Efficient constraint multi-objective optimization with applications in ship design

Winter, R. de

Chapter 2

Preliminaries

This chapter introduces the most important acronyms, the problem notations, the basics of expensive black-box optimization, the benchmark functions used to test the performance of the proposed constraint multi-objective algorithms, and the relevant performance metrics for multi-objective optimization.

2.1 Acronyms

Two separate lists of the most important acronyms of this dissertation are made. One list consists of acronyms that are frequently used when discussing optimization algorithms. The second acronym list consists of acronyms used in the ship design domain.

2.1.1 Optimization Acronyms

COBYLA Constraint Optimization BY Linear Approximation algorithm
DoE Design of Experiments
EAF Empirical Attainment difference Functions
ECDF Empirical Cumulative Distribution Function
HV Hypervolume
IGD+ Inverted Generational Distance+ metric
LHS Latin Hypercube Sample
NSGA-II Non-dominated Sorting Genetic Algorithm II
PF Pareto Frontier

RBF Radial Basis Function
SAMO-COBRA Self-Adaptive Multi-Objective Constrained Optimization by using Radial Basis Function Approximations algorithm

2.1.2 Ship Design Acronyms

ρ Water Density
ACD Accelerated Concept Design
B Breadth
Boa Breadth Overall
Cb Block Coefficient
DWT Dead Weight Tonnage
FFD Free Form Deformation
KPI Key Performance Indicators
L Length
LBP Length Between Perpendiculars
LSW Light Ship Weight
MCR Maximum Continuous Rating
RANSE Reynolds Averaged Navier-Stokes Equation
T Draft
TEU Twenty-Foot Equivalent Unit
TSHD Trailing Suction Hopper Dredger
V Service Speed

2.2 Problem Notations

In this work, the following notations are used to define optimization problems. The focus of this work is mainly on optimization problems with two or more objectives, one or more constraints, and continuous decision variables. These problems can mathematically be formulated as follows:

$$\begin{aligned} \text{minimize: } & \mathbf{f} : \Omega \to \mathbb{R}^k \ , \ \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x}))^\top \\ \text{subject to: } & g_i(\mathbf{x}) \leq 0 \ \forall i \in \{1, \ldots, m\} \\ & \mathbf{x} \in \Omega \subset \mathbb{R}^d. \end{aligned} \tag{2.1}$$

In the following list, the elements and notations are explained in more detail.

- A solution with d design variables in the continuous domain is defined as $(x_1, \ldots, x_d) = \mathbf{x} \subset \mathbb{R}^d$, here d is called dimensionality.
- Multiple decision vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are denoted as a matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$.
- The design space that consists of all solutions is denoted by $\Omega \subset \mathbb{R}^d$. The Design space in this work is bounded by a lower bound and an upper bound $\mathbf{x_{lb}} \leq \Omega \leq \mathbf{x_{ub}}$. Here $\mathbf{x_{lb}} \in \mathbb{R}^d$, and $\mathbf{x_{ub}} \in \mathbb{R}^d$.
- An objective function that evaluates a solution and returns a continuous objective function value is formulated as $f : \Omega \to \mathbb{R}$.
- Unless otherwise specified all objectives are to be minimized: $\min_{\mathbf{x} \in \mathbb{R}^d}(f(\mathbf{x}))$.
- Objectives to be maximized are without loss of generality transformed into objectives to be minimized: $\min_{\mathbf{x} \in \mathbb{R}^d}(f(\mathbf{x})) = -1 \cdot \max_{\mathbf{x} \in \mathbb{R}^d}(f(\mathbf{x}))$.
- In multi-objective problems, k denotes the number of objectives, and the set of k objectives that evaluate a solution is combined into a vector $\mathbf{f} : \Omega \to \mathbb{R}^k$, $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x}))^\top$.
- A constraint function that evaluates a solution and returns the constraint violation score is formulated as $g : \Omega \to \mathbb{R}$.
- Unless otherwise specified, the constraint feasibility boundary is 0. Constraint values $g(\mathbf{x}) \leq 0$ are defined as feasible while constraint values $g(\mathbf{x}) > 0$ are infeasible. The output of the constraint function $g(\mathbf{x})$ is therefore a value that represents the constraint violation.
- Constraints with a constraint boundary $g(\mathbf{x}) \leq c$ can be rewritten as $g(\mathbf{x}) - c \leq 0$.
- Constraints where $g(\mathbf{x}) \geq 0$ is defined feasible are refactored to $g(\mathbf{x}) \geq 0 \ \rightarrow -1 \cdot g(\mathbf{x}) \leq 0$.
- An equality constraint $h(\mathbf{x}) = 0$ can be replaced with two inequality constraints. A good practice is to add a small margin ϵ around the constraint boundary so that in case there are tiny numerical instabilities this does not influence the feasibility of the solutions. $h(\mathbf{x}) = 0 \ \rightarrow \ g(\mathbf{x}) - 0.5\epsilon \leq 0$, and $g(\mathbf{x}) + 0.5\epsilon \geq 0$.
- In optimization problems m denotes the number of constraints. The set of m constraints can be combined into a vector $\mathbf{g} : \Omega \to \mathbb{R}^m$, $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_m(\mathbf{x}))^\top$.

2.2.1 Pareto Optimal Solutions

Now that the notations are clear, a definition can be given for optimal solutions. First, the definitions are given for a local and a global optimum and later the definition of Pareto optimal and Pareto dominance is given to define which solutions are preferred over other solutions for constraint multi-objective problems.

Definition 2.1 (Local Optimal Solution)**.** A local optimal solution is a solution that is the best in its local surroundings. This solution can be improved by inspecting other regions of the design space.

Definition 2.2 (Global Optimal Solution)**.** A global optimum is the best possible solution for the entire design space. Unlike a local solution, this solution is the absolute best solution and it can not be improved anymore.

Definition 2.3 (Feasible solution)**.** A solution $\mathbf{x} \in \Omega$ is *feasible* if and only if $g_i(\mathbf{x}) \leq 0 \; \forall i \in \{1, \ldots, m\}$.

Definition 2.4 (Dominance)**.** Solution $\mathbf{y} = \mathbf{f}(\mathbf{x})$ *dominates* solution $\mathbf{y}' = \mathbf{f}(\mathbf{x}')$ if and only if $\forall_{i \in \{1,\ldots,k\}} : y_i \leq y'_i$ and $\exists_{i \in \{1,\ldots,k\}} : y_i < y'_i$, in symbols $\mathbf{y} \prec \mathbf{y}'$.

Definition 2.5 (Incomparability)**.** Solution $\mathbf{y} = \mathbf{f}(\mathbf{x})$ is *incomparable* to solution $\mathbf{y}' = \mathbf{f}(\mathbf{x}')$ if and only if $\exists_{i \in \{1,\ldots,k\}} : y_i < y'_i$ and $\exists_{i \in \{1,\ldots,k\}} : y_i > y'_i$, in symbols $\mathbf{y} \| \mathbf{y}'$.

Definition 2.6 (Indifference)**.** Solution $\mathbf{y} = \mathbf{f}(\mathbf{x})$ is *indifferent* to solution $\mathbf{y}' = \mathbf{f}(\mathbf{x}')$ if and only if $\forall_{i \in \{1,\ldots,k\}} : y_i = y'_i$, in symbols $\mathbf{y} \sim \mathbf{y}'$. Note that indifference is equivalent to equality in the objective space. It is not guaranteed to also be equivalent in the design space: $\mathbf{f}(\mathbf{x}) \sim \mathbf{f}(\mathbf{x}') \implies \mathbf{x} = \mathbf{x}'$

Definition 2.7 (Pareto-optimal solution)**.** Solution $\mathbf{x} \in \Omega$ is *Pareto-optimal* if and only if there is no solution $\mathbf{x}' \in \Omega$ for which $f(\mathbf{x}') = \mathbf{y}' \prec \mathbf{y} = f(\mathbf{x})$.

Definition 2.8 (Feasible Pareto-optimal solution)**.** Solution $\mathbf{x} \in \Omega$ is *feasible Pareto-optimal* if and only if there is no solution $\mathbf{x}' \in \Omega$ for which $f(\mathbf{x}') = \mathbf{y}' \prec \mathbf{y} = f(\mathbf{x})$ where $g_i(\mathbf{x}) \leq 0$ and $g_i(\mathbf{x}') \leq 0 \; \forall i \in \{1, \ldots, m\}$.

Definition 2.9 (Pareto optimal set)**.** The set of solutions $\mathbf{X}$ that form the *Pareto optimal set* is the set of feasible incomparable and indifferent solutions that are not dominated by any other solutions $\mathbf{X} = \{\mathbf{x}' \in \mathbf{X} \mid \nexists \mathbf{x} \in \Omega : \mathbf{f}(\mathbf{x}) \prec \mathbf{f}(\mathbf{x}')\}$.

2.3 Expensive Black Box Optimization

When solving expensive black-box optimization problems, often Bayesian optimization is chosen as the method to find the optimal solutions. An argument for using Bayesian optimization algorithms is that these algorithms typically require a small evaluation budget to find (a set of) optimal solutions. This makes Bayesian optimization a good choice when the evaluation functions are computationally expensive [53, 90].

Bayesian optimization consists of 4 steps:

1. Bayesian optimization starts with a Design of Experiments (DoE). The solutions from the DoE are evaluated with the objective and constraint functions and put in an archive of evaluated solutions.

2. Surrogate models are fitted with all data from the archive.

3. With an infill criterion the promising solution(s) are selected based on the surrogate(s) predictions. The infill criterion is optimized with an optimization algorithm.

4. The new solutions are evaluated with the objective and constraint functions and added to the archive.

Finally, the algorithm terminates if a stopping criterion is met, otherwise the algorithm goes back to step 2. See Figure 2.1 for a graphical representation.

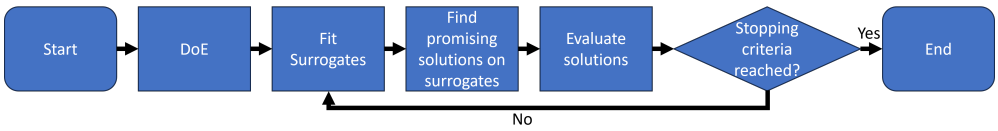


Figure 2.1: Flowchart of Bayesian optimization.

Choosing the optimal DoE strategy, surrogate models, and a local search method to find promising solutions on the surrogates is still a challenge. In the following subsections, a few DoE strategies, surrogate models, and local search methodologies are discussed that are used and referred to in this work.

2.3.1 Design of Experiments

In a design of experiments, the first set of solutions is generated and evaluated. The location in the domain where these first solutions are placed (also sometimes referred

to as initial sampling) is dependent on the DoE strategy. Several possible choices for a DoE exist, including uniform random sampling, full factorial design [17], Latin Hypercube Sampling [138], Halton sampling [73], and Sobol sampling [134]. Each of these methods has its own strengths, however, a large empirical comparison was presented by Bossek et. al. [23]. This empirical study concludes that spending as few evaluations on the DoE as possible is often beneficial because this leaves more room for evaluations proposed by the optimization algorithm [23]. This empirical finding also works best for most constraint multi-objective problems [148].

A recently proposed new sampling method is the Riesz s-energy-based sampling method [77]. The Riesz s-energy-based sampling method iteratively improves and proposes an arbitrary number of well-spaced points in the design space [21]. This method has been modified for constraint search spaces [19] so that it samples solutions only in the feasible area of the design space. This however is only practically applicable if the constraint functions are inexpensive to evaluate.

In industrial settings, solutions can often be evaluated in parallel. The number of solutions that can be evaluated in parallel are denoted with a p in this work. The number of solutions that can be evaluated in parallel is often dependent on the available computational resources and, if applicable, the number of commercial simulator licenses. The resources for parallel evaluations can be used to evaluate the initial DoE. The size of the DoE however now should be chosen based on the size of p. It is advised to choose a DoE size of $\lceil DoE_{min}/p \rceil \cdot p$, where p is the maximum possible number of simultaneous parallel evaluations and DoE_{min} the smallest DoE size required for training the first surrogate models. This way, no wall clock time is wasted, and the maximum amount of information from the objectives and constraints is gathered.

2.3.2 Surrogate Models

For the algorithm proposed in this research Radial Basis Functions (RBFs) and Kriging (also known as Gaussian process regression) are considered surrogate models. RBF and Kriging surrogates are fundamentally very similar, however, RBFs have many advantages: 1) RBFs require smaller sample sizes to fit a surrogate, 2) they are generally faster (also for larger input spaces), 3) have fewer assumptions on the underlying data, 4) deliver in many cases equal or better accuracy, 5) and with a newly developed uncertainty quantification method RBFs can now also be used in infill criteria that require this [12, 57]. Besides these fundamental arguments, an empirical comparison showed faster convergence and better results for algorithms with RBF surrogates compared to

the algorithm with Kriging surrogates [148]. For these reasons, the algorithm proposed in this work uses RBFs as surrogate models.

Radial Basis Function Interpolation

Radial Basis Functions (RBFs) are a type of mathematical function used for approximating the relationship between input and output variables [29]. The input variables are often the decision variables ($\mathbf{x}$), and the output variables are the objective ($\mathbf{f}$) or constraint ($\mathbf{g}$) value of the evaluated solutions. RBF interpolation approximates a function by fitting a linear weighted combination of RBFs [13]. The challenge is to find correct weights ($\boldsymbol{\theta}$) and a good RBF kernel $\varphi(\|\mathbf{x} - \mathbf{c}\|)$. An RBF is only dependent on the distance between the input point $\mathbf{x}$ to the center $\mathbf{c}$. The RBFs used in this work take each evaluated point as the centroid of the function, and the weighted linear combination of RBFs always produces a perfect fit through the training points. Besides the perfect fit on the training points, the linear combination of the RBFs can also give a reasonable approximation of the unknown area.

Any function which is only dependent on the distance from a specific point to another point belongs to the group of RBFs. The RBF kernels (φ) considered in this work are the *cubic* with $\varphi(r) = r^3$, *Gaussian* with $\varphi(r) = \exp\left(-(\epsilon \cdot r)^2\right)$, *multiquadric* with $\varphi(r) = \sqrt{1 + (\epsilon \cdot r)^2}$, *inverse quadratic* with $\varphi(r) = (1 + (\epsilon \cdot r)^2)^{-1}$, *inverse multiquadric* with $\varphi(r) = (\sqrt{1 + (\epsilon \cdot r)^2})^{-1}$, and *thin plate spline* with $\varphi(r) = r^2 \log r$. Note that the shape/width parameter ϵ for every individual RBF is kept constant as proposed by Urquhart et al. [159]. Moreover, all shape parameters are fixed to $\epsilon = 1$.

Finding suitable linear weighted combinations $\boldsymbol{\theta}$ of the RBFs can be done by inverting $\boldsymbol{\Phi} \in \mathbb{R}^{n \times n}$ where $\boldsymbol{\Phi}_{i,j} = \varphi(\|\mathbf{x}_i - \mathbf{x}_j\|)$:

$$\boldsymbol{\theta} = \boldsymbol{\Phi}^{-1} \cdot \mathbf{f} \tag{2.2}$$

Here $\mathbf{f}$ is a vector of length n with the function values of one of the objectives or constraints. Because $\boldsymbol{\Phi}$ is not always invertible, Micchelli introduced RBFs with a polynomial tail, better known as augmented RBFs [101]. In this work, augmented RBFs are used with a second-order polynomial tail. The polynomial tail is created by extending the original matrix $\boldsymbol{\Phi}$ with $\mathbf{P} = (1, x_{i1}, \ldots, x_{id}, x_{i,1}^2, \ldots, x_{id}^2)$, in its ith row, where x_{ij} is the j-th component of vector $\mathbf{x}_i$, for $i = 1, \ldots, n$ and $j = 1, \ldots, d$, $\mathbf{P}^\top$,

and zeros $\mathbf{0}_{(2d+1)\times(2d+1)}$, leading to $1+2d$ more weights $\boldsymbol{\mu}$ to learn.

$$\begin{bmatrix} \boldsymbol{\Phi} & \mathbf{P} \\ \mathbf{P}^\top & \mathbf{0}_{(2d+1)\times(2d+1)} \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\mu} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0}_{2d+1} \end{bmatrix} \tag{2.3}$$

Now that the weights $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ can be computed with Eq. 2.2, for each unseen input $\mathbf{x}'$ the function value (f') can be interpolated/predicted by using Eq. (2.4).

$$\begin{aligned} f' &= \boldsymbol{\Phi}' \cdot \Big[\boldsymbol{\theta}\boldsymbol{\mu}\Big] \\ f' &= \sum_{i=1}^{n} \theta_i \varphi(\|\mathbf{x}' - \mathbf{x}_i\|) + \mu_0 + \sum_{l=1}^{d} \mu_l \mathbf{x}'_l + \sum_{l=1}^{d} \mu_l \mathbf{x}'^2_l, \\ \mathbf{x} &\in \mathbb{R}^d \end{aligned} \tag{2.4}$$

Bagheri et al. also exploited similarities between RBF and Kriging surrogates to come up with an uncertainty quantification method for RBFs [12]. The formula for this uncertainty quantification method is given in Eq. (2.5).

$$\widehat{U}_{RBF} = \varphi(\|\mathbf{x}' - \mathbf{x}'\|) - \boldsymbol{\Phi}'^\top \boldsymbol{\Phi}^{-1} \boldsymbol{\Phi}' \tag{2.5}$$

where $\varphi(\|\mathbf{x}' - \mathbf{x}'\|) = \varphi(0)$ is a scalar value.

The uncertainty ($\widehat{U}_{RBF}$) of solutions far away from earlier evaluated solutions is higher compared to solutions close to earlier evaluated solutions. This uncertainty quantification method can therefore be used in infill criteria that require uncertainty quantification method to avoid getting stuck in a local optimal solution. However, as can be derived from Eq. (2.5), the uncertainty quantification method is only dependent on the input space and not on the scale of the objective and/or weights of the RBF models. It is therefore needed to use a consistent scale for both the input and the output space.

Scaling Techniques

For surrogate approximations, various scaling and transformation functions can be used to improve the surrogate fit. Four scaling and transformation techniques used in this work are described below.

Scale: The input space/decision variables are scaled into the range $[-1, 1]$ with $\mathbf{x} = 2 \cdot (\mathbf{x} - \mathbf{x_{lb}})/(\mathbf{x_{ub}} - \mathbf{x_{lb}}) - 1$. By scaling large values in the input space, computationally singular (ill-conditioned) coefficient matrices in Eq. (2.2) can

be prevented. In case the large values in the input space are kept, the linear equation solver will terminate with an error, or it will result in a large root mean square error [13]. Additionally, when fitting the RBFs, a change in one of the variables, is relatively the same change in all the other variables, making each variable in the basis equally important and equally sensitive.

STANDARDIZE: The relationship between the input space and the objective function values is modeled with the surrogates. To keep a consistent scale between the input and output scale, the output function values are standardized by using $y' = (y - \bar{y})/\sigma$. Here σ is the standard deviation of y, and $\bar{y}$ the mean of y. By using this standardization method, the uncertainty quantification from Eq. (2.5) can be used.

SCALE CONSTRAINT: The constraint evaluation function should return a continuous value, namely the amount by which the constraint is violated. Since it is possible to have multiple constraints, and each constraint is equally important, every constraint output is scaled with $c' = c/(\max(c) - \min(c))$, where $\max(c)$ is the maximum constraint violation encountered so far, and $\min(c)$ is the smallest constraint value seen so far. After scaling, the difference between $\min(c)$ and $\max(c)$ becomes 1 for all constraints, making every constraint equally important while 0 remains the feasibility boundary.

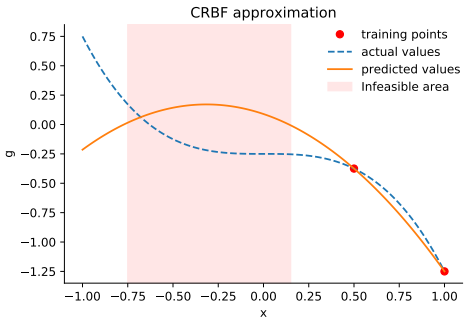
PLOG: In cases where there are very steep slopes, a logarithmic transformation of the objective and/or constraint scores can be beneficial for the predictive accuracy [125]. Therefore, the scores are transformed with the PLOG transformation function. The extension to a matrix argument $\mathbf{Y}$ is defined component-wise, i.e., each matrix element y_{ij} is subject to PLOG.

$$\text{PLOG}(y) = \begin{cases} +\ln(1+y), & \text{if} \quad y \geq 0 \\ -\ln(1-y), & \text{if} \quad y < 0 \end{cases} \tag{2.6}$$

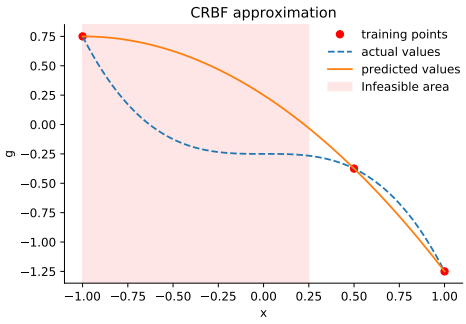
Radial Basis Functions Illustrative Examples

A visual representation (adapted from [142]) of how a Cubic RBF surrogate model is used to model a 1-dimension constraint function and how they become more accurate when more training points are added is presented in Figure 2.2. In the figures, the dashed blue line is the constraint function that the RBFs have to approximate, the red dots are the training points that have been evaluated and used to train the cubic RBF

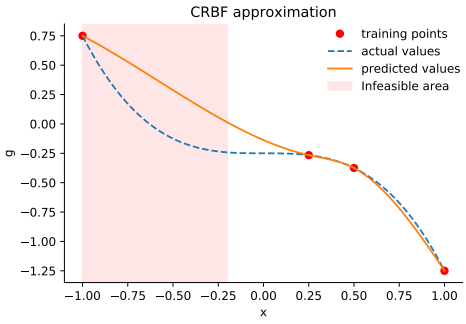
model, the solid orange line is the RBF prediction, the red shaded area is the predicted infeasible area where the RBF prediction is larger than 0 ($g(x) > 0$). Evident from the figures is that the surrogate's accuracy improves with the addition of more training points.



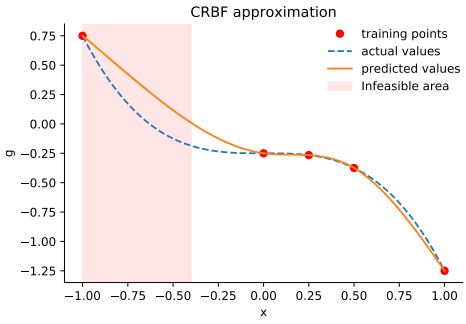
(a) RBF fit with 2 training points.



(b) RBF fit with 3 training points.



(c) RBF fit with 4 training points.



(d) RBF fit with 5 training points.

Figure 2.2: The dashed blue line is the constraint function that the Cubic RBFs have to mimic, the red dots are the training points that have been evaluated and used to train the cubic RBF model, the solid orange line is the RBF prediction given x, the red shaded area is the predicted infeasible area where the surrogate prediction is larger than 0 $g(x) > 0$.

Drawbacks of using Surrogates

There are a few scenarios for which surrogates are not ideal to use. If for example the constraint or objective is computationally cheaper than evaluating a solution on a surrogate, it is better to directly use the original constraint and or objective function. Another reason to not use surrogate-assisted algorithms is in the scenario where the

objective and or constraint function is highly multi-modal. If the global structure of the objective function is weak, and there are many local structures consisting of hills, finding an accurate fit of the to-be-modeled function with a surrogate is very difficult. This is difficult because many training points would be required to get an accurate fit of all the hills and alleys. It is for this reason that highly multi-modal problem landscapes are difficult to model with surrogates. Decision makers should therefore take the multi-modality, and expensiveness of the functions into account when selecting surrogate-assisted optimization algorithms.

2.3.3 Local Search Methodologies For Surrogate Exploration

Finding good solutions on surrogates is done by optimizing an acquisition function (also sometimes refereed to as infill criterion). The acquisition functions in this work are optimized with the so-called Constraint Optimization BY Linear Approximations algorithm (COBYLA) [120].

COBYLA linearly approximates the constraints and the acquisition function in a small trust region. In this trust region, COBYLA maximizes the acquisition function subject to the constraints by maximizing the following function:

$$\Psi(\mathbf{x}) = \hat{F}(\mathbf{x}) + \mu \left(-\max(c_i(\mathbf{x}) : i = 1, \ldots, m)\right)_+, \quad \mathbf{x} \in \mathbb{R}^d \tag{2.7}$$

Here, $\mathbf{x}$ is a solution in the input space, $\hat{F}$ is in our case the linear approximation of the acquisition function, c_i is the i-th linear approximation from the m constraint functions, the subscript $+$ means that the expression in the brackets becomes 0 if none of the constraints are violated, and μ is a self-adaptive penalty parameter that makes sure that the approximation of a new solution $\Psi(\mathbf{x}^*)$ with a smaller constraint violation and better acquisition function score is preferred over the starting solution approximation $\Psi(\mathbf{x}^0)$. After the best solution in the trust region on the linear approximations is found, it is evaluated on the constraint surrogate and acquisition function. When the linear approximation of COBYLA in the trust region underestimates the acquisition function, the trust region increases in size, while if it overestimates, the trust region becomes smaller. This way, when nearing the solutions that have the optimal acquisition score, the trust region becomes smaller and smaller until it falls below an $\varepsilon > 0$ value and COBYLA terminates.

Since COBYLA is a local optimizer, it can get stuck in a local optimum [169]. To overcome this problem, multiple instances of COBYLA are run in parallel. Searching for optimal solutions in multiple locations simultaneously is also a well-known strategy

in the Island model for parallel optimization [70], in parallel simulated annealing [91], and in ant colony optimization [51]. After all COBYLA instances have converged, all solutions are found. The solution with the best acquisition score is selected for evaluation on the expensive objective and constraint functions.

2.4 Benchmark Test Functions

For assessing the performance of various constraint multi-objective optimization algorithms a set of benchmark test functions are collected. An overview of the test functions is given in Table 2.1. In this table, the reference point (the worst possible value for each objective), the approximated Nadir point (the worst possible value for each objective among all solutions on the Pareto frontier), the number of objectives k, the number of dimensions d, the number of constraints m, and the feasibility ratio (P%) after one million random samples are given.

Table 2.1: Test function name, reference point used during optimization, Nadir point approximation based on all experiments in this work, number of objectives k, number of decision variables d, number of constraints m, percentage of feasible solutions $P(\%)$ after one million random samples.

Function	Reference point	Nadir point	k	d	m	$P(\%)$
BNH	(140, 50)	(136.00, 50.00)	2	2	2	96.92
CEXP	(1, 9)	(1.00, 9.00)	2	2	2	57.14
SRN	(301, 72)	(222.99, 2.62)	2	2	2	16.18
TNK	(2, 2)	(1.04, 1.04)	2	2	2	5.05
CTP1	(1, 2)	(1.00, 1.00)	2	2	2	92.67
C3DTLZ4	(3, 3)	(2.00, 2.00)	2	6	2	22.22
OSY	(0, 386)	(-41.81, 76.00)	2	6	6	2.78
TBTD	(0.1, 50000)	(0.1, 10000)	2	3	2	19.46
NBP	(11150, 12500)	(12500, 114.09)	2	2	5	41.34
DBD	(5,50)	(2.79, 16.86)	2	4	5	28.55
SRD	(7000, 1700)	(5879.98, 1696.46)	2	7	11	96.92
WB	(350, 0.1)	(35.31, 0.0145)	2	4	5	35.28
BICOP1	(9, 9)	(1.00, 1.00)	2	10	1	100
BICOP2	(70, 70)	(1.10, 1.11)	2	10	2	10.55
MW1	(1,7)	(1.00, 1.00)	2	8	1	0.007
MW2	(1,7)	(1.00, 1.00)	2	6	1	0.55
MW3	(1,7)	(1.00, 1.00)	2	6	2	1.32
MW11	(30,30)	(2.06, 2.04)	2	6	4	1.38
TRIPCOP	(34, -4, 90)	(7.67, -11.77, 25.91)	3	2	3	15.85
SPD	(16, 19000, -260000)	(11.16, 12435.27, -259148.04)	3	6	9	3.27
CSI	(42, 4.5, 13)	(42.77, 4.00, 12.52)	3	7	10	18.17
WP	(83000, 1350, 2.85, 15989825, 25000)	(74573, 1350, 2.85, 7874925, 25000)	5	3	7	92.06

The following functions are artificially created test functions: BNH [41], CEXP [46], SRN [50], TNK [50], CTP1 [46], C3DTLZ4 [141], OSY [41, 50], NBP [62], BICOP1 [44], BICOP2 [44], TRICOP [44], MW1 [97], MW2 [97], MW3 [97], MW11 [97].

The following functions are coming from real-world problems: Two-Bar Truss Design (TBTD) [66], Disk Brake Design (DBD) [66], Ship Parametric Design (SPD) [117],

Car-Side Impact (CSI) [83], speed Reducer Design (SRD) [105], Welded Beam (WB) [66], Water resource management Problem (WP) [83].

The test functions are selected because they are diverse, well known, and some mimic industrial problems. The Pareto frontiers of the functions vary between 2 and 5 objectives and the shapes can be classified as concave, convex, connected, disconnected, or even mixes of these characteristics [6]. The constraints of the selected test problems are also diverse since for some problems they are very strict, while for other problems (almost) the entire search space is feasible. Next to the feasibility of the problems, on some Pareto frontiers, the constraints are active, while on other problems they are not, or partially active. Next to the artificially created test functions, a set of real-world-inspired problems is selected to assess how well the optimization algorithms operate in situations resembling industrial optimization scenarios.

2.5 Algorithm Performance Metrics

The performance of algorithms can be determined with performance metrics. These metrics help to determine which algorithm perform well on benchmark problems and visualizing the evolution of the performance metric gives insight in how fast the algorithms converge. Since the focus of this work is multi-objective optimization algorithms the most used multi-objective metrics are presented and visualized.

2.5.1 Multi-Objective Performance Metrics

The two commonly used multi-objective performance metrics used in this work are the HyperVolume (HV), and the Inverted Generational Distance+ (IGD+) metric.

Hypervolume

The *hypervolume* (also known as the Lebesgue measure) translates the multi-objective problem into a unary performance score that represents the volume of the region in the objective space that is dominated by a given set of solutions [18, 175]. It is the most widely used performance metric in multi-objective optimization [128] and measures and captures the overall convergence and diversity of the set of solutions forming the Pareto front. The HV is calculated by determining the volume of the region in the objective space between the solutions on the obtained Pareto front and a pre-defined reference point (also sometimes referred to as the anti-optimal point [158]). For a single solution on two objective problem, this is easy to compute as it is the surface

between the solution and the reference point. If more solutions are on the Pareto front, the overlapping region is only counted once, this is done by calculating the union of the overlapping regions. The formal definition of the hypervolume is given in Definition 2.10.

Definition 2.10 (Hypervolume Indicator)**.**

$$HV(\mathbf{Y}, \mathbf{f}^{ref}) = \Lambda_k(\cup_{\mathbf{y_i} \in \mathbf{Y}}[\mathbf{y}, \mathbf{f}^{ref}]) \tag{2.8}$$

here Λ_k denotes the Lebesgue measure on $\mathbb{R}^k$, with k being the number of objective functions, $\mathbf{y}_i$ the i-th Pareto optimal solution, $\mathbf{Y}$ all Pareto optimal solutions, and $\mathbf{f}^{ref}$ the reference point in k dimensions.

The HV is a useful measure for comparing the performance of different optimization algorithms, as well as for comparing different solution sets with each other. Solution sets with higher HV are considered better compared to solutions with lower HV. The HV of three solutions is visualized in Figure 2.3, where the triangles represent the evaluated non-dominated solutions, and the reference point is indicated by a star. The surface of the shaded area is the HV of this particular Pareto front.

Inverted Generational Distance +

Another performance metric often used in multi-objective optimization is the *inverted generational distance+* metric (IGD+) [82]. It is used as a measure to check how close the known Pareto frontier is to the obtained dominated area. The IGD+ metric evaluates diversity and convergence as follows:

$$IGD^+(A, S) = \frac{1}{\mid S \mid} \left(\sum_{i=1}^{|S|} (d_i^+)^2 \right)^{\frac{1}{2}} \tag{2.9}$$

Here S is the known Pareto front, A is the dominated area by a Pareto front Y obtained by an algorithm, and d_i^+ is the smallest Euclidean distance from a solution on the known Pareto front s_i to the dominated area of A. This way, if the obtained dominated area of the solutions found by the algorithm is far away from the known Pareto front, the IGD+ value increases. A smaller IGD+ value is therefore preferred over a larger IGD+ value. The IGD+ metric and the distances of 8 known solutions are visualized in Figure 2.4.

The IGD+ metric can only be used on test instances where the Pareto front is

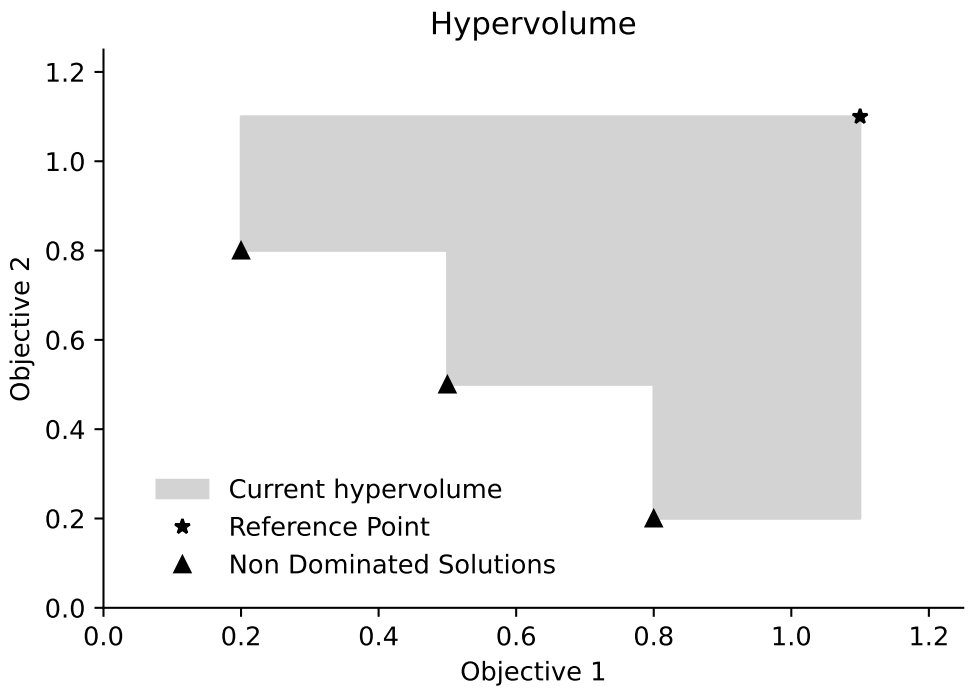


Figure 2.3: Visual representation of hypervolume. Total hypervolume between the 3 solutions and the reference point is $0.3 \times 0.8 + 0.3 \times 0.5 + 0.3 \times 0.3 = 0.48$.

known. In this work, the Pareto front used for the IGD+ metric is approximated by combining all obtained feasible Pareto efficient points of all experiments, then normalizing the objective scores, and finally selecting a well-spread set of solutions. The true Pareto front of computationally expensive engineering problems are often impossible to determine, therefore this revised metric is only used on test problems.

2.5.2 Optimization Result Visualizations

One of the key goals of Bayesian optimization is finding the global optimal (set of) solutions in as few evaluations as possible. Visualizing the set of optimal solutions in multi-objective optimization is usually done on a 2-dimensional or 3-dimensional Pareto frontier. When more than 3 objectives are to be visualized, or the objectives should be visualized together with the constraints and decision variables, then a parallel coordinate plot can be used. To visually compare the obtained Pareto frontiers of more than one algorithm run Empirical Attainment Function (EAF) difference plots

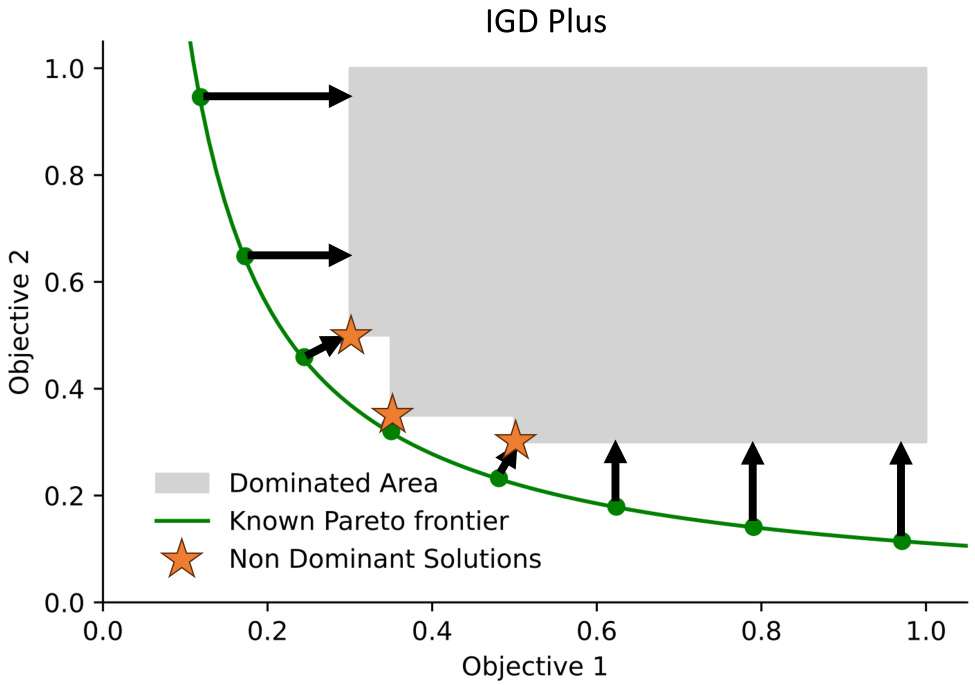


Figure 2.4: IGD+ score visualized on a two objective problem. IGD+ is the average length of the arrows from the well-spread known solutions to the closest point of the dominated area.

can be used for two-dimensional Pareto frontiers. The downside of visualizing only the final result is that information about how fast the results are found is missing. Convergence plots fortunately can give answer to this issue. On convergence plots, a performance score is plotted after every function evaluation from a run of an optimization algorithm on a specific test function. However, when comparing the convergence of algorithms on a set of different function evaluations Empirical Cumulative Distribution Function (ECDF) plots can be used. In the following subsections, all the visualization techniques are explained in more detail.

Pareto Frontier Plot

The Pareto frontier plot is a visual representation of the objective scores from Pareto-optimal solutions. The objective score of each Pareto efficient solution is plotted on the Pareto frontier. The Pareto frontier this way shows the trade-off between the objectives. Improvements of a solution in one objective on the Pareto frontier can only be made by sacrificing any of the other objectives. An example Pareto frontier

on the Two-bar Truss Design (TBTD) problem [66] obtained with the SAMO-COBRA algorithm (the SAMO-COBRA algorithm will be introduced in Chapter 5) is given in Figure 2.5.

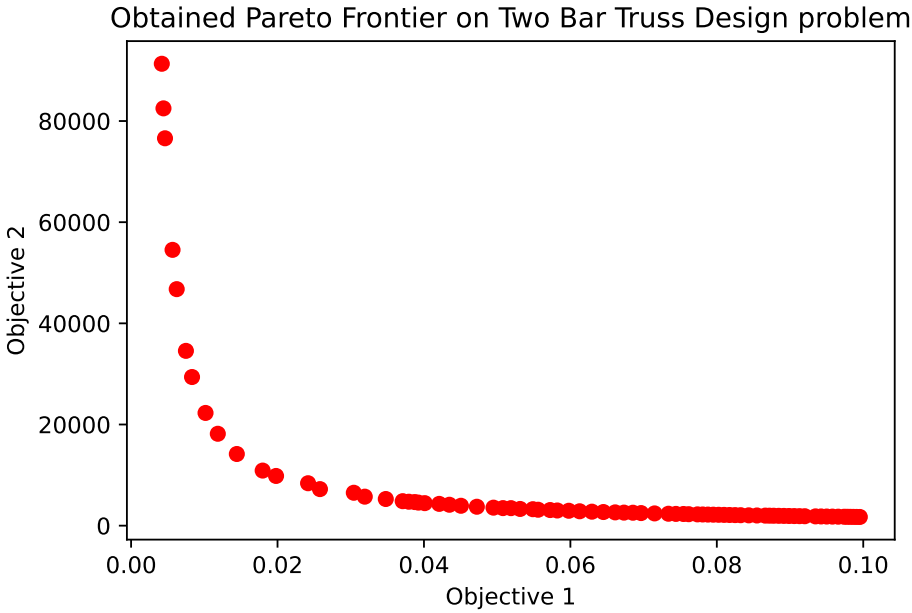


Figure 2.5: Pareto Frontier of TBTD Problem obtained with SAMO-COBRA algorithm after 120 function evaluations.

By plotting the Pareto frontiers of different algorithms, the objective space can be inspected. Inspection of the objective space shows in which regions which algorithm performs better. An illustrative example of the SAMO-COBRA algorithm and the SA-NSGA-II algorithm (The SA-NSGA-II algorithm will be introduced in Chapter 5) of the TBTD problem is presented in Figure 2.6.

From this figure it can be concluded that SAMO-COBRA has obtained slightly better and many more good solutions for objective 2, while the SA-NSGA-II algorithm has achieved better results in objective 1 since it found very small values for this objective.

Parallel Coordinate Plots

Two dimensions can be plotted on a regular Pareto frontier with two axes. In higher dimensions, parallel coordinate plots can be used [78]. In a parallel coordinate plot,

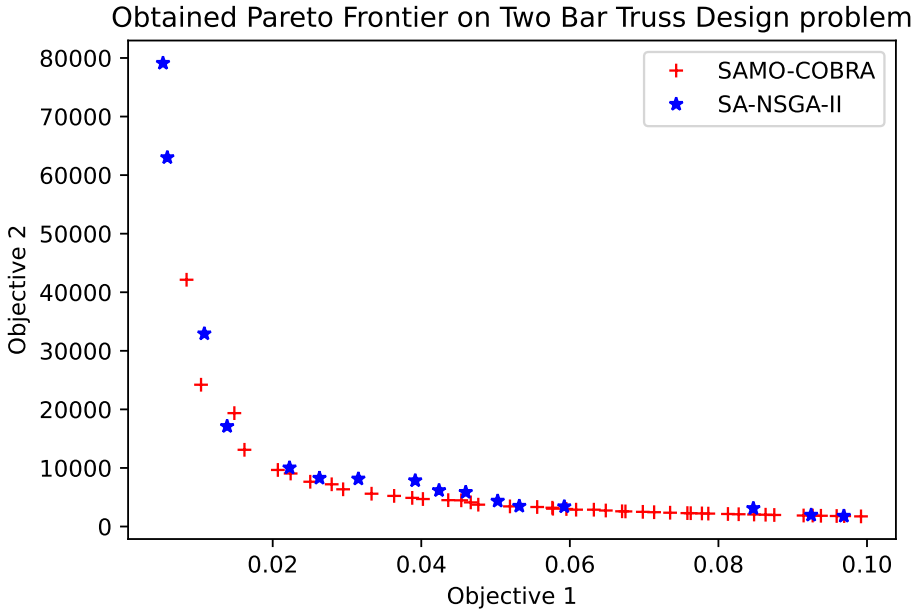


Figure 2.6: Pareto Frontiers on TBTD problem obtained with the SAMO-COBRA and SA-NSGA-II algorithm after 120 function evaluations.

each vertical axis represents a parameter, objective, and/or constraint while the horizontal axis has no meaning except for that it is the minimum and maximum value that has been found for the objective. Each solution in the parallel coordinate plot is represented by a line that connects the vertical axes. An example of a parallel coordinate plot of the Water resource management Problem (WP) [83] is presented in Figure 2.7.

The first thing that can be concluded from this parallel coordinate plot is that the solutions on each axis are very well spread since there are no large gaps between the solutions. The second thing that can be concluded is that there is a very clear inverse correlation between objective 3 and objective 4.

Empirical Attainment Difference Function

Comparing the obtained Pareto frontiers of different algorithms of one run per algorithm can still be done with a regular Pareto frontier plot as in Figure 2.6. However, when the algorithms have been used to optimize the benchmark problem more than once and the results are each time a bit different due to the stochastic nature of the

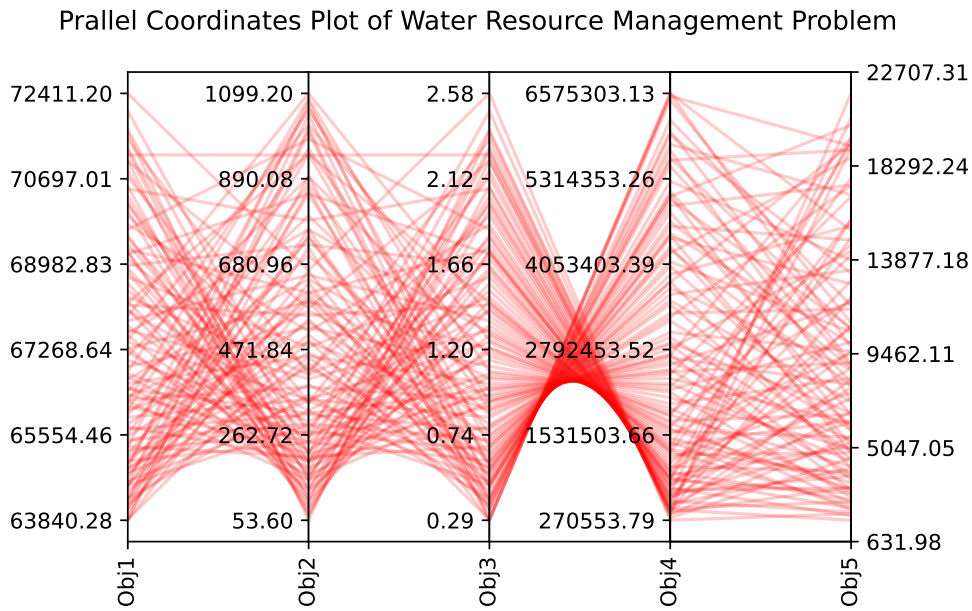


Figure 2.7: Parallel coordinate plot of objective scores from the 5 objective water resource management problem.

algorithms then comparing them becomes more challenging. For this reason, Empirical Attainment Difference Functions (EAF) have been developed [96]. In the EAF difference plots the dark areas mark where the two algorithms have obtained different results. The more frequent a certain area is dominated by an algorithm the darker the grayscale is. An example of an EAF difference plot on the TBTD problem is given in Figure 2.8. The EAF difference plot again confirms what was shown earlier in Figure 2.6, the SAMO-COBRA algorithm manages to find the minimum values of objective 2 on the Pareto frontier, while SA-NSGA-II found smaller values for objective 1.

Convergence Plots

On convergence plots, the x-axis typically shows the number of function evaluations required to achieve a performance score that is on the y-axis. By analyzing the convergence plots it can be identified after how many evaluations the algorithm has found a good (set of) optimal solution(s) and has converged. If multiple algorithms (or algorithm configurations) are plotted together, the convergence can be compared. An

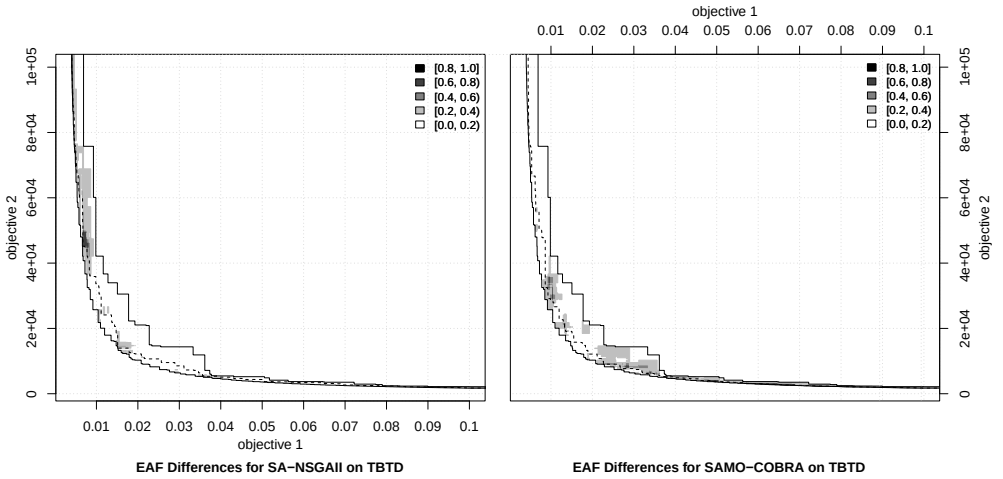


Figure 2.8: Empirical Attainment Difference Function plot on TBTD problem for comparing 10 independent SA-NSGA-II and 10 SAMO-COBRA runs.

illustrative example for a convergence plot on the TBTD problem is given in Figure 2.9.

In the convergence plot where the hypervolume is to be maximized, it is shown that the SAMO-COBRA algorithm with batch size 1 achieves the highest hypervolume after 120 function evaluations. Besides the final score the convergence plot also shows the performance after fewer function evaluations. Inspection of this tells us that the SAMO-COBRA with batch size 1 obtains the best results and converges faster compared to the other configurations.

Empirical Cumulative Distribution Functions

Empirical Cumulative Distribution Functions (ECDF) [76, 166] are used to visualize the convergence of the different algorithms on a set of test functions simultaneously in one plot. An ECDF plot is based on a set of target values that are linearly distributed (sometimes also other distributions are chosen) between zero and the maximum achievable performance score per test function. The proportion of target values attained by the algorithm is the score reported on the ECDF curve. To be able to visualize the performance of an algorithm on multiple different benchmark functions, the corresponding ECDF target score proportions are aggregated. This then results in a curve that grows as more target values are reached. A formal description, adapted from [166], is given in the following definition:

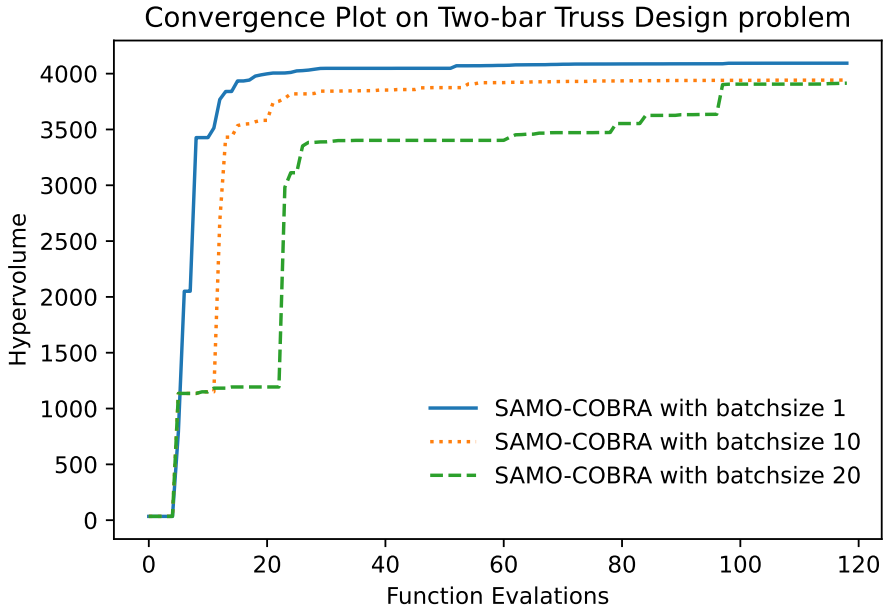


Figure 2.9: Illustrative example of a convergence plot of SAMO-COBRA algorithm with different batch sizes on TBTD problem.

Definition 2.11. An ECDF curve requires to select a set $\{v_1, \ldots, v_n\}$ of target values. The ECDF shows for each budget t the fraction $| \{(i,j) \mid 1 \leq i \leq r, 1 \leq j \leq n\} | / r \cdot n$ of the (run r, target value) pairs (i, v_j) that satisfy that $V(A, f, t, i) \geq v_j$. Here $V(A, f, t, i)$ denotes the function value of the best among the first t evaluated solution candidates in run i. So the ECDF is expressed as $\hat{F}(t) = \frac{1}{nr} \sum_{j=1}^{n} \sum_{i=1}^{r} \mathbf{1}_{V(A,f,t,i) \geq v_j}$, where $\mathbf{1}_C$ denotes the indicator variable, which is one where the condition C is satisfied.

Two ECDF curves with hypervolume as the performance metric, 10 independent runs per benchmark problem from Section 2.4, are given in Figure 2.10. The ECDF curve of the SAMO-COBRA algorithm is completely above the results from the SA-NSGA-II algorithm. This shows that for each budget the SAMO-COBRA algorithm achieves a larger portion of the target values compared to the SA-NSGA-II algorithm. This means that on average the SAMO-COBRA algorithm finds a better hypervolume and also on average converges faster.

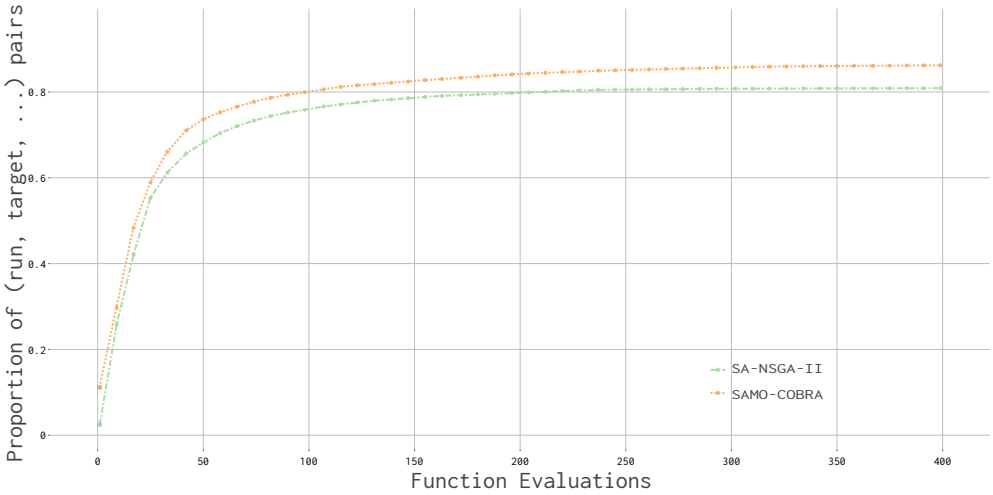


Figure 2.10: ECDF Curve that aggregates the results for the SA-NSGA-II and SAMO-COBRA algorithm that have optimized all benchmark problems from section 2.4 10 times.