorithms [32], [42], [43], [45]. With equal optimization parameters and computational cost, algorithms that obtain higher *HV*s are preferred. In the optimization problems

in this study, the reference point is obtained based on the largest (smallest) value found for a minimization (maximization) objective by feasible solutions through all the runs. Also, the objective values are normalized before hypervolume calculation due to the potential difference in the scale and unit of the objective values. When a population is completely infeasible, the hypervolume would be equal to zero.

3 Proposed constraint-handling method and implementation

3.1 Motivation

In addition to the usual non-dominated sorting on the feasible solutions, infeasible solutions can also be sorted based on the non-domination principle by ignoring their violation and only considering their objective values. Sorted infeasible solutions sets are denoted as $\boldsymbol{\gamma}_{(j)}^{(i)}$ where *i* indicates the generation number and subscript *j* indicates the rank of the non-dominated infeasible front. Infeasible solutions with better objective values than the first non-dominated feasible front are proper candidates for repairing. $\boldsymbol{\alpha}^{(i)}$ is the set of solutions in $\boldsymbol{\gamma}^{(i)}$ that dominate $\boldsymbol{\omega}_{(1)}^{(i)}$, shown in Figure 5.

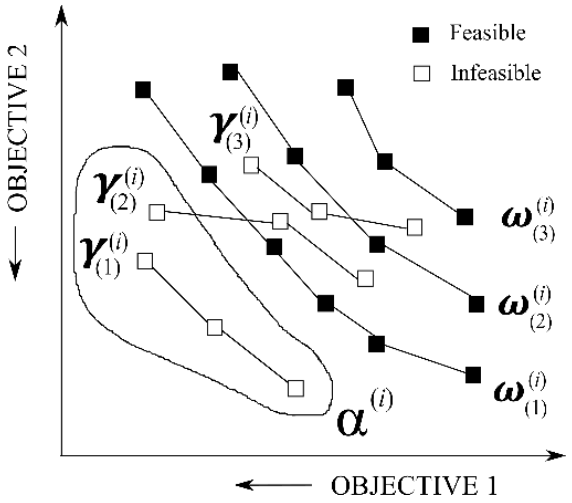

Figure 5. Possible location of sorted solutions in the population

It is very likely that in each generation, GA operators create some infeasible. The efficiency of the algorithm would be improved if the repair algorithm successfully manages to create feasible solutions with valuable objective values that dominate the existing feasible solutions in the population (Figure 6).

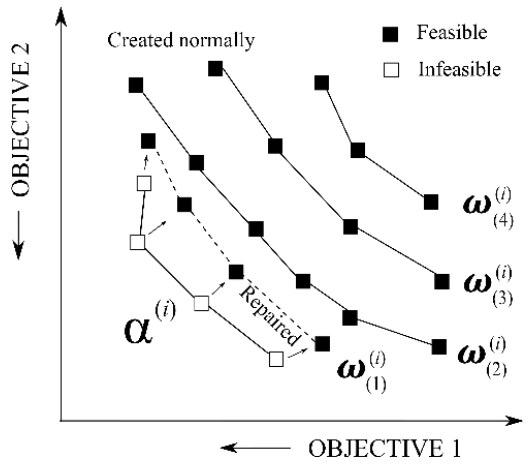

Figure 6. Situation where infeasible solutions after successful repairing dominate the current feasible solutions.

In any optimization problem, constraints are functions of the variables, i.e. $\boldsymbol{g} = \boldsymbol{g}(\boldsymbol{x})$. In most problems with available formulations of the constraints, this connection can be simply specified. Also, for problems which use numerical tools for assessment of the functions, such connection can be approximately determined. For instance, in structural design, stress-related constraints can be considered as functions of cross-section dimensions of the structure. When a constraint is violated, this connection allows us to flag the variables responsible for the violation. Establishing the connection between constraints and variables, infeasibility can be traced back to variables. The aim of the proposed method is to replace the value of such variables with the value of those variables in solutions which are not violating that specific constraint. Even if the population is completely infeasible, which is common at early generations, the approach is applicable. In a completely infeasible population, infeasibility of the solutions may be due to different constraints. Therefore, the solutions can play the 'donor' role if they are feasible for the constraint that a repair candidate is violating. In addition, in a case where all solutions are violating the same constraint, it is good practice to consider the solution with the lowest violation as the donor.

The proposed repair method is divided into two approaches that have different aims: 1. creating feasible solutions from completely infeasible population and 2. repairing infeasible solutions based on the feasible solutions in the population. These two approaches are described in the following.

3.2 Repair1 (repair of a completely infeasible population)

As mentioned above, in a completely infeasible population, the infeasibility of the solutions can be due to different constraints that each solution may have violated. The variable-constraint link is the guide for deciding on the variables that need to be changed. In repairing variables of a specific constraint, solutions that satisfy that constraint are the potential donors. In the case where all solutions are violating the same constraint, the solution with the least amount of violation is picked out as the donor solution. The idea behind repairing is to make small changes to the solution, yet keeping the valuable position of the solution in the objective space. For this purpose, the donors are prioritized from solutions in the best ranked non-dominated fronts and nearest to the candidate solution. Therefore, donors may be different based on the position of the repair candidate. Also, the donor is not necessarily the same for all infeasible variables of a solution. Algorithm Repair1 is described in the pseudo code below. Two groups of the solutions are selected to be repaired:

a) N_1 infeasible solutions with the lowest sum of the normalized constraint violation which potentially require smaller changes in their variables to become feasible (Repair1a).

b) N_2 infeasible solutions with the best performance in the objective space. For this group, the priority is given to solutions with firstly the best non-dominated rank and secondly larger crowding distances. These solutions have more potential to create solutions with competitive objective values and keep the diversity of the solutions in the fronts (Repair1b).

Repair1. Repairing completely infeasible population

$\boldsymbol{\gamma}^{(i)}$ = Set of infeasible solutions in the current population *i* (= complete population *i*)

Sort $\boldsymbol{\gamma}^{(i)}$ based on (a) the sum of normalized constraint violation, (b) non-domination rank (despite being infeasible) and crowding distance.

Start *Repair 1a*

Form the set of repair candidates $\boldsymbol{R_1}$, where $|\boldsymbol{R_1}| = N_1$, starting with the solutions which have the lowest sum of normalized constraint violation.

For each $\boldsymbol{R_1}$ member **do** (*Replace procedure*)

Calculate Euclidean distance in the normalized **Y** to other solutions in $\boldsymbol{\gamma}^{(i)}$.

Form the set of donor solutions $\boldsymbol{D}$: Select solutions which have the best non-domination rank and give priority to those with the lowest Euclidean distance; continue until whole population is in $\boldsymbol{D}$.

Flag infeasible variables using the defined connection between constraints and variables.

For each infeasible variable **do**

Replace infeasible variable with the corresponding feasible variable from the highest-ranked member of $\boldsymbol{D}$.

If the variable is infeasible for all members of $\boldsymbol{D}$, replace it based on the solution that has the smallest constraint violation caused by the variable.

End for

End for

End *Repair 1a*

Start *Repair 1b*

Form the set of repair candidates $\boldsymbol{R_2}$, where $|\boldsymbol{R_2}| = N_2$, starting with the solutions which have the best non-domination rank and the highest crowding distance value.

For each $\boldsymbol{R_2}$ member **do**

Repeat *Replace procedure* as above.

End for

End *Repair 1b*

Non-domination sorting and crowding distance calculations are performed following Deb et al. [8]. $N_1 + N_2 \leq N$ since the population size has to remain unchanged. The analysis of the test problems shows that these problems are not very sensitive to the parameters N_1 and N_2 as long as they are large enough (Appendix A).

Part of the population is created by normal GA operators to provide certain randomness and diversity necessary for the evolutionary algorithms. Crossover and mutation can significantly alter a solution, hence they are not applied to children made with the repair procedure.

Success of the repair method depends on the accuracy of the defined constraint-variable link. However, an approximate connection should also help the algorithm to get closer to feasibility which is helpful for the usual GA operators to find feasible solutions faster. After discovering feasible solutions, Repair2 is applied to the algorithm.

3.3 Repair2 (repair of a partially infeasible population)

Once feasible solutions are found, they form the set $\boldsymbol{\omega}_{(1)}^{(i)}$. In Repair2, the repair donor is from $\boldsymbol{\omega}_{(1)}^{(i)}$, a feasible non-dominated solution, which is closest to the infeasible repair candidate in the objective space. Therefore, it is more likely to create a feasible solution with small changes in the infeasible solution. N_R solutions are repaired at maximum and the rest of the population is created with the usual GA operators. The repair candidates are chosen from infeasible solutions that dominate the best feasible solutions. If the set $\boldsymbol{\alpha}^{(i)}$ is empty, it means that there are no infeasible solutions better than the present feasible solutions in the population and the whole children population would be created by GA operators. If $\boldsymbol{\alpha}^{(i)}$ is not empty, Repair2 is applied and the rest of the children (N - N_R) are created using normal procedure; tournament selection, crossover and mutation. As with the Repair1, non-domination sorting and crowding distance calculation are performed following Deb et al. [36]. The pseudo code of Repair2 is described below.

Repair2. Repairing partially infeasible population

$\boldsymbol{\omega}_{(1)}^{(i)}$ = The (first) non-dominated front in the current parent population
$\boldsymbol{\gamma}^{(i)}$ = Set of infeasible solutions of the population
$\boldsymbol{\alpha}^{(i)}$ = Members of $\boldsymbol{\gamma}^{(i)}$ that dominate $\boldsymbol{\omega}_{(1)}^{(i)}$ (= Potential repair candidates)
If $\left|\boldsymbol{\alpha}^{(i)}\right| > N_R$ **then**

Rank solutions in $\boldsymbol{\alpha}^{(i)}$ based on (a) non-domination and (b) crowding distance.

Form the set of repair candidates $\boldsymbol{R}$: Select solutions which have the best non-domination rank and give priority to those with the largest crowding distance; continue until $|\boldsymbol{R}| = N_R$.

Else

$N_R = \left|\boldsymbol{\alpha}^{(i)}\right|$ and form repair candidates set $\boldsymbol{R} = \boldsymbol{\alpha}^{(i)}$.

End if
For each $\boldsymbol{R}$ member **do**

Flag infeasible variables using the defined connection between constraints and variables.

Calculate Euclidian distance in the normalized **Y** to neighboring feasible solutions in $\boldsymbol{\omega}_{(1)}^{(i)}$.

Replace infeasible variables using values from the closest feasible solution in $\boldsymbol{\omega}_{(1)}^{(i)}$.

End for

3.4 Implementation of proposed repair method to NSGA-II

The proposed constraint handling method is implemented into NSGA-II as one of the most common multi-objective optimization algorithms for various problems. NSGA-II has shown successful results in reaching the well-spread true Pareto frontier of test benchmark problems [1], optimization of truss structures [46] and building optimization problems [34]. The important characteristics of this algorithm are the non-domination and crowding distance sorting in addition to the elitist based behavior of the algorithm. As described earlier, in each generation, the newly created children population is compared with the current population (parents) and the bests of the two groups are selected. This approach steadily leads the search toward the optimum of the problem.

The repair algorithm would only make changes in the children population which means that the repaired solutions occupy part of the children population set while the rest of the children are created as usual. Depending on whether a parent population is completely infeasible or partly feasible, repair algorithm 1 or 2 would be active, respectively. The children would be evaluated and then the intermediate population is created from the union of the children and parent population. The intermediate population is sorted and N best solutions will be chosen as the new population to pass to the next generation. The repair candidates are determined from the intermediate population as a bigger available pool and passed to the next generation to occupy part of the children population. Figure 7shows the modified algorithm.

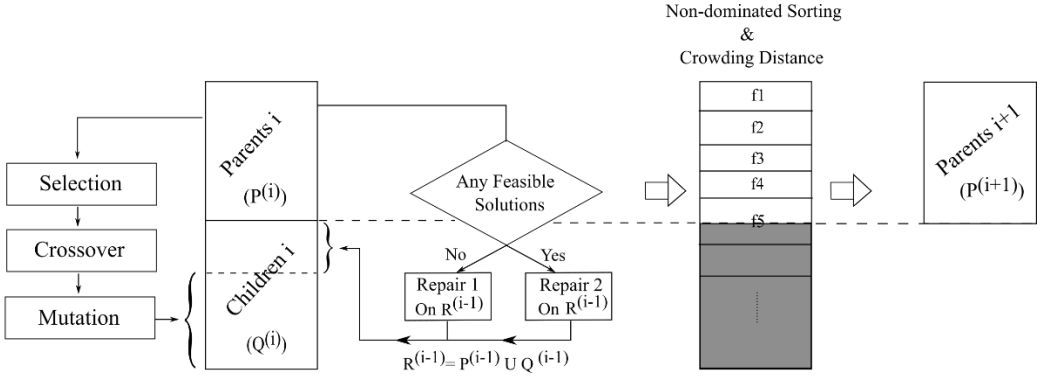

Figure 7. Modified NSGA-II procedure with repair algorithm

4 Test cases

4.1 OSY problem

The proposed algorithm is at first tested on OSY optimization benchmark problem before proceeding with a more challenging problem. This problem was first proposed by Osyczka and Kundu [47] and has been used for evaluating the performance of optimization algorithms in many studies as a challenging multi-objective constrained problem [1], [42], [48]. The OSY problem is defined as:

$$\min \begin{Bmatrix} f_1(\boldsymbol{x}) = -25(x_1-2)^2-(x_2-2)^2-(x_3-1)^2-(x_4-4)^2-(x_5-1)^2 \\ f_2(\boldsymbol{x}) = \sum_{i=1}^{6} x_i^2 \end{Bmatrix}$$

$$s.t. = \begin{cases} g_1(\boldsymbol{x}) = x_1 + x_2 - 2 \geq 0 \\ g_2(\boldsymbol{x}) = 6 - x_1 - x_2 \geq 0 \\ g_3(\boldsymbol{x}) = 2 + x_1 - x_2 \geq 0 \\ g_4(\boldsymbol{x}) = 2 - x_1 + 3x_2 \geq 0 \\ g_5(\boldsymbol{x}) = 4 - (x_3-3)^2 - x_4 \geq 0 \\ g_6(\boldsymbol{x}) = (x_5-3)^2 + x_6 - 4 \geq 0 \end{cases} \quad (21)$$

$$0 \leq x_1, x_2, x_6 \leq 10,$$
$$1 \leq x_3, x_5 \leq 5,$$
$$0 \leq x_4 \leq 6.$$

The problem includes two objectives and six constraints. The variables are continuous and their bounds are defined as above. The constraint-variable connection is easily defined as each constraint is an explicit function of two variables. The Pareto frontier of OSY is located on the boundaries of the feasible space [48] which makes this problem suitable for evaluating the performance of the proposed repair method. Optimization parameters are defined in Table 1. The polynomial mutation and SBX operators' indexes are set to 20 and mutation probability is set to (1/Number of variables) as recommended for this problem in the literature [48]. The population and generation parameters are assigned different values in the literature for this problem. This is not uncommon for benchmark problems since they are relatively inexpensive in terms of computational time. In comparing the algorithms, it is important to keep the parameters the same for both algorithms. In this study, the population size of 100 and 200 generations are chosen based

on the preliminary analysis presented in Appendix A. 100 independent runs are performed for each algorithm with randomly generated initial populations. Size of the repair sets, N_1 = 35, N_2 = 35 and N_R = 10 are assigned based on the preliminary study presented in Appendix A.

Table 1. Optimization parameters for OSY problem

N	Generations	Runs	p_c / η_c	p_m / η_m	N_1	N_2	N_R
100	200	100	0.5/20	0.167/20	35	35	10

Figure 8 presents the final fronts obtained by the original and proposed NSGA-II and MOEA/D for comparison. As shown in this figure, all algorithms were successful in finding the Pareto front with sufficient number of runs and iterations.

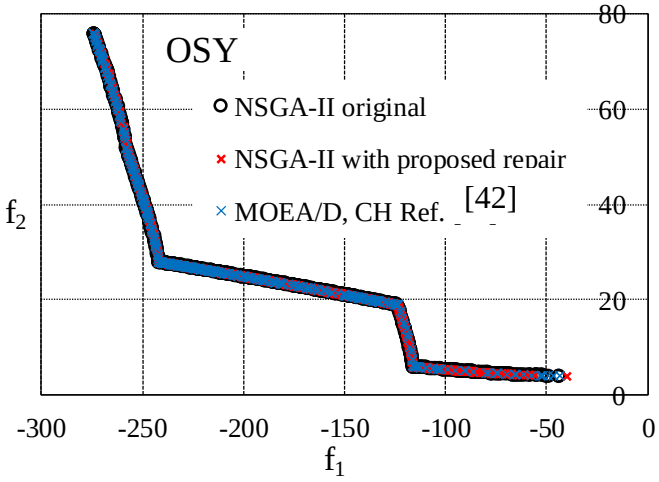

Figure 8. The final non-dominated fronts of OSY, considering all independent runs.

Median hypervolumes of the fronts obtained by the algorithms through the generations based on the 100 runs are presented in Figure 9. The results are also compared with the performance of NSGA-II with two state-of-art constraint handling methods described in subchapter 2.7. The proposed algorithm has better performance especially at the beginning of the search. The NSGA-II results converge near the end of the search process. MOEA/D starts with less competitive results initially but ends up with slightly better results.

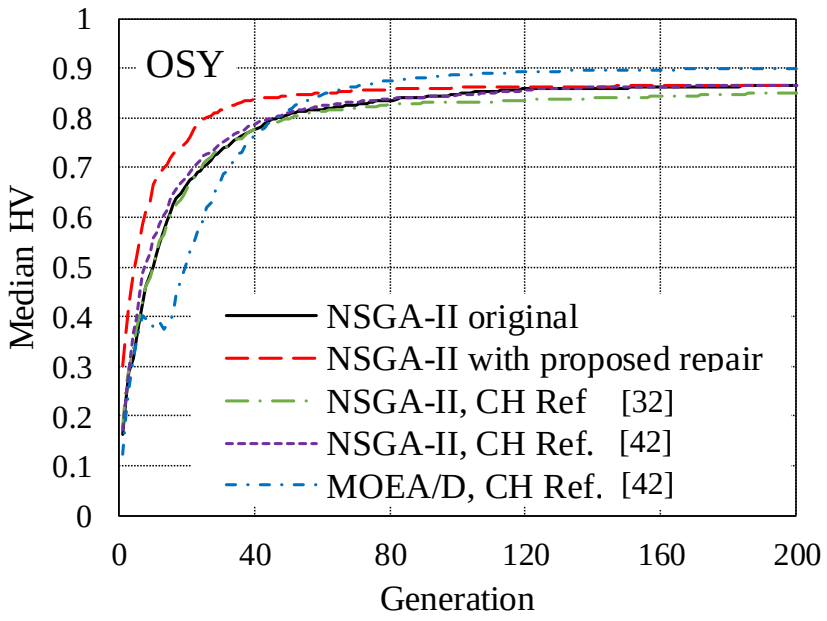

Figure 9. Median hypervolume performance metric for OSY problem through generations

4.2 Structural optimization of a tanker

The problem is defined as the optimization of the longitudinal structural members of a tanker hull that need to sustain normal service loads. The goal is to minimize the mass of the structure and maximize the safety in the deck. The tanker has a typical chemical/product layout shown in Figure 10, with length 180 m, breadth 32 m, depth 18 m, and a draught of 11.5 m. Problem modeling, optimization and result analysis are conducted using MATLAB. The midship section of the tanker is considered for optimization since the structure is symmetric about the centerline ($y = 0$).

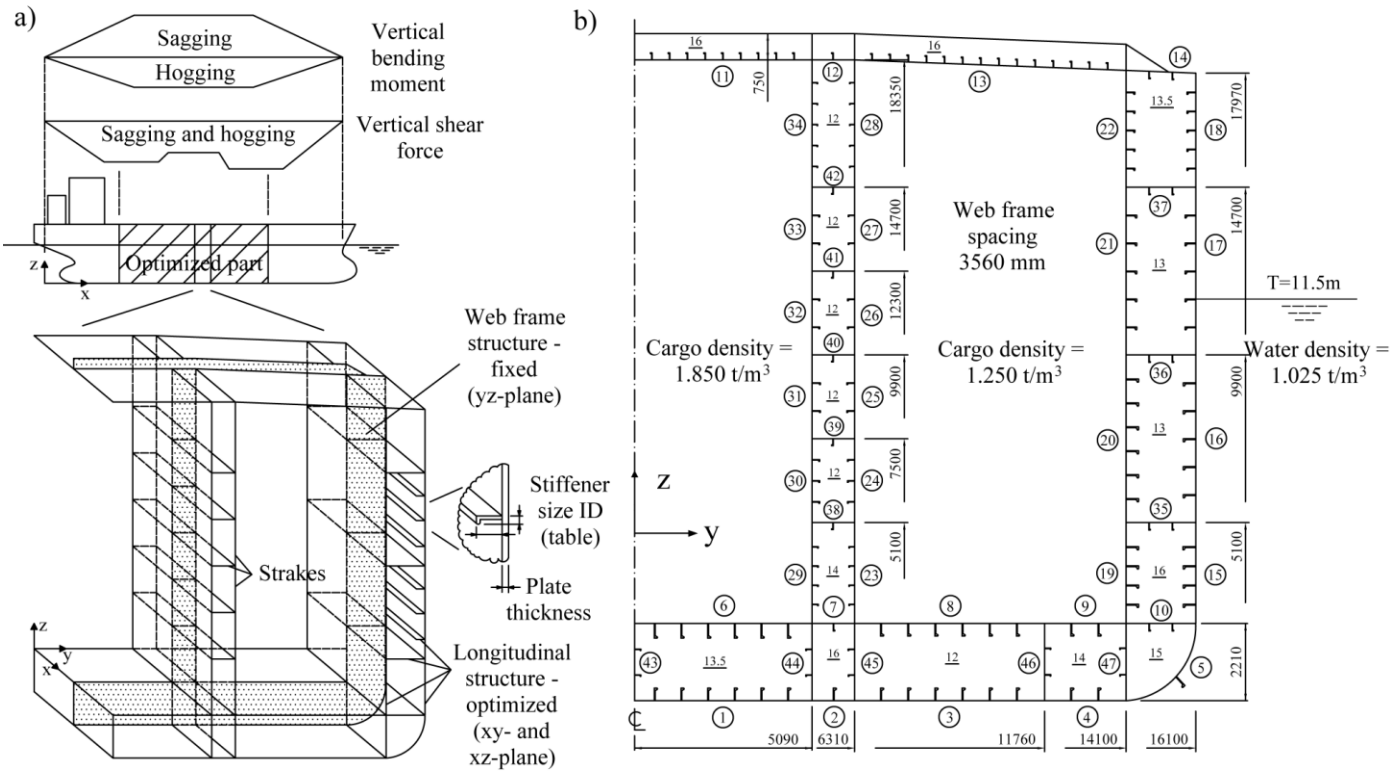

Figure 10. a) 3D view of the optimized structure and b) half of the main frame section with dimensions in mm, indicating strake numbers (in circles) and thickness of the transverse plates (underlined).

The optimization is performed for 47 strakes of the midship section. Each strake is defined by five parameters: plate thickness, stiffener size, number of stiffeners, stiffener type and strake's material type. Plate thickness and size of the stiffeners are considered as variables to be optimized and the other three parameters are considered fixed. Therefore, the problem includes 94 variables in total. The choice of decision variables will influence the final values of the objectives and a larger number of variables would result in more optimum solutions, however, the aim of the study is to compare the two versions of the optimization algorithm and this comparison remains qualitatively the same for a given problem.

Variables are considered discrete as it is common in industrial problems. Plate thicknesses are assumed to be available for every millimeter. Stiffeners are bulb flats (also known as Holland profiles) and their sizes are obtained from profiles offered by steel manufacturers [49]. Two

variable bounds (defined in Appendix B) with a larger and smaller share of feasible space are considered for comparing the performance of the algorithm.

Optimization of transverse members, i.e. the thickness of the web frames and their spacing, is not performed. Their fixed value is underlined in Figure 10(b). Number of the stiffeners in each strake is also shown in Figure 10(b).

Binary encoding is applied for this problem due to the convenience for discrete variables and also preliminary simulations revealed better performance of the algorithms with binary encoding.

The lateral pressures due to the loading from sea and cargo result in vertical bending moment and shear force subjected to the hull girder. The hydrostatic loading on each point on the structure is specified based on the height and densities of the cargo and sea as indicated in Figure 10.

Global loads on the ship hull girder are presented in Table 2. It should be noted that the highest magnitudes for force and moment distributions in Figure 10(a) are selected for each loading condition.

Table 2. Hull girder loads on the tanker [50]

Loading Condition	Sagging	Hogging
Vertical bending moment (kNm)	$-2.93 \cdot 10^6$	$2.41 \cdot 10^6$
Vertical shear force (kN)	$48 \cdot 10^3$	$48 \cdot 10^3$

Stresses due to the hull girder global loads are calculated applying the Coupled Beam (CB) method [51]. The method is validated with FEM results and it is commonly used for primary response of the hull in ship structures [52]–[54]. The idea is to divide the hull section into a number of longitudinal beams which can deform under bending and shear loadings. The beams are connected with shear and vertical springs to model slip or deformation of the beams. Local response of strakes under hydrostatic loading is calculated with the uniformly distributed simple beam model and is added to the stresses calculated due to the hull girder loading. The structure is made of steel with Young's modulus 206 GPa and Poisson ratio of 0.3. Yield strength of the cargo plating is equal to is 440 MPa and the rest of the structure has yield strength equal to 355 MPa.

Eight standard failure criteria are checked for each strake: plate yield and buckling, stiffener yield, lateral and torsional buckling, stiffener's web and flange buckling, (see [55], [56] for detailed definition). Also, a crossing over criterion [57] is used to ensure prevention of more gradual panel collapse, which requires that the global buckling strength be larger than plate or stiffener's buckling strength. Total number of constraints in the problem is 376. The constraint values are normalized using the normalization function proposed in [58]:

$$g_j(\boldsymbol{x}) = \frac{a_j(\boldsymbol{x}) - |b_j(\boldsymbol{x})|}{a_j(\boldsymbol{x}) + |b_j(\boldsymbol{x})|} \tag{22}$$

Where for the structural element j, $a_j(\boldsymbol{x})$ is the capacity that can be tolerated by the structure and $b_j(\boldsymbol{x})$ is due to the current loading situation. Therefore, constraint values can range between -1 and 1. Where negative values indicate constraint violating designs and zero represents the constraint boundary.

Two conflicting objectives are considered to be optimized in this problem: 1) Minimizing mass of the structure and 2) Maximizing adequacy of the deck. Mass is calculated by multiplying the steel density of 8 t/m^3 and cross-section of the longitudinal members by the whole length of the ship and adding 21.44 t as the mass of transverse members every 3.56 meters. The second objective which is formulated as the sum of the normalized constraint values of deck strakes aims to increase the safety of the deck. As in Equation (22), the constraint values are maximum when the stress reaches zero. Reducing the stresses in the deck will increase its endurance and fatigue life. The deck is proposed as the critical part of the structure due to its distance from the neutral axis of the structure and lower redundancy in comparison to other parts with double-plated construction; see Figure 10. The two objectives are in conflict since lighter designs have smaller cross-sectional dimensions which will result in higher stress and lower adequacy of the deck. Contrary, maximization of the adequacy objective demands designs that are deeper in the feasible space which means heavier structural members. The values of the objective functions are normalized in the algorithm procedures for unbiased evaluation between two objectives.

4.2.1 Optimization parameters

Optimization parameters used in the tanker problem are defined in Table 3. Population size N=100 and number of generations, Gen=500, are assigned based on the preliminary study for the available

computational resources (See Appendix A). 30 independent random initial populations are generated and used for all the algorithms for a fair comparison. The binary encoding is used due to the discrete identity of the variables and better performance of the algorithms in the preliminary studies for this problem and chromosomes of 400-bit-long are defined for the problem.

Table 3. Optimization parameters for tanker hull problem

N	Generations	Runs	p_c	p_m	N_1	N_2	N_R
100	500	30	0.9	0.003	35	35	10

Crossover probability is set to 0.9 as the common value in the literature [45], [46], [59]. Mutation probability is set to 0.003 which is slightly higher than the typical value (1/number of bits=0.0025) due to the complexity of the problem and in order to provide better diversity in the solutions [60]. Size of the repair sets, N_1 = 35, N_2 = 35 and N_R = 10 are assigned based on the preliminary study presented in Appendix A.

For the repair method, the constraint-variable connection should be defined. Since explicit function of constraints is not available for this problem, we assume that each constraint is function of a single variable. For each constraint type of a strake, either the plate thickness or the stiffener size is defined as the key variable. In reality, the stresses in the plates and stiffeners are functions of different parameters. Basically, second moment of inertia that is defined by all the members in the hull cross-section determines the global stresses which are superimposed to local stresses. However, local constraints are mainly sensitive to local plate and stiffener dimensions. The constraint-variable connection in Table 4 is a simplification based on assuming one variable with the most effect on each constraint.

Table 4. Eight constraints for each strake and the variables assumed to have the most influence defined for repairing purpose.

Constraint	Effective variable
Plate yielding	Thickness of the plate
Plate buckling	Thickness of the plate
Stiffener yielding	Size of the stiffener
Stiff. web buckling	Size of the stiffener
Stiff. flange buckling	Size of the stiffener
Stiff. lateral buckling	Size of the stiffener
Stiff. torsional buckling	Size of the stiffener
Crossing over	Size of the stiffener

4.2.2 Optimization results

In practical problems, the initial results during optimization can have higher importance than reaching the true Pareto frontier of the problem. In structural engineering, for instance, the computational resources are limited and time is an important decision factor as the design should meet specified deadlines. Therefore, performance comparison of the algorithms is essential for practical problems where search might be stopped prematurely. Figure 11 shows the median values of the hypervolume metric throughout the process. The proposed algorithm starts with non-zero hypervolume right after the start and maintains larger values in comparison to other algorithms throughout the generations. This indicates that the solutions made by the proposed algorithm are better in objective values (dominating) and/or have more diversity. The two constraint handling approaches from the literature slightly improve NSGA-II performance in the first 100 generations but they are still not as good as the proposed method results. Constraint handling from Ref. [42] converges to results of the original NSGA-II while the performance of the algorithm with constraint handling in Ref. [32] is even worse.

Non-zero hypervolume values for the repair algorithm are due to the success of the repair in making feasible solutions from the first attempt in all cases while the original NSGA-II takes 10-54 generations for finding the first feasible solution. In the early stage of optimization, in

generation 50 the hypervolume of the proposed and original algorithms are 0.59 and 0.38, respectively. The values reach 0.78 and 0.71 at the end of the 500th generation. Even though the results have not converged completely and further improvement in the results could be achieved with prolonging the process, the qualitative comparison of the algorithms would likely remain the same.

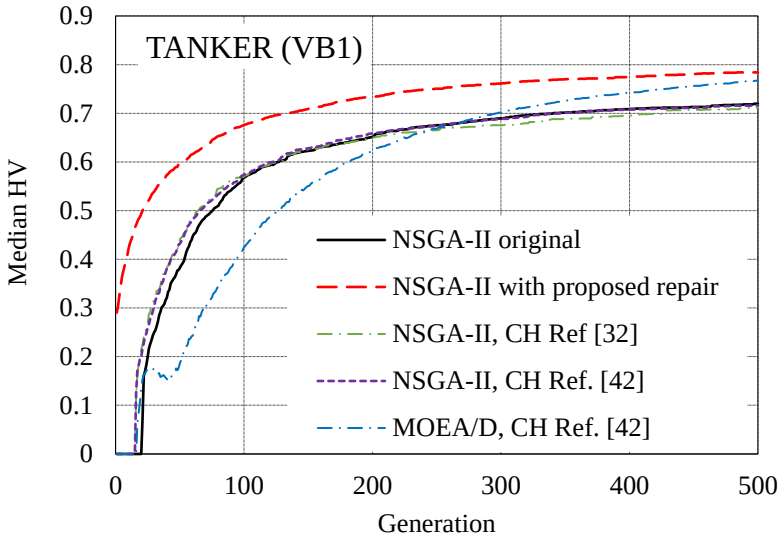

Figure 11. Median hypervolume performance metric for tanker hull problem through generations

The non-dominated fronts achieved by the algorithms in all the runs are shown in Figure 12. The proposed algorithm shows better convergence and wider spread of the solutions. It should be noted that finding the optima is not the main goal of this study but comparison of the algorithms and judgment on the effectiveness of the proposed repair approach is the major focus. MOEA/D obtained slightly better convergence in comparison with original NSGA-II but it could not outperform the proposed algorithm. The other examined constraint handling methods did not significantly improve the performance of NSGA-II. The part of the front which is associated with heavier designs and greater adequacy is easier to reach due to the compatibility of this part of the design space with constraints and all the algorithms merge in this part. Conversely, as moving toward lighter designs, the proposed algorithm shows better convergence and even spread in the parts which were not reached by the other algorithms.

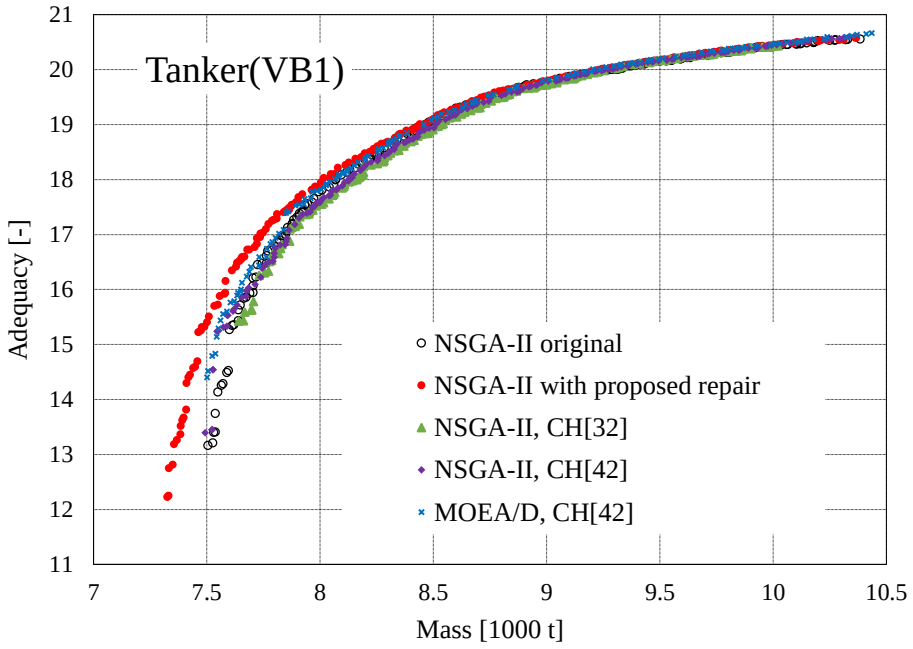

Figure 12. Final non-dominated fronts of the tanker problem, considering all independent runs.

Each optimization run for the tanker problem with a standard desktop computer takes $1.21011 \cdot 10^5$ s and $1.21045 \cdot 10^5$ s with the proposed and original NSGA-II, respectively. Considering the performance of the algorithms with difference in the processing time less than 0.03% underlines the value of the proposed method for the engineering problems.

The motivation of the proposed repair method is based on the idea of benefiting from part of the population which originally is discarded due to infeasibility. Evolutionary algorithms create a new set of solutions in every generation. In this engineering problem, initial populations in all runs are completely infeasible because of the large number of constraints. It takes significant number of generations and time for original NSGA-II to find the first feasible design; see Figure 11. For a deeper understanding, Figure 13 shows the number of infeasible solutions made in the children populations through all the generations in one run with the original NSGA-II algorithm. It takes multiple generations for the algorithm to start creating feasible solutions. After that, the number of infeasible designs reduces, but is still significant, rising from about 25% of the population at early generations to about 50% at the end of the search. This means that a great number of infeasible designs are created in every generation and then discarded as being inferior to feasible designs. It is interesting to note that considerable number of them dominate generation's best feasible front,

i.e. about 20% of the population. This justifies the reason for the effectiveness of the repair method in improving the performance of the original algorithm.

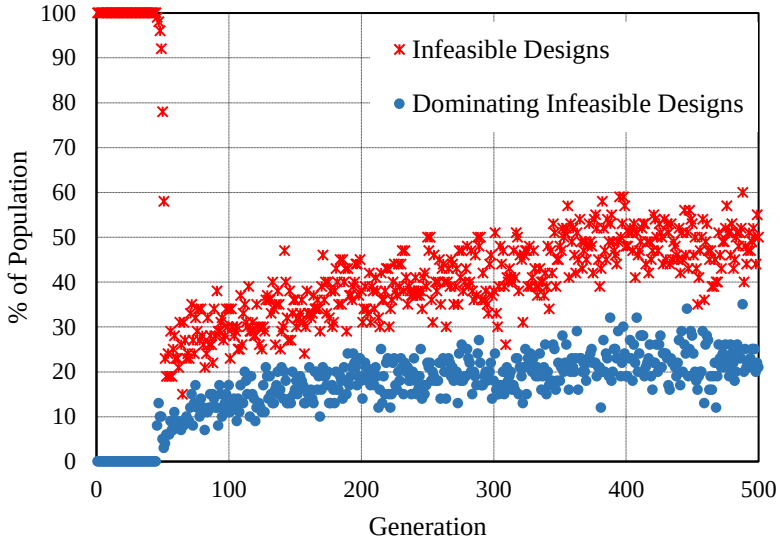

Figure 13. Infeasible designs with original NSGA-II in one run.

4.2.3 Effect of repairing

For better understanding the effect of the proposed method on the optimization process, the first two generations of the modified algorithm are analyzed here. All of the 30 initial populations were randomly created and they were all infeasible. The solutions in the initial populations violated between 7-35 constraints. In all runs, Repair1 algorithm was able to create feasible solutions in the first generation, see Figure 11. Repair2 algorithm was activated for the rest of the generations for repairing non-dominated infeasible solutions. The original NSGA-II took at least 10 generations for finding the first feasible solution and 54 generations at the latest.

Figure 14 shows the detailed performance of the proposed version of NSGA-II in the first two generations from a randomly chosen run. It clearly shows the beneficial effect of the repair method. Non-dominated solutions in Figure 14 are connected with a line. Figure 14(b) shows the children made by Repair1 (colored) and GA operators. Repair 1a which works on solutions closer to the feasible space resulted in heavier solutions compared to solutions made by Repair1b. Larger

stiffeners and thicker plates generally cause smaller violations of the constraints. Non-dominated front of solutions from Repair1b have a larger spread and more members since the repair candidates are chosen from infeasible solutions sorted based on non-domination rank and crowding distance. Repair1 algorithm was successful to make 22 feasible solutions in total out of the 70 candidates (~31%) while unsuccessful attempts were very close to feasibility with only 1-3 violated constraints and very competitive objective values (Figure 14(b),(e)). These solutions would be able to make their way out of the intermediate population pool and are highly potential parents for making valuable children which is an indirect help of the repair method to the algorithm. The remaining children made by GA operators were all infeasible even though Figure 14(e) shows that the average number of violated constraints has slightly decreased which indicates the relatively slow progress in the usual process of the original algorithm toward **Ω.** Hence, the repair method can effectively reduce the number of function evaluations required for reaching the feasible space.

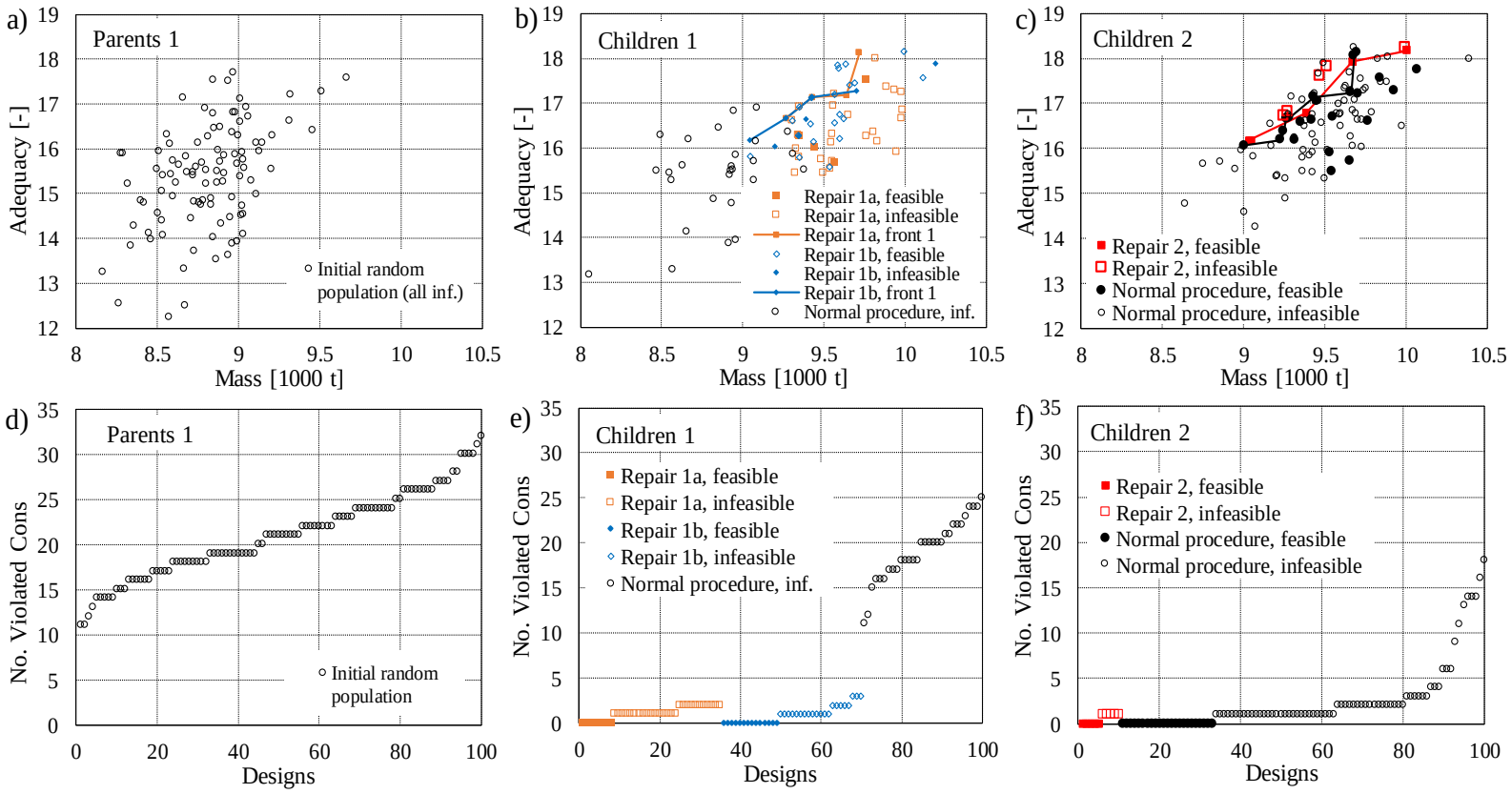

Figure 14. Objective space for (a) initial population, (b) children population 1 (repair 1) and (c) children population 2 (repair 2). Number of violated constraints for (d) initial population, (e) children population 1 and (f) children population 2.

Before starting the next generation, the algorithm checks if there are any feasible solutions in the final population of the generation. In that case, Repair2 would be activated. The repair candidates and donors are also based on the solutions of the current intermediate population for a bigger pool that contains more infeasible solutions. Repair2 does non-domination sorting on the infeasible solutions regardless of their constraint violation and only considering their objective values. Infeasible solutions that dominate solutions in the first non-dominated feasible front are considered as the repair candidates. Up to 10% of the population size is occupied by repaired solutions made by Repair2 based on the preliminary studies for the problem (see Appendix A). This will provide the proper balance between the repair effect and GA operators since after reaching the feasible space the randomness and heuristics of the GA operators are essential for the search.

Figure 14(c) presents the children of the second generation in the objective space. 10 solutions were repaired here which five of them successfully became feasible with three dominating the non-dominated front of the normal children. The other repaired solutions also have competitive objective values while only violating one constraint, Figure 14(f), which would be valuable for the next generation.

The repair method was able to create feasible solutions although the constraint-variable connection was simplified and inaccurate. Defining a more precise connection could result in more feasible solutions.

4.2.4 Influence of variable bounds

Variable bounds are an important factor that affects the complexity of optimization problems. Wider variable bounds enlarge the design space and can prolong the optimization process. Also, variable bounds are defined to avoid unrealistic or unattainable values in practical problems. For instance, size for structural members should be available from the manufacturers or the position of the members should not exceed the designed workspace. Additionally, the variable bounds should be defined in a way that they include feasible solutions comprising the optimum. However, it is often hard to define variable bounds beforehand since commonly we do not know where the optimum lies and which variable values would satisfy all the constraints. Inappropriate variable bound definition can increase the share of the infeasible space unintentionally. This can degrade

the algorithm performance since it would be more difficult for the algorithm to find the feasible space.

In this subchapter, variable bounds are changed for the tanker problem toward the lighter structures; smaller stiffener sizes and plate thicknesses (see Appendix B), to examine this situation on the performance of the algorithms. The length of the binary chromosome is kept unchanged which means that the number of the possible permutations or size of the design space is the same as before. The bounds are still in the available range provided by the manufactures and allowed in the ship classification societies [50].

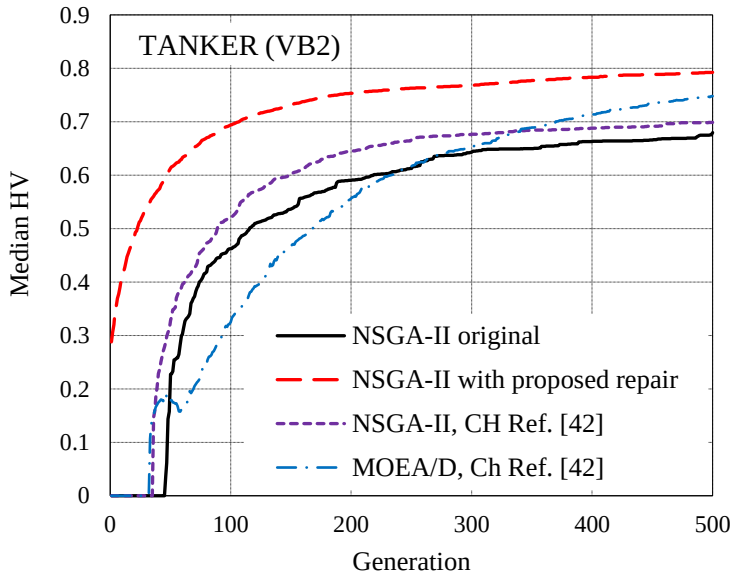

Figure 15. History of the median hypervolume metric for the tanker problem with the second variable bounds.

30 independent runs are performed with new random initial populations with the new variable bounds for each algorithm. All the optimization parameters remained unchanged. Figure 15 shows the median hypervolume metric of the populations throughout generations. The difference between the performance of the original and modified NSGA-II is even more severe compared to the first variable bounds. The constraint-handling method from [42] improves NSGA-II performance, however, it is not able to match the results of the proposed method. MOAE/D outperforms original NSGA-II in the middle of the search but cannot exceed the proposed method. The constraint handling method from Ref. [32] is not examined since it showed poor performance in the first variable bound case, see Figure 11 and Figure 12. While the proposed algorithm creates feasible

solutions from the second generation, it takes 34 to 72 generations for the original NSGA-II to find the first feasible solutions. The superiority of the proposed method is maintained through the process, with the final median HV of 0.68 and 0.79 for the original and proposed algorithms, respectively. The value of HVs in Figure 11 and Figure 15 should not be directly compared since the reference points of the HV calculations are not the same. The reference point and HV metric of an optimization problem are calculated based on the solutions obtained by corresponding algorithm runs. Nonetheless, the qualitative comparison of the algorithm performances is still valid.

Final non-dominated fronts from all the runs are presented in Figure 16. As can be seen, the proposed method has better spread and convergence. The proposed algorithm was able to obtain the part of the front populated with lighter designs, the part which is unattained by other algorithms. This is mainly because these designs are in the vicinity of the infeasible space and it is challenging for the algorithms to get close to them without violating the constraints. Variable values for the lightest designs obtained in first and second variable bounds are presented in Appendix B.

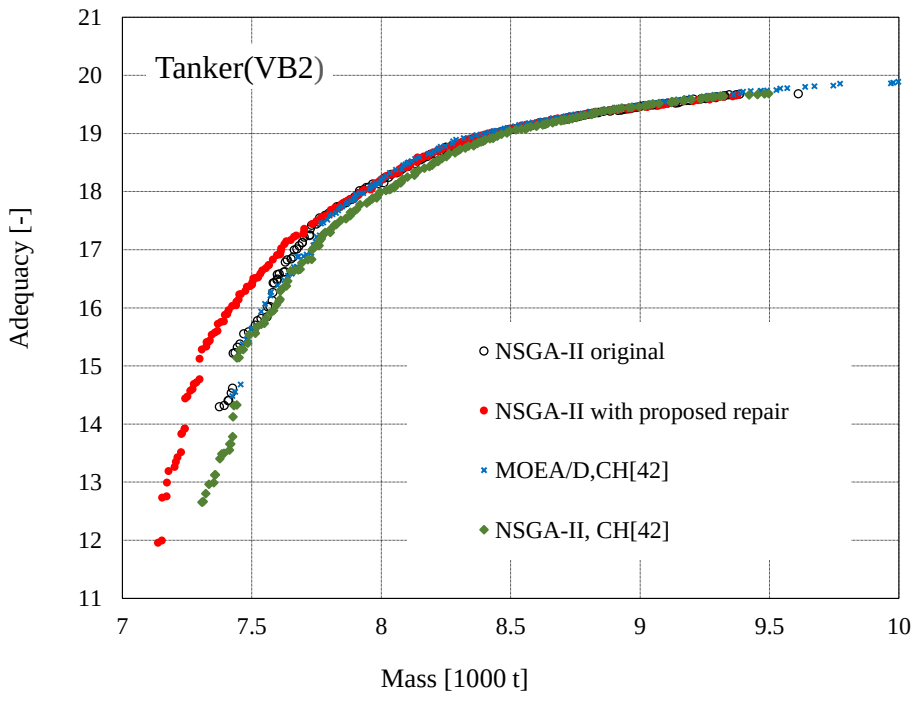

Figure 16. The final non-dominated fronts, considering all independent runs, for the second variable bounds.

In evolutionary algorithms, for a successful search, the algorithm has to work properly in exploring and exploiting the design space [61]. Exploration is searching different parts of the space while exploitation is searching in the neighborhood of the existing solutions. Exploration at the beginning of the search is important in order to find the potential regions. Consequently, the solutions would be very different at this phase. As the algorithm tends to exploitation latter, the difference in the solutions is smaller and the search is more local. If the ratio between these two phases is not balanced, it may result in premature convergence to local optima. For the tanker problem, prolonging the search process in finding the feasible region results in the loss of the diversity and lower hypervolume values, as seen from the performance of the original NSGA-II. The relatively long procedure in reaching the feasible space makes the solutions quite similar, meaning that crossover and mutation operators are not effective enough to provide the expected convergence and spread of the front.

5 Conclusion

5.1 Contributions and significance

In this thesis, a novel constraint handling method was developed for the non-dominance based genetic algorithm. The proposed method, called the repair method, was developed in order to improve the performance of the optimization algorithm by making use of valuable infeasible solutions made during the process. The goal was not only obtaining results with high quality but also reducing the number of attempts the algorithm has to make for finding feasible solutions. Infeasible solutions that dominate the first non-dominated feasible front are repaired. The repair is done based on the constraint-variable connection which can be identified after problem definition with certain confidence. Variable values are replaced for the corresponding violated constraints based on this connection. The source for this variable exchange would be non-dominated feasible solutions which are closest to the solution under repair. If there are no feasible donors in the population, infeasible solutions are used that do not violate the same constraints. The approach does not require pre-knowledge for making solutions feasible or preventing infeasible solutions to be made. The proposed approach relies on identifying variables with the most influence on constraints which is mainly based on the definition of the problem. This approach is applicable to problems where each constraint is function of a small group of variables. Problems OSY and tanker hull structure are studied in this thesis to present the effectiveness of this approach. Conclusions can be made as follows:

1. Efficiency: The proposed method was able to find feasible and high-quality solutions in fewer number of function evaluations. The proposed method showed better progress through the objective space especially at the beginning of the search which is very important where computational resources are limited and early acceptable solutions are desired. Additionally, the proposed method was successful in obtaining highly competitive results in comparison with the other algorithms after adding the difficulty of the constraint handling by shrinking the feasible design space.
2. Convergence: With equal function evaluations, the proposed method was able to reach equal or better front compared to the original NSGA-II algorithm and other constraint handling methods from the literature.
3. Diversity: With equal number of function evaluations, the proposed method was able to provide a better spread in the non-dominated front in comparison to the original NSGA-II and

other constraint handling methods used here. In the tanker problem, part of the front with lower weights which is a conflicting objective with the constraints is only found by the proposed method.

4. Complexity: The approach requires identifying variables which influence constraints the most. This was straightforward in the OSY problem with available function definitions. For the tanker problem in which the functions are computed numerically, simplification was made by defining a single variable responsible for a constraint. Despite the simplification, improvements in performance were achieved. Future study should assess the effect of the accuracy of the constraint-variable connection on the performance.
5. Computational cost: Extra non-domination sorting for the repair method is the main computational effort that the repair method adds to the algorithm. Increased computational time was negligible in the tanker engineering problem where significant computational time is spent on evaluation of each solution.

5.2 Future work

The case studies in this thesis were problems which have constraints that are functions of a small share of the variables. This is common in structural engineering problem for instance optimization of buildings, bridges or marine vessels. Applicability and performance of the proposed method for other problems should be further studied. The computational time due to the repair method can be reduced by optimizing the coding and function callings in MATLAB later for further improvement. Also, the influence of the accuracy of constraint-variable connection should be further analyzed. The proposed repair method is based on the constraint-variable relation which is associated with the definition of the problem. When this relation is perceived, the role of the algorithm is only ranking the solutions for determining the repair-worthy infeasible solutions and high-quality feasible donors. While different algorithms have different strategies for valuing their obtained solutions, the future work in the line of this research would be adaptively modifying the repair method for different optimization algorithms which are population-based or keep a memory of the solutions.

References

[1] K. Deb and D. Kalyanmoy, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[2] J. Nocedal, *Numerical optimization*. Springer, 2006.

[3] J. S. Arora, *Introduction to Optimum Design*, 4th ed. Academic Press, 2017.

[4] B. Steinberg, G. Yin, and F. S. Lobato, *Multi-Objective Optimization Problems Concepts and Self- Adaptive Parameters with Mathematical and Engineering Applications*. Springer, 2017.

[5] R. Michael, V. Torczon, and M. W. Trosset, "Direct search methods : then and now," vol. 124, pp. 191–207, 2000.

[6] R. Hooke and T. A. Jeeves, "Direct Searh Solutions of Numerical and Statistical Problems," pp. 212–229, 1960.

[7] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming for large-scale nonlinear optimization," vol. 124, pp. 123–137, 2000.

[8] S. Koziel, *Computational Optimization, Methods and Algorithms*. Springer, 2011.

[9] H. e Oliveira Junior, "The Many Aspects of Global Optimization," in *Evolutionary Global Optimization, Manifolds and Applications*, Cham: Springer International Publishing, 2016, pp. 3–12.

[10] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, 1995, vol. 4, pp. 1942–1948 vol.4.

[11] A. Gogna and A. Tayal, "Metaheuristics : review and application," *J. Exp. Theor. Artif. Intell.*, vol. 25, no. 4, pp. 503–526, 2013.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Am. Assoc. Adv. Sci.*, vol. 220, no. 4598, pp. 671–680, 1983.

[13] A.-T. Nguyen, S. Reiter, and P. Rigo, "A review on simulation-based optimization methods applied to building performance analysis," *Appl. Energy*, vol. 113, pp. 1043–1058, 2014.

[14] E. Mezura-montes and C. A. Coello, "Constraint-handling in nature-inspired numerical optimization : Past , present and future," *Swarm Evol. Comput.*, vol. 1, no. 4, pp. 173–194, 2011.

[15] A. Homaifar, C. X. Qi, and S. H. Lai, "Constrained Optimization Via Genetic Algorithms," *Simulation*, vol. 62, no. 4, pp. 242–253, 1994.

[16] J. A. Joines and C. R. Houck, "On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with GA ' s," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence,Orlando, FL*, 1994, vol. 2, pp. 579–584.

[17] S. Kazarlis and V. Petridis, "Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms," in *Parallel Problem Solving from Nature --- PPSN V*, 1998, pp. 211–220.

[18] B. Tessema, G. G. Yen, and S. Member, "A Self Adaptive Penalty Function Based Algorithm for Constrained Optimization," in *IEEE International Conference on Evolutionary Computation*, 2006, no. 1, pp. 246–253.

[19] Q. Long, "A constraint handling technique for constrained multi-objective genetic algorithm," *Swarm Evol. Comput.*, vol. 15, pp. 66–79, 2014.

[20] Z. SMichalewicz, *Genetic Algorithms+Data Structures = Evolution Programs*. Springer

Berlin Heidelberg, 1992.
[21] M. Schoenauer and Z. Michalewicz, “Evolutionary computation at the edge of feasibility,” in *Parallel Problem Solving from Nature*, 1996, pp. 245–254.
[22] S. Koziel and Z. Michalewicz, “A decoder-based evolutionary algorithm for constrained parameter optimization problems,” in *Parallel Problem Solving from Nature --- PPSN V*, 1998, pp. 231–240.
[23] C. A. Coello Coello, “Constraint-Handling Techniques Used with Evolutionary Algorithms,” in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, 2016, pp. 563–587.
[24] J. Paredis, “Co-evolutionary constraint satisfaction,” in *Parallel Problem Solving from Nature --- PPSN III*, 1994, pp. 46–55.
[25] E. Mezura-Montes and C. A. C. Coello, “Constrained Optimization via Multiobjective Evolutionary Algorithms,” in *Multiobjective Problem Solving from Nature: From Concepts to Applications*, J. Knowles, D. Corne, K. Deb, and D. R. Chair, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 53–75.
[26] K. Deb, “An efficient constraint handling method for genetic algorithms,” *Comput. Methods Appl. Mech. Eng.*, vol. 186, no. 2, pp. 311–338, 2000.
[27] S. Salcedo-sanz, “A survey of repair methods used as constraint handling techniques in evolutionary algorithms FEASIBLE,” *Comput. Sci. Rev.*, vol. 3, no. 3, pp. 175–192, 2009.
[28] K. Deb, “Multi-Objective Optimization Using Evolutionary Algorithms : An Introduction,” pp. 1–24, 2011.
[29] A. Klanac and J. Jelovica, “A concept of omni-optimization for ship structural design,” *Adv. Mar. Struct. - Proc. MARSTRUCT 2007, 1st Int. Conf. Mar. Struct.*, 2007.
[30] K. Deb, P. Jain, N. K. Gupta, and H. K. Maji, “Multiobjective Placement of Evolutionary Algorithms,” vol. 27, no. 3, pp. 480–492, 2004.
[31] M. G. Optimization, R. Kumar, P. P. Parida, and M. Gupta, “Topological Design ’ of Communication Networks using,” in *Proceedings of the 2002 Congress on Evolutionary Computation*, pp. 425–430.
[32] A. E. I. Brownlee and J. A. Wright, “Constrained , mixed-integer and multi-objective optimisation of building designs by NSGA-II with fitness approximation,” *Appl. Soft Comput. J.*, vol. 33, pp. 114–126, 2015.
[33] A. Hojjati, M. Monadi, A. Faridhosseini, and M. Mohammadi, “Application and comparison of NSGA-II and MOPSO in multi-objective optimization of water resources systems,” no. 2011, pp. 323–329, 2018.
[34] A. Nguyen, S. Reiter, and P. Rigo, “A review on simulation-based optimization methods applied to building performance analysis,” vol. 113, pp. 1043–1058, 2014.
[35] A. Konak, D. W. Coit, and A. E. Smith, “Multi-objective optimization using genetic algorithms: A tutorial,” *Reliab. Eng. Syst. Saf.*, vol. 91, no. 9, pp. 992–1007, 2006.
[36] K. Deb, A. Member, A. Pratap, S. Agarwal, and T. Meyarivan, “A Fast and Elitist Multiobjective Genetic Algorithm :,” vol. 6, no. 2, pp. 182–197, 2002.
[37] D. E. Goldberg and D. Kalyanmoy, *A Comparative Analysis of Selection Schemes Used in Genetic Algorithms*, vol. 1. Morgan Kaufmann Publishers, Inc.
[38] K. Deb and B. Ram, “Simulated Binary Crossover for Continuous Search Space,” vol. 9, pp. 115–148, 1995.
[39] Z. Qingfu and L. Hui, “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition,” *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, 2007.

[40] W. Peng, Q. Zhang, and H. Li, “Comparison between MOEA / D and NSGA-II on the Multi-Objective Travelling Salesman Problem,” pp. 309–324, 2009.
[41] H. Li and Q. Zhang, “Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II,” *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 284–302, 2009.
[42] T. Ray, R. Sarker, and K. Alam, “An Adaptive Constraint Handling Approach Embedded MOEA / D,” *2012 IEEE Congr. Evol. Comput.*, pp. 1–8, 2012.
[43] M. Asafuddoula, T. Ray, and R. Sarker, “A Decomposition-Based Evolutionary Algorithm for Many Objective Optimization,” *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 445–460, 2015.
[44] E. Zitzler and L. Thiele, “Multiobjective Evolutionary Algorithms : A Comparative Case Study and the Strength Pareto Approach,” vol. 3, no. 4, pp. 257–271, 1999.
[45] J. J. Durillo, A. J. Nebro, F. Luna, C. A. C. Coello, and E. Alba, “Convergence speed in multi-objective metaheuristics : Efficiency criteria and empirical study,” no. May, pp. 1344–1375, 2010.
[46] G. Zavala, A. J. Nebro, F. Luna, and C. A. Coello, “Structural design using multi-objective metaheuristics . Comparative study and application to a real-world problem,” pp. 545–566, 2016.
[47] A. Osyczka and H. Shi, “A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm,” vol. 99, no. Goldberg 1989, pp. 94–99, 1995.
[48] Z. Fan, “A Comparative Study of Constrained Multi-objective Evolutionary Algorithms on Constrained Multi-objective Optimization Problems,” *2017 IEEE Congr. Evol. Comput.*, pp. 209–216, 2017.
[49] “Bulb Flats.” .
[50] DNV GL, “RULES FOR CLASSIFICATION Ships,” *http://www.dnvgl.com*, 2019. .
[51] H. Naar, P. Varsta, and P. Kujala, “A theory of coupled beams for strength assessment of passenger ships,” vol. 17, no. 2004, pp. 590–611, 2005.
[52] J. Raikunen, E. Avi, H. Remes, J. Romanoff, I. Lillemäe-Avi, and A. Niemelä, “Optimisation of passenger ship structures in concept design stage,” *Ships Offshore Struct.*, vol. 14, no. sup1, pp. 320–334, 2019.
[53] P. Taylor *et al.*, “Ships and Offshore Structures Hull-superstructure interaction in optimised passenger ships,” no. February 2015, pp. 37–41, 2013.
[54] J. Jelovica and A. Klanac, “Optimization of crashworthy marine structures,” vol. 22, pp. 670–690, 2009.
[55] D. N. V. G. L. As, “RULES FOR CLASSIFICATION Ships Part 3 Hull Chapter 8 Buckling,” no. January, 2017.
[56] O. F. Hughes, *Ship Structural Design*. 1988.
[57] O. F. Ã. Hughes, B. Ghosh, and Y. Chen, “Improved prediction of simultaneous local and overall buckling of stiffened panels,” vol. 42, pp. 827–856, 2004.
[58] O. Hughes, F. Mistree, and V. Zanic, “A Practical Method for the Rational Design of Ship Structures,” *J. Sh. Res.*, vol. 24, pp. 101–113, 1980.
[59] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, “A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization : NSGA-II.”
[60] W. Banzhaf, F. D. Francone, and P. Nordin, “The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets,” in *Parallel Problem Solving from Nature --- PPSN IV*, 1996, pp. 300–309.
[61] M. Mernik, “Exploration and Exploitation in Evolutionary Algorithms : A Survey,” vol. 45,

no. 3, pp. 1–33, 2013.

Appendices

Appendix A: Influence of control parameters related to size of population sets

NSGA-II and repair algorithm:

Figure A.1 shows the influence of the population size on hypervolume metric for tanker optimization problem. Each curve is a median of 30 runs with the original NSGA-II. Population of 100 individuals achieved the highest performance metric value after equal number of 50,000 design evaluations. Therefore, population of 100 is chosen for all the runs and analysis in this study. It is also used for the OSY problem, as commonly adopted in literature for optimization test problems[36], [45], [46]

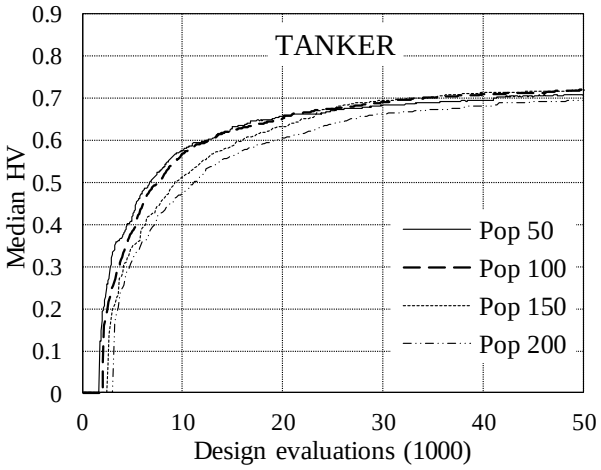

Figure A.1. Hypervolume metric for number of design evaluations for different population sizes in tanker problem, NSGA-II algorithm

Influence of the size of the repaired solutions in repair1 on the hypervolume metric in OSY and the tanker problems is shown in Figure A.2 and Figure A.3, respectively. The HV values are based on the median of 30 runs for each problem with the proposed version of NSGA-II. First generation is run for all cases since the algorithm successfully created at least one feasible solution. HV is improved at first by increasing the N_1 and N_2 and then reaches a plateau when they go above 20-40%. The stagnation can be explained by the fact that only the non-dominated solutions are contributed in the HV calculations yet larger share of repaired solutions are not necessarily adding to the non-dominated front.

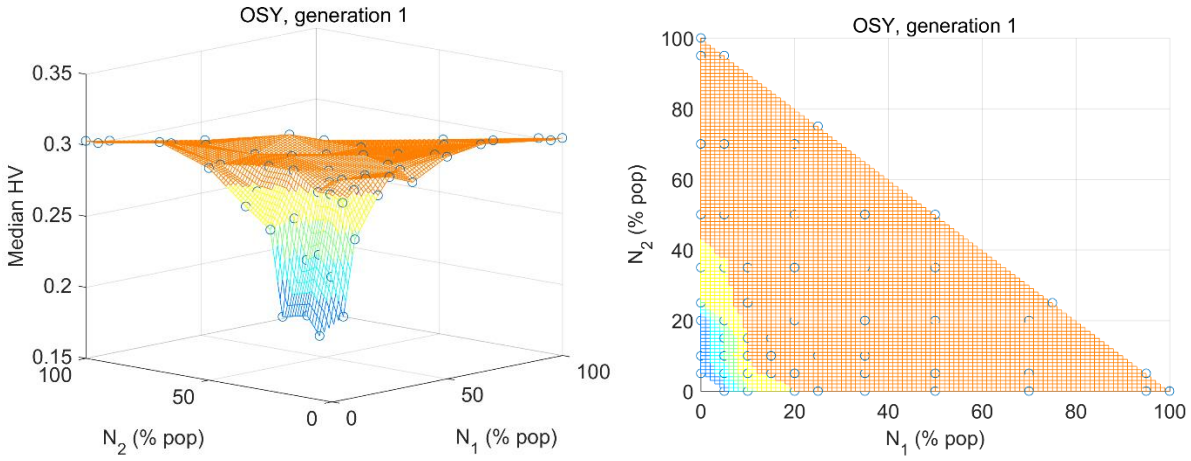

Figure A.2. Hypervolume metric for OSY problem as a function of N1 and N2.

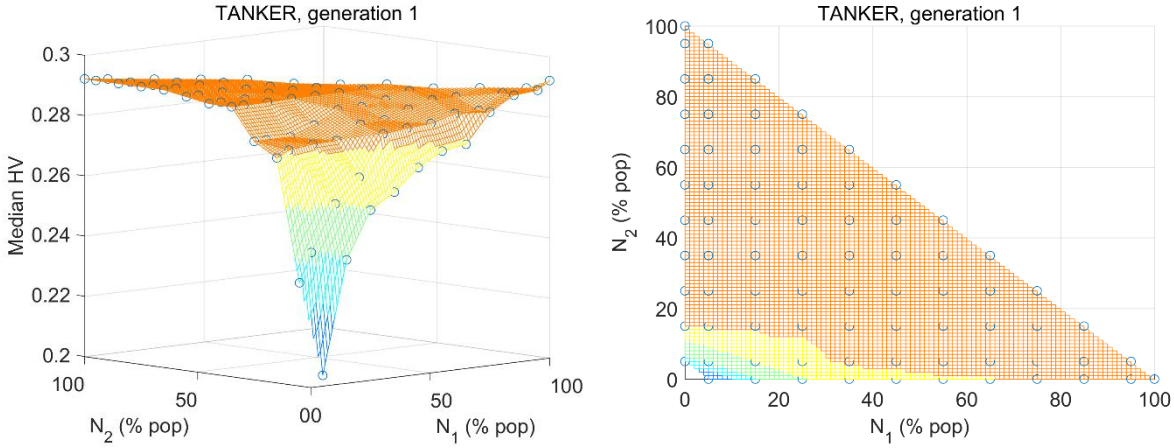

Figure A.3. Hypervolume metric for tanker problem as a function of N1 and N2.

To study the influence of N_R, three combinations of N_1 and N_2 with high HV in the first generation (Figure A.2 and Figure A.3) were selected: 0-95, 35-35 and 95-0. Box plots of the performance indicator are shown in Figure A.4. The results are obtained for 100 runs of 40 generations for the OSY and 30 runs of 100 generations for the tanker case to analyse the effect of Repair2 on the performance of the algorithm. It can be seen that N_R in the range 5-20% yields similar results. This can be explained due to the fact that as the algorithm proceeds, the solutions get closer and repair results are likely to become similar and hence not contribute to the HV. Also, not all repaired solutions necessarily end up in the non-dominated front. A control case was run where $N_1 = N_2 = 35$ and $N_R = 0$, which clearly performed worse than with non-zero N_R

values, indicating the beneficial effect of repairing throughout the optimization process. The best performance (resulting in the highest median) was with $N_1 = N_2 = 35$ and $N_R = 10$, for both problems.

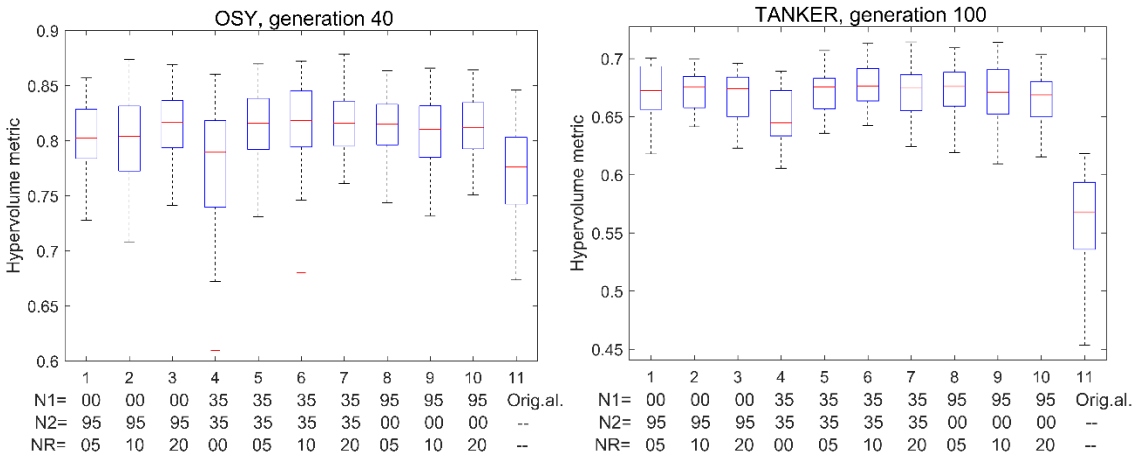

Figure A. 4. Box plots of hypervolume metric at intermediate stages of optimization

Constraint-handling approach from Ref [32] as described in 2.7.2 is used for algorithm performance comparison. For this method the control parameter *a* should be determined. Figure A.5 shows the influence of α on the performance for OSY. Larger *a* yields better performance in early stages of optimization (generation 40), while smaller *a* is better in the end. In the case of tanker, the same trend was observed. Thus, *a*=1 (i.e. 1% of the population) is selected.

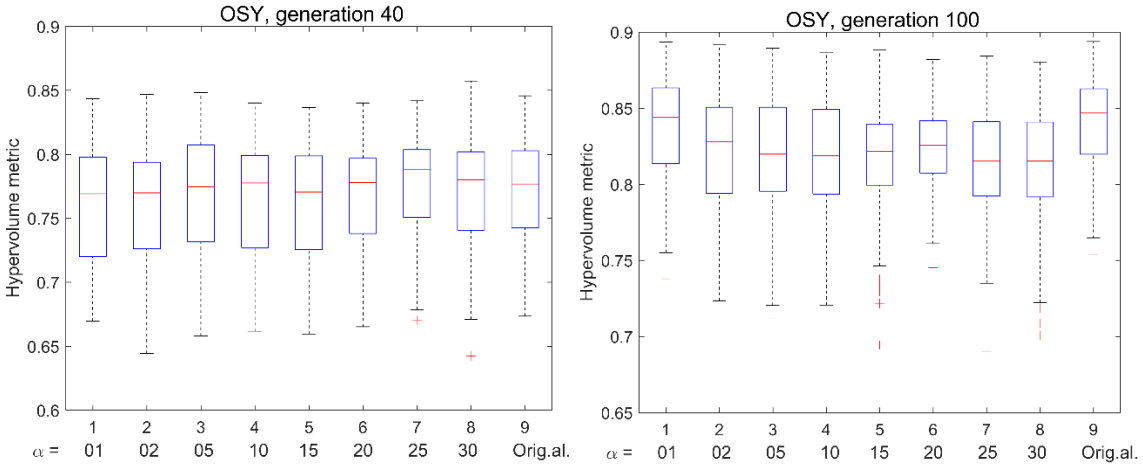

Figure A.5. Hypervolume metric for NSGA-II with constraint-handling from Ref. [31]

MOEA/D:

Population and neighborhood size parameters in MOEA/D algorithm are analyzed for OSY and tanker problems, presented in Figure A.6 and Figure A.7, respectively. Population 100 is selected for both problems and neighborhood of size 20 is selected based on the analysis with population 100 for both problems.

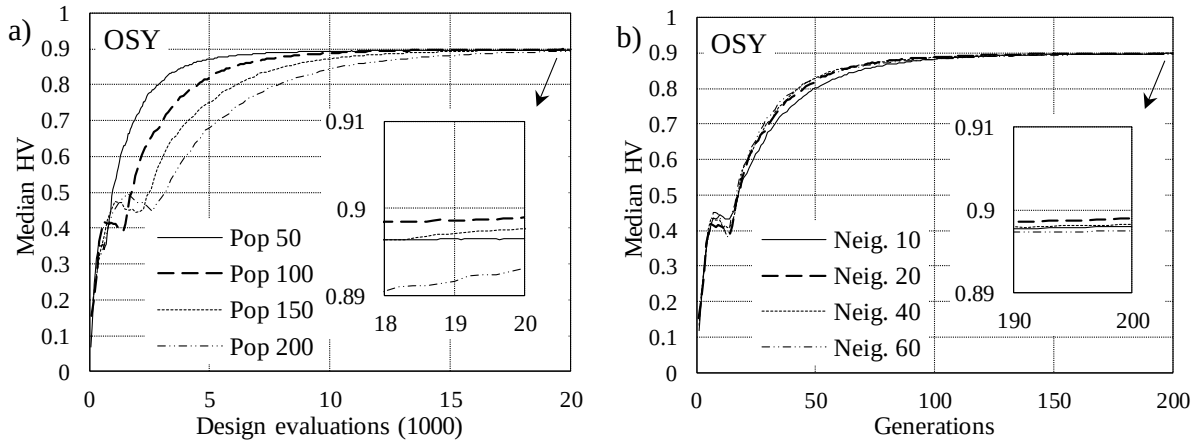

Figure A.6. Hypervolume metric for a) number of design evaluations for different population sizes b)in each generation with population 100 and different neighborhood sizes in OSY problem, MOEA/D algorithm

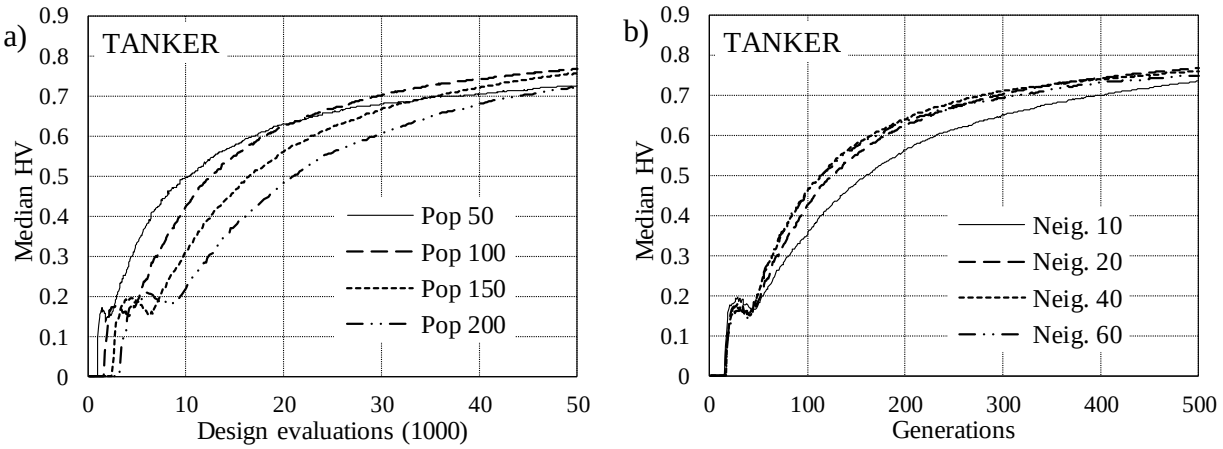

Figure A.7. Hypervolume metric for a) number of design evaluations for different population sizes b) in each generation with population 100 and different neighborhood sizes in tanker problem, MOEA/D algorithm

Appendix B: Tanker problem - variable bounds and optimized structural configurations

Two designs, the lightest for each of the variable bounds, are shown in Table B.2. They are selected since they reside in the most heavily constrained region of the design space. Their objective values are presented in Table B.1. Strake identification numbers are located in Figure 10.

Table B.1. Objective values of the two selected non-dominated solutions

	Mass [t]	Adequacy [-]
Design 1	7325.3	12.225
Design 2	7139.1	11.958

Table B.2. Variable bounds for the tanker problem and two selected non-dominated solutions. The first row for a strake shows plate thickness in millimeter. The second row shows the identification number of the bulb profile, as given in Table B.3.

Strake #	x_{min} (VB1)	x_{max} (VB1)	x_{min} (VB2)	x_{max} (VB2)	Design 1 (VB1)	Design 2 (VB2)
1	9	24	5	20	14	14
	26	41	12	27	30	26
2	9	24	5	20	13	12
	26	41	17	32	35	26
3	9	24	5	20	14	17
	29	44	12	27	32	26
4	9	24	5	20	14	16
	29	44	12	27	32	26
5	9	24	7	22	19	17
	29	44	22	37	40	31
6	9	24	7	22	18	19
	29	44	26	41	34	31
7	9	24	5	20	9	9
	29	44	1	16	39	4
8	9	24	7	22	16	16
	29	44	20	35	29	29
9	9	24	7	22	16	16
	29	44	20	35	31	29
10	9	24	5	20	9	9
	29	44	1	16	29	4
11	5	20	5	20	10	10
	13	44	13	44	29	27
12	5	20	5	20	20	18
	29	44	29	44	38	43
13	6	21	6	21	10	9
	15	30	15	30	18	23
14	11	26	11	26	22	15
	29	44	29	44	41	40
15	9	24	5	20	10	9
	13	44	1	32	20	27
16	9	24	5	20	9	9

	13	44	1	32	21	18
17	9	24	5	20	10	10
	13	44	1	32	18	12
18	9	24	5	20	9	9
	13	44	1	32	14	13
19	9	24	5	20	13	13
	13	44	4	35	28	28
20	9	24	5	20	13	13
	13	44	4	35	23	24
21	9	24	5	20	13	13
	13	44	4	35	21	27
22	9	24	5	20	10	9
	13	44	4	35	13	20
23	5	20	5	20	17	13
	6	37	1	32	26	26
24	5	20	5	20	13	13
	6	37	1	32	26	21
25	5	20	5	20	13	11
	6	37	1	32	26	32
26	5	20	5	20	13	13
	6	37	1	32	20	20
27	5	20	5	20	11	13
	6	37	1	32	18	21
28	5	20	5	20	10	9
	6	37	1	32	17	19
29	5	20	5	20	17	16
	6	37	6	37	30	32
30	5	20	5	20	14	15
	6	37	6	37	26	30
31	5	20	5	20	17	17
	6	37	6	37	26	26
32	5	20	5	20	15	15
	6	37	6	37	23	24
33	5	20	5	20	14	13
	6	37	1	32	26	31
34	5	20	5	20	11	13
	6	37	1	32	23	20
35	5	20	5	20	5	5
	10	41	1	32	13	5
36	5	20	5	20	5	5
	10	41	1	32	13	2
37	5	20	5	20	13	9
	10	41	1	32	10	6
38	5	20	5	20	5	5
	6	21	1	16	7	4
39	5	20	5	20	5	5
	6	21	1	16	13	1
40	5	20	5	20	5	5
	6	21	1	16	14	4
41	5	20	5	20	7	6
	6	21	1	16	11	7
42	5	20	5	20	8	9
	6	21	1	16	7	7

43	6	21	5	20	10	9
	13	28	3	18	13	5
44	6	21	5	20	10	9
	13	28	3	18	14	9
45	6	21	5	20	10	9
	13	28	3	18	15	5
46	6	21	5	20	10	9
	13	28	3	18	15	5
47	6	21	5	20	9	9
	13	28	3	18	17	7

Table B.3. Stiffeners' dimensions

Identification number	Thickness (mm)	Height (mm)
1	5	100
2	6	100
3	7	100
4	8	100
5	5	120
6	6	120
7	7	120
8	8	120
9	7	140
10	8	140
11	10	140
12	7	160
13	8	160
14	9	160
15	8	180
16	9	180
17	10	180
18	9	200
19	10	200
20	12	200
21	10	220
22	12	220
23	10	240
24	11	240
25	12	240
26	10	260
27	11	260
28	12	260
29	11	280
30	12	280
31	11	300

32	12	300
33	13	300
34	12	320
35	13	320
36	12	340
37	14	340
38	13	370
39	15	370
40	14	400
41	16	400
42	14	430
43	15	430
44	17	430