Thesis for the Degree of Master of Science

A surrogate-based parameter tuning heuristic for Carmen crew optimizers

---

Staffan Häglund

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
GOTHENBURG UNIVERSITY
Göteborg, Sweden 2010

CHALMERS UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences

Visiting address:
Chalmers Tvärgata 3
Göteborg, Sweden

Postal address:
Mathematical Sciences, Chalmers Univ. of Technology and Göteborg Univ.
412 96 Göteborg

Telephone:
031-772 1000

Telefax:
031-16 19 73

Web page:
http://www.chalmers.se/math/

Abstract

This project has been performed at Jeppesen Systems AB (formerly Carmen Systems) in Gothenburg, a company making optimization software for the airline and railway industry.

Crew planning for airlines and railway leads to large and difficult optimization problems, governed by many hundreds of rules and parameters set by the government, the union and the airlines themselves. The rules and parameters set by the airlines can be changed in order to create better solutions for the crew planning but they interact in non-trivial ways making manual parameter tuning difficult.

In this thesis an algorithm is developed and implemented that automates the selection of parameter values to find the best solution. The problem is a so-called expensive black-box optimization problem and the chosen algorithm is based on surrogate modeling with radial basis functions (RBF) and the use of a merit function selecting promising parameter settings to investigate. The model also includes expensive black-box constraints as well as several objectives.

The algorithm is implemented into a GUI, connecting to the crew optimizers used at Jeppesen Systems and allowing for parameter tuning for their heavy optimization runs. The algorithm and program are then tested and analyzed assessing their performances.

A surrogate-based parameter tuning heuristic for Carmen crew optimizers

Staffan Häglund

Author:	Staffan Häglund, Engineering Mathematics Master's Program, Chalmers University of Technology
Supervisor:	Andreas Westerlund, Optimization Expert, Jeppesen Systems AB
Examiner:	Michael Patriksson, Professor, Mathematical Sciences, Chalmers Univ. of Technology and Göteborg Univ.
Sponsor:	Jeppesen Systems AB

Acknowledgements

For their help in the work that has resulted in this thesis, I would like to thank a few people.

First of all I would like to thank the people at the pairing- and rostering departments at Jeppesen Systems for great few months which have been very rewarding. In particular I would like to thank my supervisor Andreas Westerlund for all the support I have been given and Henrik Delin for all the help with system related issues. I also want to extend thanks to Tomas Gustafsson, Head of Optimization at Jeppesen Systems, for the opportunity to do my thesis at Jeppesen Systems and for giving me the customer perspective.

Lastly I would like to thank Malin Nilsson for providing support and cheering me up whenever the programming troubled me.

Contents

1 Introduction and problem description

This is a Master's Thesis for the degree Master of Science in Engineering Mathematics at Chalmers University of Technology. The project has been performed at Jeppesen Systems in Gothenburg during spring and summer of 2010. At Jeppesen Systems software is produced to optimize, among other things, pairing and rostering optimization problems as a part of the crew planning for the airline and railway industry. The problems to be solved are modeled as generalized set partitioning problems with constraints determined by rules set by the union and legislation but also by the airline itself. The constraints arising from legislation and the union are considered fixed whereas the rules and parameters set by the airline can be changed to e.g. minimize cost. There are more than a hundred set of rules and parameters determined by the airline, so an exhaustive search, trying to find the best parameter setting, is impossible due to the combinatorial explosion and costly simulations. Therefore, in determining which parameter values to choose, experience and empirics must be used to determine which parameters has the most effect on e.g. the cost. The scope of this thesis concerns just this: attempting to automate and facilitate the selection of good parameter values.

The outline for this thesis is as follows. In the remainder of this section the airline planning process and terminology is presented, leading up to a black-box optimization problem. This optimization problem is characterized mathematically and some related research is discussed. Further, a mathematical model for this optimization problem is introduced. Moreover, the general requirements on the algorithm are discussed as well as the specified goal of the master thesis work. In Section 2 some mathematical theory is presented giving the reader a background to the discussion that follows. Topics such as multiobjective optimization, surrogate modeling, merit functions and experimental design are discussed. In Section 3 the different parts of the algorithm, which is implemented in this thesis, are presented, using and further discussing the theory in Section 2. Further, Section 4 presents the algorithm implementation: how the parts in the previous sections are pieced together into an algorithm as well as some discussion regarding implementation. In order to use the algorithm on Jeppesen's pairing and rostering problems it is implemented into a GUI which is described in Section 5. Lastly, in Section 6, Section 7 and Section 8 some numerical results are presented and discussed.

1.1 About Jeppesen

Jeppesen Systems AB is a subdivision of Jeppesen which is a subsidiary of Boeing Commercial Aviation Services. Until 2006, when bought by Jeppesen, Jeppesen Systems AB was an independent company called Carmen Systems AB which is a name that can still be seen for example in names of products. In this report Jeppesen will refer to Jeppesen Systems AB. Jeppesen Systems AB has its headquarters in Gothenburg with approximately 300 employees and this is also where the work for this thesis has been carried out.

Jeppesen Systems AB is a software company developing products mainly for optimizing planning and scheduling for the airline and railway industry with emphasis on the airlines. As much as 25 % of the world's airline crews are in some way scheduled using Carmen software [Gus09]. Jeppesen was also given the prestigious Edelmann Award in 2000 for their work on making the handling of flight maps more efficient as well as the INFORMS Prize in 2010 for outstanding organization-wide use of operations research throughout the company [Gus09, INF10].

1.2 Terminology

The airline industry has its own terminology. The most basic unit in a schedule is a *leg* which is a non-stop flight from one airport to another. A *flight* is both a commercial and administrative

concept but in the administrative sense a flight consists of (usually) one or several legs and it has a unique combination of flight carrier and flight number per day.

A sequence of legs constituting a day's work for a crew member (i.e. work between rest periods) is called a *duty*. A sequence of duties beginning and ending at the same base is called a *trip* or a *pairing* where a crew member always must start and end at its home base. As an example SAS has several home bases, where CPH, ARN and OSL are the three largest ones.

From these trips a crew member's individual schedule is created, called *roster*. This specifies which trips that are assigned to the particular crew member as well as vacation, training, reserve duty etc. The schedules are given to the crew usually 2-6 weeks in advance depending on the airline.

If legs do not match up in the roster, meaning that the end of a flight is not at the same airport as the start of the next, transit is required. This transit is called *deadheading* and can be done by train, taxi or airplane. If the deadhead takes place with another airline's flight, it is usually called an OAG[1] flight. The deadheading, or OAG, is planned as a passive leg in the crew's roster [Jep].

1.2.1 Airline planning

The overall method for planning is rather well established in the airline community. Presented somewhat simplified it consists of the following six steps

1. Timetable construction
2. Fleet Assignment
3. Tail Assignment
4. Crew Pairing
5. Crew Rostering
6. Crew Tracking / Day of Operation

where the output from step 1 serves as input for step 2 and so on. Usually, each step is treated as a separate optimization problem. However, the trend is that a pure step-by-step optimization method is avoided more and more since that may lead to sub-optimization. Information from subsequent steps are needed earlier in the planning process in order to achieve good results. Therefore, a current topic of research is to integrate two or more steps into a new, single optimization problem.

The timetable is usually created one year in advance where the previous traffic patterns as well as estimates of passenger demand to and from airports are considered. This is done trying to maximize passenger revenue with the constraints given by the available aircrafts and the timeslots available at the airports.

An airline usually has several kinds of aircrafts (fleet) taking different number of passengers, requiring different number of crew etc. The fleet assignment problem is the task of matching the aircraft types to the different legs to maximize profit. This is done taking into consideration passenger demand on certain legs. In tail assignment a particular aircraft is assigned to each flight, giving each aircraft a "personal" schedule where also maintenance requirements need to be considered.

The crew planning problem is considered to be a very hard problem. It is therefore split into two parts: *crew pairing* and *crew rostering*. In crew pairing, trips (also referred to as pairings)

[1] OAG = Official Airline Guide

are created from legs. All trips should together cover every leg to its crew need: different flights need different number of crew e.g. some airplanes require one flight attendant, one purser, one pilot and one co-pilot whereas other require additional crew. The trips are anonymous, meaning that they have not yet been assigned to any particular crew member. In order to make a good paring, information about the crew is needed: the number of crew, their home bases etc. Assigning the trips to crew members is the major task in crew rostering. Also other activities such as vacation, training, reserve duty etc. are assigned to crew members.

All planning steps 1 to 5 are finished at least two weeks before operation and in an ideal world the planning would end there. However, people get sick, flights are delayed etc. which require changes in the planning. This is done from roster publication to the day of operation and is called Crew Tracking. Then there is "Day of Operation" which includes the real-time changes during operation.

The general planning process described is rather well established as mentioned earlier and therefore it is also used in the Carmen software produced by Jeppesen. There are a lot of software and code required for each of the planning steps, and at Jeppesen, the most important GUI used to connect to all their software routines, is called Studio. In Studio there are tools for importing timetables and OAG files, building rotations, switching rules on and off, changing parameters, building/modifying duties, creating reports etc.

This thesis concerns the crew planning, hence the crew planning steps: pairing and rostering. There are different optimization solvers for the pairing and rostering problems that utilize their specific structure. A relevant model to introduce for these problems is a generalized set partitioning model:

$$\begin{aligned} \min f \quad &= \mathbf{c}^T\mathbf{v}, \\ s.t \qquad A\mathbf{v} &= \mathbf{1}^{\bar{m}} && \text{(1a)} \\ \mathbf{h}(\mathbf{v}) &\leq \mathbf{0}^{\bar{k}} && \text{(1b)} \\ \mathbf{v} &\in \{0,1\}^{\bar{n}}, && \text{(1c)} \end{aligned}$$

with the objective coefficients c_j, binary variables $\mathbf{v} = (v_1, \ldots, v_{\bar{n}})$, binary constraints matrix A and function $\mathbf{h}(\mathbf{v})$. Ignoring constraints (1b), the classic set partitioning problem is obtained which is very well studied [BP76, FK90, GS09].

In simplified terms both the pairing and rostering problems can be modeled as generalized set partitioning problems. For example, the pairing problem can be described as follows: Given $\bar{n}$ trips and $\bar{m}$ legs, introduce variables $v_j \in \{0,1\}$ where $v_j = 1$ if trip j is used and $v_j = 0$ otherwise. Each trip has a cost c_j and it is of interest to minimize the cost function $\mathbf{c}^T\mathbf{v}$. Each trip has to be covered exactly once so the $\bar{m} \times \bar{n}$ constraint matrix $A = \{a_{ij}\}$ is created where $a_{ij} = 1$ if leg i is a part of trip j and 0 otherwise. The number of columns in A is typically huge and hence the columns can normally not be completely enumerated, requiring solution methods where enumeration is not performed. One such method that is widely used is *column generation*. Further, there are also other constraints, given by (1b), that are added. An example of such constraints are base constraints which can be obtained by putting $\mathbf{h}(\mathbf{v}) = B\mathbf{v} - \mathbf{p}$. The base constraints give limits on total amount of work produced by crew with a particular home base where $\mathbf{p}$ is the limit on the base production [Gus99].

In theory the pairing problem is modeled as a generalized set partitioning problem but for implementation it is modeled as a generalized set covering problem, hence allowing more than one crew on each position on the flights. In doing so, among other things, more flexibility is allowed with respect to deadheads and it gives faster convergence for most larger problems [Gus99]. The generalized set covering problem can be formulated as in (1) but with (1b) replaced by $A\mathbf{v} \geq \mathbf{1}^{\bar{m}}$.

The rostering optimization problem can also be formulated as a generalized set partitioning problem, see (1), but rosters are chosen instead of trips, and they are chosen in such a way that all trips are covered exactly once, together with other tasks such as reserve duty and training [KK04].

1.2.2 Rules and parameters

For an airline there are thousands of rules and regulations that must be considered when scheduling the crew. There are different kinds of rules determined by the government, union and the airlines themselves. The first kind, legal rules, determined by the government usually concerns safety, requiring that the crew have sufficient rest and the right training. Union rules usually try to improve the social quality of the crew, for example not allowing too long trips so that crew do not have to stay away from home base for too long. The rules and parameters set by the airlines are used to improve the quality of the solution. For example, the pairing optimizer has a tendency to produce longer trips since it increases productivity. However, longer trips are harder to fit into good rosters. Therefore the airline sets a limit on how long trips can be since they know empirically that the end cost will be higher if they do not. The rules set by the airlines are usually on/off; like "no more than 4 aircraft changes per duty", but they also have parameters that can be tuned to find the best solution[2]. Also the penalties discussed later fall into the class of parameters.

For the Pairing case, the rules determine if a pairing is legal, where a legal pairing corresponds to one column in the constraints matrix A in (1). Therefore not all possible combinations of legs will constitute a column in A, however, the number of legal pairings is usually still huge, rendering enumeration impossible or at least impractical.

What is the best solution is something that can be discussed and there are three main aspects that have to be taken into account: social quality, stability and the real cost. Social quality are characteristics of a trip that a large majority of the crew members consider good, such as no repetitive duties in a trip, no long connection times, not more than one early start in a trip etc. Stability means that the roster is not very sensitive to delays and other changes. Stability is increased by for example making sure that the connection time between legs is not too short and that there are no aircraft changes within a duty [Jep]. These measures of quality are quantified by Key Performance Indicators.

1.2.3 Key performance indicators

For the Pairing in the Carmen Products, a solution is evaluated by *Key Performance Indicators* (KPIs) which measures different aspects of quality, stability and cost, for example

- Total Cost
- Deadhead Time
- Aircraft Changes
- Duty Time Per Working Day
- APC[3] Total Rule Cost

[2] Also rules can be tuned but it is not as common. However, it might not be known whether a rule should say "no more than 4" or "no more than 5 aircraft changes per duty". Then the p in the rule "no more than p aircraft changes per duty" can be seen as a parameter corresponding to the rule in question.

[3] APC=Automated Pairing Construction

The Total Cost is usually the KPI that is minimized and it includes what is called *real costs* and *soft costs*. The real costs are real world costs such as duty-day costs, hotel nights, OAG costs, overtime costs etc. The soft costs are penalties for lack of quality and poor stability including short connection penalty, excessive aircraft change penalty etc.

The costs are measured in the fictive currency Carmen dollars. If a rule may be violated but it is preferred that it is not, it should be seen as a quality issue and be a penalty instead. For example, the rule "do not allow crew change after charter flights" could be seen as the penalty "crew change after charter flights should be penalized by 5000 Carmen $" [Jep]. Penalties and rules are usually combined e.g. 1 aircraft change per duty is okay where the penalty is linearly or nonlinearly increased for 2, 3 and 4 aircraft changes per duty and a rule is set that there cannot be more than 4.

So consider the real costs H_j for a trip j as well as the parameters/rules r_k, $k = 1,2,\ldots,\mathcal{M}$, then the cost c_j for trip j as given in (1) can be seen as composed of H_j as well as the penalty functions ψ_k for the parameters/rules r_k:

$$c_j = H_j + \sum_{k=1}^{\mathcal{M}} \psi_k(r_k).$$

The penalty function ψ_k increases when approaching undesirable values for parameter/rule r_k and when an absolute rule is set, e.g. $r_k \in \mathbb{Z}$ must not exceed 4, then $\psi_k = \infty$ for $r_k > 4$.

1.3 Problem description

As described earlier, the task of setting values for parameters to achieve the best solution is a difficult problem and experience and empirics must be used. The scope of this thesis concerns the automation of this and the goal of the project is two-folded. The goal involves the parts *Parameter Tuning* and *Prediction Model*, namely

Parameter Tuning

- Tune in parameters for shorter simulation times/better quality.
- Give "the best" parameter setting for a given Carmen crew optimization problem.

Prediction Model

- Create a prediction model to determine the KPI values (e.g. cost) for any set of parameters without any additional expensive evaluations.

For the parameter tuning there are some requirements and restrictions which highly determine which methods to use. First, a number d of parameters are chosen and bounds are set for them, creating the search domain Ω. The number of parameters usually should not exceed $d = 10$. Some of the d parameters are integer valued, some are binary and some can be seen as continuous.

One KPI shall be chosen as an objective function to be minimized although support for two or more KPIs as objectives would be good. With the objective(s) chosen there may be more KPIs that there is no interest in minimizing but only keeping within some bounds, call them KPI constraints or output constraints. It should be possible to choose a number η of those KPI constraints and set their bounds constituting their range[4] $\mathcal{D}$. Then the feasible solutions should have the corresponding KPI constraints' values within the bounds $\mathcal{D}$.

Given this the goal for the Parameter Tuning can be reformulated as:

[4]The most commonly used definition of range is adopted where for $f : U \to V$ the range is $f(U)$, hence the image of U under f.

(I) *Give the lowest value(s) for the KPI objective(s) in the bounded parameter domain* Ω *given that the KPI constraints are within the bounds* $\mathcal{D}$.

In addition, no assumptions can be made concerning the objective KPI(s) nor the KPI constraints when it comes to monotonicity or any other structural information.

The parameter tuning and the prediction model shall be implemented in RUSA (RUle Sensitivity Analysis), a program written as a thesis project in 2008 by Joel Driessen [Dri08]. The programming language used in RUSA is Python hence in order to incorporate the additional code into RUSA it should preferably be in Python. The implementation must also allow for parallel function evaluations. The requirements for the GUI features for the additions to RUSA are discussed in Section 5.

So, a quick review of the requirements and limitations:

1. d parameters with user-defined upper and lower bounds constituting Ω.
2. Some dimensions are integer valued, others are continuous.
3. One KPI as objective but preferably providing support for more.
4. KPI constraints within user-defined bounds $\mathcal{D}$.
5. No assumptions about the objective KPI(s) nor the KPI constraints.
6. Implementation in RUSA (Python).
7. Parallel computations must be used.

1.4 Costly global optimization

Problem (I) described in Section 1.3 can be classified as an expensive black-box optimization problem. Information about the objective is only given through sampling and every evaluation of the objective is expensive with respect to CPU time. Further, there are black-box constraints where information is also only given through sampling. The black-box (KPI) constraints are given by an upper bound u_i and a lower bound l_i where the upper bound can be infinity. The parameter domain Ω is also given by finite box-constraints and some dimensions are integer valued.

Given this, the problem at hand can be classified as a mixed-integer costly (expensive) global black-box nonconvex problem with nonlinear black-box constraints. Introduce the following sets: $\mathcal{I} = \{i : 1 \leq i \leq \eta\}$ and $\mathcal{J} = \{j : 1 \leq j \leq d\}$ where d is the dimension of the parameter domain and η is the number of output constraints. Then the problem can be formulated mathematically as

$$\begin{array}{lll} \min & f(\chi_1, \chi_2, \ldots, \chi_d) & \\ s.t. & l_i \leq g_i(\chi_1, \chi_2, \ldots, \chi_d) & \leq u_i, \quad i \in \mathcal{I} \\ & l_j^v \leq \chi_j & \leq u_j^v, \quad j \in \mathcal{J} \\ & \chi_j \in \mathbb{Z} \ \ \forall j \in \mathbb{I} & \end{array} \tag{2}$$

where $l_i \leq g_i(\chi_1, \chi_2, \ldots, \chi_d) \leq u_i, i \in \mathcal{I}$ are the black-box constraints, in this thesis also called output constraints or KPI constraints depending on the context. The decision vector

$$\boldsymbol{\chi} = (\chi_1, \chi_2, \ldots, \chi_d) \in \mathbb{R}^d \tag{3}$$

is bounded by box-constraints and the parameters whose indices are given by $\mathbb{I}$ are integer valued, $\mathbb{I} = \{j : \chi_j \in \mathbb{Z}\}$. A solution with e.g. $\chi_j = 4$ corresponds to the j^{th} parameter having a value 4.

Denoting the box-constrained set of decision vectors by

$$\Omega = \{\boldsymbol{\chi} = (\chi_1, \ldots, \chi_d) : l_j^v \leq \chi_j \leq u_j^v, j \in \mathcal{J}\}, \tag{4}$$

where the superscript v stands for *variable*, the set of feasible output constraints vectors by

$$\mathcal{D} = \{\mathbf{g}(\boldsymbol{\chi}) = (g_1(\boldsymbol{\chi}), \ldots, g_\eta(\boldsymbol{\chi})) : l_i \leq g_i(\boldsymbol{\chi}) \leq u_i, i \in \mathcal{I}\}, \tag{5}$$

the set of feasible decision vectors taken into account integrality and output constraints by

$$\Omega_{\mathbb{I}\mathcal{D}} = \{\boldsymbol{\chi} = (\chi_1, \ldots, \chi_d) \in \Omega : \chi_j \in \mathbb{Z} \ \ \forall j \in \mathbb{I}, \ \mathbf{g}(\boldsymbol{\chi}) \in \mathcal{D}\}, \tag{6}$$

the problem can be written on a more compact form:

$$\boldsymbol{\chi}^* = \arg \min_{\boldsymbol{\chi} \in \Omega_{\mathbb{I}\mathcal{D}}} f(\boldsymbol{\chi}). \tag{7}$$

The task of global optimization is to find the set of parameters in the feasible region $\Omega_{\mathbb{I}\mathcal{D}}$ for which the objective function obtains its lowest value. In other words $\boldsymbol{\chi}^*$ is a *global optimizer* to (2) if $f(\boldsymbol{\chi}^*) \leq f(\boldsymbol{\chi})$ for all $\boldsymbol{\chi} \in \Omega_{\mathbb{I}\mathcal{D}}$. On the other hand $\hat{\boldsymbol{\chi}}$ is a *local optimizer* of (2) if $f(\hat{\boldsymbol{\chi}}) \leq f(\boldsymbol{\chi})$ for all $\boldsymbol{\chi} \in \Omega_{\mathbb{I}\mathcal{D}}$ in some neighborhood of $\hat{\boldsymbol{\chi}}$. When the objective has several local minima there could obviously be minima that are local but not global and hence local search methods are bound to get stuck. Therefore some global search method is needed to find the global minimum with some level of reliability.

It should be noted that in (2) nothing says that the feasible domain without the integrality constraints, $\{\boldsymbol{\chi} \in \Omega : \mathbf{g}(\boldsymbol{\chi}) \in \mathcal{D}\}$, is connected which complicates things since proofs of convergence for many methods are based on compactness of the domain and that a generated sequence of points is dense in the domain [Gut01, RS05, HQE08]. However, generating even close to a dense set of points within reasonable time with an expensive objective function is impossible. Even so, a non-connected feasible domain is always harder to optimize over in general. It is not given that the objective function $f(\boldsymbol{\chi})$ is continuous although it is approximated as such since otherwise the problem is a lot harder to solve.

In order to solve (2) a method is needed for problems where

1. The computation of $f(\boldsymbol{\chi})$ is very expensive.
2. The function is a black box; hence no analytical derivatives are available.
3. The search domain is subjected to box constraints and nonlinear black-box constraints.
4. Some dimensions are integer valued.
5. The method should rather easily be extended to multiobjective optimization.

The optimization of expensive black-box functions is a challenging problem and several approaches have been suggested in the literature. The larger parts of them are based on response surface techniques, most of which need to utilize every computed function value. An excellent review of the most important development is given by Jones in [Jon01].

Many surrogate methods have been developed using statistical approaches, called Kriging, see e.g. [Gut01, JSW98] where the basics are discussed in Section 2.1.3. There are also methods based on radial basis function interpolation, RBF methods, first discussed in [Gut01, Pow99]. The idea of the general RBF algorithm is to create a surrogate model of the expensive black-box function using evaluated points and then use merit functions to find new points to evaluate. The merit function, depending on the surrogate, is inexpensive to evaluate and to optimize the

merit function an external global optimization solver is usually used. If there are e.g. non-linear and integrality constraints involved it is common to use an external global optimization solver that can handle non-linear and integrality constraints. Expensive constraints can be treated by adding them to the objective function in the following way [HQE08, Qut09]

$$\hat{f}(\boldsymbol{\chi}) = f(\boldsymbol{\chi}) + \sum_{i \in \mathcal{I}} w_i \max\left\{0, g_i(\boldsymbol{\chi}) - l_i, u_i - g_i(\boldsymbol{\chi})\right\}, \tag{8}$$

and minimize $\hat{f}(\boldsymbol{\chi})$ instead of $f(\boldsymbol{\chi})$ where w_i are weights and l_i and u_i are lower and upper bounds respectively for output constraint i. The difficulty for the problem in this thesis is that the constraints are expensive but they are also black-box, hence no knowledge of them are given beforehand and every estimation of them are rough at best. Also no commercial solvers, such as the global optimization solvers in the TOMLAB package, can be used due to licensing reasons.

A few methods from the literature are the *Efficient Global Optimization Algorithm* (EGO) [JSW98] which uses the Kriging framework and so does Gutmann's bumpiness minimization method [Gut01]. Examples of methods using radial basis functions are CORS [RS05] and the qualSolve algorithm [JPRW09]. The three methods mentioned last are discussed in more detail in Section 2.

There are also methods based on pattern search such as the Generalized Pattern Search (GPS) algorithm [Tor97] and the Mesh Adaptive Direct Search (MADS) method [AJD06, AO06] which is an extension of the former.

The surrogate modeling method adopted in this thesis is the method of radial basis functions. The algorithm includes surrogate modeling, merit functions as well as experimental design and these are the subjects discussed in Section 2.

2 Theory and background

In this section some theory and background is presented that is used when constructing the algorithm. Different approaches are presented and discussed and these will be the base for the choice in the algorithm implementation.

This section is more mathematical in nature with emphasis on theory and background and reading it will give much insight into the choices made in the following sections. However, for the casual reader it is not necessary to thoroughly study this section in order to enjoy the rest of the thesis. Instead, when required in the subsequent sections, this section can be used as a reference.

First, in Section 2.1 two versions of surrogate modeling are presented where emphasis is on the RBF interpolation. That is followed by Section 2.2 discussing different ways of creating merit functions: one-stage and two-stage methods as well as three examples of merit functions where the Quality function is the one of most interest. In Section 2.3 some experimental design techniques are discussed and in Section 2.4 the topic is the global optimization solver DIRECT. From these approaches, the parts most suitable for problem (I) in Section 1.3 are selected and in some cases further discussed, which is described in Section 3.

2.1 Surrogate modeling

A surrogate model, or response surface, is an interpolation of sampled points, predicting the costly function for points not yet sampled. Jones [Jon01] discusses that non-interpolating surfaces such as fitted quadratic surfaces are unreliable because the function may not sufficiently capture the shape of the function. Instead it is better to interpolate the data and for that there are two widely used methods: Radial Basis Functions and the Kriging method (DACE).

2.1.1 Interpolation of scattered data in $\mathbb{R}$

Interpolating data is the task of constructing new data points within the range of a discrete set of known data points, or put in a different way, finding a function that corresponds to the data for some discrete set of points. More formally it can be stated that given points $X = \{x_1, x_2, \ldots, x_n\}$ in $\mathbb{R}$ and corresponding function values $\mathbf{f} = \{f_1, f_2, \ldots, f_n\}$, a function $P(x)$ (e.g. in C^0) is sought such that

$$P(x_k) = f_k, \forall k \in \{1,2,\ldots,n\}.$$

Since the set of continuous functions is an infinite dimensional function space interpolation is not at all unique, but limiting oneself to one dimension and e.g. the space of polynomials of degree less than n, denoted Π_{n-1}, then uniqueness holds. For Π_{n-1} it is given that for points $x_1, x_2, \ldots, x_n \in \mathbb{R}$ the following holds

$$P(x_k) = \sum_{l=1}^{n} c_l x_k^{l-1}, \forall k \in \{1,2,\ldots,n\}$$

which is equivalent to solving the linear system $\hat{A}\mathbf{c} = \mathbf{b}$ where $\hat{A}_{kl} = x_k^{l-1}$, $b_k = f_k$ and $\mathbf{c} = (c_1, \ldots, c_n)$ is the vector of coefficients. Since Π_{n-1} is isomorfic to $\mathbb{R}^n$ there is a unique solution if and only if all points x_k are distinct. The function space Π_{n-1} has several desirable properties, especially that it is independent of the points $\{x_k\}$. Although this approach gives a unique solution it is not practical to use in implementations since the polynomial order increases with the number of points and that usually leads to large oscillations.

A more appealing approach which avoids the problem mentioned is the use of *splines* where piecewise polynomial functions are used to interpolate the data. Any order of the splines can be used but cubic splines are a common choice. Consider points $a = x_1 < x_2 < \ldots < x_n = b$ and the intervals $J_k = [x_k, x_{k+1}]$ where the polynomials are defined on each of these intervals. With a function space

$$\mathcal{S}_3(X) = \{S \in C^2((a,b)) : S|_{J_k} \in \Pi_3(\mathbb{R}), k = 1,2,\ldots,n\}$$

the problem can be formulated as finding $S(x) \in S_3(X)$ such that $S(x_k) = f_k, k \in \{1,2,\ldots,n\}$. This function space however is not independent of the set of points X. Moreover, with n points, hence $n-1$ splines each with four degrees of freedom, there are $4(n-1)$ degrees of freedom in total. At each inner node, constraints are set for the function values as well as first- and second derivatives in order to ensure that $S \in C^2((a,b))$. Therefore $\mathcal{S}_3(X)$ is an $n+2$ dimensional function space, but the data only gives n degrees of freedom, hence the problem is not uniquely solvable. There are different ways of modifying the function space to ensure that a unique solution exists. With the conditions that the second derivative is zero at the end points, the Natural Cubic Spline ($\mathcal{NS}_3(X)$) is given and another condition is the so-called "not-a-knot" condition: $\mathrm{d}^3S/\mathrm{d}x^3 = 0$ for $x \in \{x_2, x_{n-1}\}$.

In comparison, the function spaces $\mathcal{S}_3(X)$ and $\mathcal{NS}_3(X)$ are not location independent which is a drawback compared to Π_3, however there are no large oscillations. In trying to extend this to higher dimensions there are several discouraging problems e.g. ordering the points can be done using triangles in two dimensions but even for that case the order of the function space is generally unknown. This method is obviously not suitable for higher dimensions although there are ways to reformulate the natural spline problem.

Natural cubic spline interpolants to functions in one variable are given as a solution to a variational calculation. Powell [Pow92] shows that given data $X = \{x_k : k = 1,2,\ldots,n\}$ and $\{f(x_k) : k = 1,2,\ldots,n\}$ and a function $S(x)$ with a square integrable second derivative such that $f(x_k) = S(x_k), k = 1,2,\ldots,n$, then

$$\mathcal{B}(S) = \int_{-\infty}^{\infty} \left[S''(x)\right]^2 dx \tag{9}$$

is minimized under the interpolation requirements $f(x_k) = S(x_k), k = 1,2,\ldots,n$ if and only if $S(x)$ has the following form

$$S(x) = \sum_{k=1}^{n} \lambda_k \phi(|x - x_k|) + p(x)$$

where

$$\sum_{k=1}^{n} \lambda_k = \sum_{k=1}^{n} \lambda_k x_k = 0$$

and where $\phi(r) = r^3, r \geq 0$ and $p(x) = \hat{a}x + \hat{b}$. This discussion can be generalized further and it leads us to consider multivariate splines using radial basis functions. The expression for $\mathcal{B}(S)$ in (9) is a measure of how much the function $S(x)$ is alternating since $S''(x)$ denotes the curvature. Minimizing a measure of the curvature will give a non-bumpy surface.

2.1.2 Interpolation and approximation using RBF

A radial basis function is a function $\phi(\mathbf{x}) = \phi(||\mathbf{x}||)$: hence ϕ is a function of the Euclidean distance to the origin only, $\phi = \phi(r)$. The radial basis function interpolation problem is as

follows. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ be any given set of pairwise different points with corresponding data $f_1, \ldots, f_n \in \mathbb{R}$ where n and d are any positive integers. We seek a function $S(\mathbf{x})$ that interpolates the data $\{(\mathbf{x}_i, f_i) : i = 1,2,\ldots,n\}$, hence that satisfies

$$\begin{aligned} S(\mathbf{x}) &= \sum_{i=1}^{n} \lambda_i \phi(||\mathbf{x} - \mathbf{x}_i||), \quad \mathbf{x} \in \mathbb{R}^d \\ S(\mathbf{x}_i) &= f_i, \ \ i = 1,2,\ldots,n \end{aligned} \tag{10}$$

or equivalently $\Phi\lambda = \mathbf{b}$ where $\Phi_{ij} = \phi(||\mathbf{x}_i - \mathbf{x}_j||)$, $b_i = f_i$ and $\lambda = (\lambda_1, \ldots, \lambda_n)$. The coefficients λ_i are real numbers and the norm $||\cdot||$ is the Euclidean norm in $\mathbb{R}^d$. The matrix equation $\Phi\lambda = \mathbf{b}$ can be solved uniquely to determine the vector λ if and only if Φ is invertible. Since Φ depends on the radial function ϕ it is safe to assume that the invertability of Φ depends on the choice of radial function. In order to find necessary restrictions on the function space to ensure a unique solution, some definitions are introduced.

Positive definiteness
A continuous function $\tilde{\Phi} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is positive definite if for every set of pairwise different points $\mathbf{x}_1, \cdots \mathbf{x}_n$ and every $\mathbf{c} = (c_1, \cdots, c_n) \in \mathbb{R}^n \backslash \{0\}$ it holds that

$$\sum_{i,j=1}^{n} c_i c_j \tilde{\Phi}(\mathbf{x}_i, \mathbf{x}_j) > 0. \tag{11}$$

By choosing $\tilde{\Phi}$ as a positive definite radial function it holds that $\tilde{\Phi}(\mathbf{x}_i, \mathbf{x}_j) = \phi(||\mathbf{x}_i - \mathbf{x}_j||)$ and hence

$$\mathbf{c}^T \Phi \mathbf{c} = \sum_{i,j=1}^{n} c_i c_j \phi(||\mathbf{x}_i - \mathbf{x}_j||) > 0 \tag{12}$$

where $\Phi_{ij} = \phi(||\mathbf{x}_i - \mathbf{x}_j||)$ are the elements of the positive definite matrix Φ.

It is only necessary to consider the positive definite case since if $\mathbf{c}^T\Phi\mathbf{c} < 0$ then $-\mathbf{c}^T\Phi\mathbf{c} > 0$ hence then $-\Phi$ is positive definite. Ensuring that a matrix is positive (negative) definite also ensures that the matrix is invertible and hence the radial basis function interpolation has a unique solution. In developing a criterion for positive definiteness for radial functions the concept of a completely monotone function is required.

Completely monotone [Mic86, Fas03]
A function $\varphi : [0, \infty) \to \mathbb{R}$ which is in $C[0, \infty) \cup C^\infty(0, \infty)$ and which satisfies

$$(-1)^l \varphi^{(l)}(r) \geq 0, \quad r > 0, \ \ l = 0,1,2,\cdots$$

is called completely monotone on $[0,\infty)$

Criterion for positive definiteness for radial functions [Fas03]
A radial function φ is completely monotone on $[0, \infty)$ if and only if $\varphi(||\cdot||^2)$ is positive definite on $\mathbb{R}^d$ for all d.

This criterion can easily be used to determine which radial functions that are positive definite and it is easy to show that the previously mentioned spline r^3 is not. In order to extend the family of functions that can be used as interpolates, the requirement of positive definiteness is relaxed and the concept of conditional positive definiteness is introduced.

Conditional positive definiteness [Mic86]
A continuous function $\tilde{\Phi} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ *is conditionally positive definite of order* m *if for every set of pairwise different points* $\mathbf{x}_1, \cdots \mathbf{x}_n$ *and every* $\mathbf{c} = (c_1, \cdots, c_n) \in \mathcal{V}_m \backslash \{0\}$ *it holds that*

$$\sum_{i,j=1}^{n} c_i c_j \tilde{\Phi}(\mathbf{x}_i, \mathbf{x}_j) > 0$$

where

$$\mathcal{V}_m = \left\{ \mathbf{c} \in \mathbb{R}^n : \sum_{i=1}^{n} c_i p(\mathbf{x}_i) = 0, \quad \forall p \in \Pi_{m-1}(\mathbb{R}^d) \right\}. \tag{13}$$

By choosing ϕ such that $\tilde{\Phi}(\mathbf{x}_i, \mathbf{x}_j) = \phi(||\mathbf{x}_i - \mathbf{x}_j||)$ then ϕ is conditionally positive definite if the same holds for $\tilde{\Phi}$.

By using a conditionally positive radial function ϕ in creating the splines, positive definiteness can only be ensured for $\lambda \in \mathcal{V}_m$ hence the matrix Φ can become singular for $\lambda \notin \mathcal{V}_m$. Therefore it is necessary to reformulate problem (10) to ensure that a unique solution exists. The formulation is as follows [Pow92]. Given any set of pairwise different points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and data $f_1, \ldots, f_n \in \mathbb{R}$ find $\lambda \in \mathbb{R}^n$ and $\mu \in \mathbb{R}^{\hat{m}}$ such that

$$\begin{aligned} S(\mathbf{x}) &= \sum_{j=1}^{n} \lambda_j \phi(||\mathbf{x} - \mathbf{x}_j||) + \sum_{k=1}^{\hat{m}} \mu_k p_k(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d \\ S(\mathbf{x}_i) &= f_i, \quad i = 1,2,\ldots,n \\ \sum_{i=1}^{n} \lambda_i p_k(\mathbf{x}_i) &= 0, \quad k = 1,2,\ldots,\hat{m} \end{aligned} \tag{14}$$

where $\hat{m}$ is the dimension of $\Pi_m(\mathbb{R}^d)$ and $p_1, \cdots, p_{\hat{m}}$ are the basis of the space $\mathcal{V}_m$. Let P be the matrix

$$P = \begin{pmatrix} p_1(\mathbf{x}_1) & \cdots & p_{\hat{m}}(\mathbf{x}_1) \\ \vdots & & \vdots \\ p_1(\mathbf{x}_n) & \cdots & p_{\hat{m}}(\mathbf{x}_n) \end{pmatrix} \tag{15}$$

then $\mathcal{V}_m$ is the space of all $\mathbf{c} \in \mathbb{R}^n$ that satisfy $P^T \mathbf{c} = \mathbf{0}^{\hat{m}}$ and (14) can be written as a system of equations

$$\begin{pmatrix} \Phi & P \\ P^T & \mathbf{0}^{\hat{m} \times \hat{m}} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0}^{\hat{m}} \end{pmatrix} \tag{16}$$

where $\Phi_{ij} = \phi(||\mathbf{x}_i - \mathbf{x}_j||)$ and $P_{ij} = p_j(\mathbf{x}_i)$ and for a unique solution, the matrix

$$\mathcal{A} = \begin{pmatrix} \Phi & P \\ P^T & \mathbf{0}^{\hat{m} \times \hat{m}} \end{pmatrix} \in \mathbb{R}^{(n+\hat{m}) \times (n+\hat{m})} \tag{17}$$

is non-singular. We seek a condition on P (or actually on the set of points X) upon which $\mathcal{A}$ is non-singular and hence for which it can be proven that a unique solution exists. For that the notion of point set unisolvence is introduced.

Unisolvence
The points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$ *with* $n \geq \hat{m} = dim \Pi_m(\mathbb{R}^d)$ *are called* $\Pi_m(\mathbb{R}^d)$ *unisolvent if the zero polynomial is the only polynomial from* $\Pi_m(\mathbb{R}^d)$ *that vanishes at all of the points in* X*.* For example, assume that $\sum_{i=1}^{\hat{m}} \mu_i p_i(\mathbf{x}) \in \Pi_m(\mathbb{R}^d)$ and that $\sum_{i=1}^{\hat{m}} \mu_i p_i(\mathbf{x}) = 0 \ \forall \mathbf{x} \in X$ then it

follows that $\mu_i = 0$ for $i = 1,2,\ldots,\hat{m}$.

$\mathcal{A}$ is invertible
Consider $(\lambda, \mu)^T$ in the nullspace of $\mathcal{A}$. Then it follows that

$$\begin{aligned} \Phi\lambda + P\mu &= \mathbf{0}^n, \\ P^T\lambda &= \mathbf{0}^{\hat{m}}, \end{aligned} \tag{18}$$

and if $\lambda = \mathbf{0}^n, \mu = \mathbf{0}^{\hat{m}}$ then the nullspace of $\mathcal{A}$ is the empty set and hence $\mathcal{A}$ is invertible. Multiplying the first row of (18) with λ^T gives

$$0 = \lambda^T\Phi\lambda + \lambda^T P\mu = \lambda^T\Phi\lambda + (P^T\lambda)^T\mu$$

and since $P^T\lambda = \mathbf{0}^{\hat{m}}$ it follows that $\lambda^T\Phi\lambda = 0$. Moreover, since Φ is conditionally positive definite it must be that $\lambda = \mathbf{0}^n$. This gives $P\mu = \mathbf{0}^n$ and hence $\sum_{i=1}^{\hat{m}} \mu_i p_i(\mathbf{x}_k) = 0$ for $k = 1,2,\cdots,n$. Since $\sum_{i=1}^{\hat{m}} \mu_i p_i(\mathbf{x}) \in \Pi_{m-1}(\mathbb{R}^d)$, if requiring the set of points X to be unisolvent, then from the definition of unisolvence it follows that $\mu = \mathbf{0}^{\hat{m}}$.

From this a theorem can be constructed which has already been proven above:

Theorem (Unique solution for interpolation problem)
Suppose ϕ is conditionally positive definite of order m and X is $\Pi_{m-1}(\mathbb{R}^d)$ unisolvent. Then (16) is uniquely solvable.

Choosing radial basis functions
Up until now the focus has been on the requirements on the set of points X and on the radial function $\phi(r)$ that ensure a unique solution to the interpolation problem. Thus far it has not been shown which form the radial functions can have and in which space the corresponding polynomial lies. The radial basis function $\phi(r)$ must have a shape that provides good interpolation results while at the same time be conditionally positive definite to ensure a unique solution to the interpolation problem. Common choices of ϕ are shown in Table (1) and the polynomial degrees are justified below.

Table 1: Different choices of radial basis functions

RBF	$\phi(r) > 0$	$p(\mathbf{x})$	$\mathrm{m}_0 = dim(\mathrm{p(x)})$
Cubic	r^3	$\hat{\mathbf{a}}^T\mathbf{x} + \hat{b}$	1
Thin plate spline	$r^2 \log r$	$\hat{\mathbf{a}}^T\mathbf{x} + \hat{b}$	1
Linear	r	$\hat{b}$	0
Multiquadratic	$\sqrt{r^2 + \gamma^2},\ \gamma > 0$	$\hat{b}$	0
Gaussian	$\exp(-\gamma r^2),\ \gamma > 0$	0	-1

Recalling the definition of $\mathcal{V}_m$ from (13) we formally denote $\mathcal{V}_0 = \mathbb{R}^n$ and hence $\Pi_{-1} = \{0\}$. Obviously $\mathcal{V}_{m+1} \subset \mathcal{V}_m$ for all $m \geq 0$. Powell [Pow92] shows that in the cubic and thin plate spline cases

$$\mathbf{c}^T\Phi\mathbf{c} > 0 \quad \forall \mathbf{c} \in \mathcal{V}_2 \backslash \{0\},$$

in the linear and multiquadratic cases

$$\mathbf{c}^T\Phi\mathbf{c} < 0 \quad \forall \mathbf{c} \in \mathcal{V}_1 \backslash \{0\},$$

and in the Gaussian case

$$\mathbf{c}^T \Phi \mathbf{c} > 0 \quad \forall \mathbf{c} \in \mathbb{R}^n \backslash \{0\}.$$

Define m_0 to be 1 in the cubic and thin plate spline case, 0 in the linear and multiquadratic cases and -1 in the Gaussian case. After choosing a form of ϕ, let m be an integer such that $m > m_0$, then ϕ is conditionally positive definite of order m and hence (14) is uniquely solvable as long as X is unisolvent. Table 1 shows a compilation of common radial functions with the corresponding minimal polynomial order required to ensure conditional positive definiteness. With a unique solution to (16) the surrogate is uniquely determined and it has the form

$$S(\mathbf{x}) = \sum_{j=1}^{n} \lambda_j \phi(||\mathbf{x} - \mathbf{x}_j||) + \sum_{k=1}^{\hat{m}} \mu_k p_k(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d. \tag{19}$$

2.1.3 Kriging method

Suppose the aim is to make a prediction about the value $f(\bar{\mathbf{x}})$ for some point $\bar{\mathbf{x}}$ in the domain. Before sampling any points there will be an uncertainty about the value of the function at this point. This uncertainty is modeled by saying that the value of the function at $\bar{\mathbf{x}}$ is given by a normally distributed random variable $Y(\bar{\mathbf{x}})$ with mean μ and variance σ^2. The correlations between the random variables for points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ are given by

$$Corr[Y(\mathbf{x}_i), Y(\mathbf{x}_j)] = \exp\left(-\sum_{l=1}^{d} \theta_l |x_{il} - x_{jl}|^{p_l} \right), \tag{20}$$

where $\mathbf{x}_k = (x_{k1}, x_{k2}, \ldots, x_{kd})$ and it is assumed that $\theta_l \geq 0$ and $0 < p_l \leq 2$. Large values of θ_l serve to model functions that are highly active in the l^{th} variable and p_l determines the smoothness of the function in the l^{th} direction.

The uncertainty about the function's values at n points can be represented as $\mathbf{Y} = (Y_1(\mathbf{x}_1), \ldots, Y_n(\mathbf{x}_n))^T$. The distribution of $\mathbf{Y}$ depending on $\mu, \sigma, p_l, \theta_l, l = 1,2,\ldots,d$ will characterize how the function is expected to vary when moving in different directions. To estimate these parameters, they are chosen to maximize the likelihood of the observed data according to the likelihood function

$$\frac{1}{(2\pi)^{n/2}\sigma^n \|\mathbf{R}\|^{1/2}} \exp\left(\frac{(\mathbf{f} - \mathbf{1}\mu)^T \mathbf{R}^{-1} (\mathbf{f} - \mathbf{1}\mu)}{2\sigma^2} \right), \tag{21}$$

where $\mathbf{f} = (f_1, \ldots, f_n)^T$ is the vector of function values and the i,j-th element of $\mathbf{R}$ is given by (20). In practice it is the logarithm of (21) that is used to compute the parameters together with equations for $\hat{\mu}$ and $\hat{\sigma}^2$ [Jon01]. Take a point $\mathbf{x}^*$ with some guessed function value f^* and make it the $n + 1^{th}$ point. Further, calculate the parameters by maximizing the likelihood function. These parameters reflect how the function varies as described earlier. Now, an augmented log-likelihood function can be derived [Jon01] describing how consistent the point ($\mathbf{x}^*$,f^*) is with the observed variation. It is therefore rather intuitive that the prediction for f^*, the so called Kriging predictor, is given by maximizing that augmented log-likelihood function. The Kriging predictor has the following form

$$\hat{f}(\mathbf{x}^*) = \hat{\mu} + \mathbf{r}\mathbf{R}^{-1}(\mathbf{f} - \mathbf{1}\hat{\mu}) \tag{22}$$

where

$$\hat{\mu} = \frac{\mathbf{1}^T \mathbf{R}^{-1} \mathbf{f}}{\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1}}, \quad \mathbf{r} = \begin{pmatrix} Corr[Y(\mathbf{x}^*), Y(\mathbf{x}_1)] \\ \vdots \\ Corr[Y(\mathbf{x}^*), Y(\mathbf{x}_n)] \end{pmatrix} \tag{23}$$

and where $\mathbf{R}$ and $\mathbf{f}$ are as given earlier.

For $\mathbf{x} = (\chi_1, \chi_2, \ldots, \chi_d)$, calling $\varphi(\mathbf{x}) = \exp\left(-\sum_{l=1}^{d} \theta_l |\chi_l|^{p_l}\right)$ then the i^{th} element of $\mathbf{r}$ is just $\varphi(\mathbf{x}^* - \mathbf{x}_i)$ and denoting the i^{th} element of $\mathbf{R}^{-1}(\mathbf{f} - \mathbf{1}\hat{\mu})$ by λ_i and $\hat{\mu} = \hat{b}$ then it holds that

$$\hat{f}(\mathbf{x}^*) = \hat{b} + \sum_{i=1}^{n} \lambda_i \varphi(\mathbf{x}^* - \mathbf{x}_i), \tag{24}$$

which is a form similar to the form of the RBF surrogate in (14). However $\varphi(\mathbf{u}) \neq \varphi(\|\mathbf{u}\|)$ for any norm $\|\cdot\|$ unless $p_l = 1\ \forall l = 1,2\ldots,d$ (Null($\|\mathbf{u}\|$)=$\{0\}$ since $\theta_l \geq 0$ but subadditivity and positive homogeneity only holds simultaneously if $p_l = 1\ \forall l = 1,2\ldots,d$). Therefore $\varphi(\cdot)$ is not a radial basis function, but it has a simliar form where $\varphi(\cdot)$ depends on the estimated parameters, found as described earlier.

2.2 Merit functions

Merit functions use the interpolated surrogate function to find promising areas of the design space to evaluate. The merit functions are not expensive to evaluate which is an advantage to the costly black-box function.

Many forms have been suggested for merit functions and the main qualifications required are that they guide the search towards unexplored areas of the search domain and/or towards promising areas with low function values. A merit function that is purely global would choose the point which is the furthest away from any other evaluated point, hence a space filling search. This would not in general give an accurate value of the minima. The extreme on the other end is always finding the minimum of the surrogate S_{min}. The surrogate approximates the costly black-box function hence there is a build-in uncertainty that has to be considered when finding new points. Therefore, always sampling the minima will render a purely local search and it is therefore easy so get stuck in a local minima.

The key to success is combining these two sometimes contradictory requirements to explore regions of the domain that have not yet been sampled and regions that have promising function values. Here, three forms of merit functions are presented: Gutmann's bumpiness minimization, the CORS method as well as the Quality function where the emphasis for this thesis is placed in the latter.

2.2.1 One stage/two stage methods

According to Jones [Jon01] surrogate based method for solving expensive global optimization problems can be classified as either one-stage or two-stage, where most methods are two-stage. In the first stage a surrogate model is fitted to the data, estimating the required coefficients. Then in the second stage these coefficients are considered "true" and the surrogate surface is used to compute new points. The potential pitfall with two-stage methods is that with an initial sample the error in the surrogate could be large. Therefore, considering the model "true" may lead to erroneous results such as searching too locally or stopping prematurely.

For one-stage methods the initial step of surrogate fitting is skipped and the two stages are "merged" in some sense. The response surface mathematics is used to evaluate a hypothesis about the location of the optimum. For example, the credibility of the hypothesis that the model passes through a point $\mathbf{x}^*$ with function value (target value) f^* can be assessed through observing the properties of the best-fitting response surface. Intuitively, the response surface that gives the smoothest surface may be seen as the most probable although it depends on the applications.

The first of the tree merit functions presented below is a one-stage method where the role of the target value f^* can be observed.

2.2.2 Target values - minimize bumpiness

The notion of bumpiness is motivated by the discussion around the definition of $\mathcal{B}(S)$ given in (9). There it was noted that minimizing $\mathcal{B}(S)$ will minimize the overall curvature of the function in some sense. To be precise, $S''(\mathbf{x})$ is a measure of the curvature and expression $\mathcal{B}(S)$ can be seen as a semi-norm and a semi-inner product of the curvature of $S(\mathbf{x})$,

$$\mathcal{B}(S(\mathbf{x})) = \int_{\mathbb{R}} [S''(\mathbf{x})]^2 dx = \langle S''(\mathbf{x}), S''(\mathbf{x}) \rangle = \|S''(\mathbf{x})\|.$$

A semi-norm $\|\cdot\|$ satisfies all the properties of a norm, except that $\|\mathbf{u}\| = 0$ does not imply $\mathbf{u} = 0$. It can be shown [Gut01] that the radial function S for a given ϕ which satisfies (14) also minimizes the semi-norm $\langle g,g \rangle^{1/2}$ on the set of all functions g of the form (19) for a given ϕ that satisfies $g(\mathbf{x}_k) = f_k,\ \ k = 1,2,\ldots,n$.

Given a set of sampled points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and an estimated target value f^* we want to find the point $\mathbf{x}^*$ which most probably has the value f^*. For every $\bar{\mathbf{x}} \notin \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ an RBF $S(\mathbf{x})$ can be created that satisfies the interpolation conditions

$$\begin{array}{rl} S(\mathbf{x}_k) & = f(\mathbf{x}_k), \quad k = 1,2,\ldots,n, \\ S(\bar{\mathbf{x}}) & = f^*. \end{array} \quad . \tag{25}$$

The new point $\mathbf{x}^*$ is chosen as the value of $\bar{\mathbf{x}}$ whose surrogate model minimizes the measure of bumpiness $\mathcal{B}(S(\bar{\mathbf{x}}))$. A target value f^* should always be used that is lower than the minimum of the surrogate model S_{min}. If the difference $S_{min} - f^*$ is large then the algorithm aims for a big improvement hence a global search whereas for a low value a modest improvement will do, rendering a local search.

Gutmann [Gut99] shows that minimizing $\mathcal{B}(S(\mathbf{x}))$ subjected to the interpolation conditions (25) is equivalent to minimizing the utility function $g_n(\mathbf{x})$ defined as

$$g_n(\mathbf{x}) = (-1)^{m_0+1} \mu_n(\mathbf{x}) [S(\mathbf{x}) - f^*]^2, \quad \mathbf{x} \in \Omega \backslash X; \tag{26}$$

hence $\mathbf{x}^* = \arg\min_{\mathbf{x} \in \Omega \backslash X} g_n(\mathbf{x})$. m_0 is described earlier as the lowest polynomial order required for conditional positive definiteness and $\mu_n(\mathbf{x}^*)$ is the coefficient corresponding to $\mathbf{x}^*$ of the radial basis interpolation function solution L (Lagrangian function) which satisfies $L(\mathbf{x}_k) = 0,\ k = 1,\ldots,n$, and $L(\mathbf{x}^*) = 1$. More details can be found in [Gut99, Gut01] by Gutmann.

2.2.3 CORS method

In [RS05] Shoemaker and Regis introduce the CORS method, *Constrained Optimization using Response Surfaces* to deal with expensive black box optimization. The CORS method aims to find the minimum of a surrogate function given that you are a certain distance away from previously evaluated points. Say that $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ are previously evaluated points, then the maximum distance from any $\tilde{\mathbf{x}} \in \Omega$ to any point in X is

$$\Delta_i = \max_{\tilde{\mathbf{x}} \in \Omega} \min_{\mathbf{x}_j \in X} \|\tilde{\mathbf{x}} - \mathbf{x}_j\|.$$

Denoting the surrogate function in the i^{th} iteration by $S^i(\mathbf{x})$ the problem can be formulated as follows:

$$\begin{array}{ll} \min & S^i(\mathbf{x}) \\ \text{s.t.} & \|\mathbf{x} - \mathbf{x}_j\| \geq \beta_i \Delta_i, \ \ \forall \mathbf{x}_j \in X \\ & \mathbf{x} \in \Omega \end{array} \tag{27}$$

where β_i is a parameter $0 \leq \beta_i \leq 1$. If $\beta_i = 1$ the maximin point is found and hence a global search is conducted whereas with $\beta_i = 0$ the minimum of $S^i(\mathbf{x})$ is found which constitutes local search. The parameter β_i is cycled through to balance global and local search. Problem (27) is generally non-convex but the surrogate and constraints as well as their gradients are cheap to evaluate and hence gradient-based optimization solvers can be used. Another option is to run a global optimization method such as Constrained DIRECT (cf. Section 2.4) and refine by starting a non-linear program solver from that point.

2.2.4 Quality function

As before, consider a set of points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ sampled from the search domain Ω. In [JPRW09] a merit function of the following form is suggested

$$Q(\mathbf{y}) = \int_{\Omega} \left(U_X(\mathbf{x}) - U_{X \cup \mathbf{y}}(\mathbf{x})\right) \omega(S(\mathbf{x})) d\mathbf{x}, \tag{28}$$

where

$$U_X(\mathbf{x}) = \min_{\mathbf{x}_i \in X} \|\mathbf{x}_i - \mathbf{x}\| \tag{29}$$

is the space filling part, favoring parts of the domain not yet sampled and $\omega(S(\mathbf{x}))$ is a weight function favoring parts with low surrogate values.

The merit function is called the Quality function and is a two-stage method where the weight function can be altered to fit the requirements; hence it is potentially very flexible. The downside is the integration which requires a substantial amount of computational power when going to higher dimensions. However, the integral is justified by making it less favorable to be close to the boundary of Ω. This is good since more information is given when creating the surrogate if the points are not at the boundary of Ω but a certain distance away from it. In simple terms, this non-attraction to the boundary can be explained by the fact that the integration causes the fitness of a point to depend not only on the features of the points itself but also on the features of the surrounding points. The contribution to the integral from the spatial part is largest at the point $\mathbf{y}$ and then decreasing with increasing distance to $\mathbf{y}$. In integrating around a point that lies on the boundary of Ω, the neighborhood that lies outside of Ω will not contribute to the integral hence making it a less favorable point when comparing with inner points of the domain.

2.3 Experimental design

All surrogate based algorithms need an initial set of points in order to get started. To build the first interpolation surface, $n \geq d+1$ points are needed where d is the dimension of the search space $\Omega \subset \mathbb{R}^d$. The procedure of selecting initial points is often referred to as Experimental Design and there are many different approaches. Here the methods of Random sampling, Maximin and Latin Hypercube are discussed where the focus is on the latter.

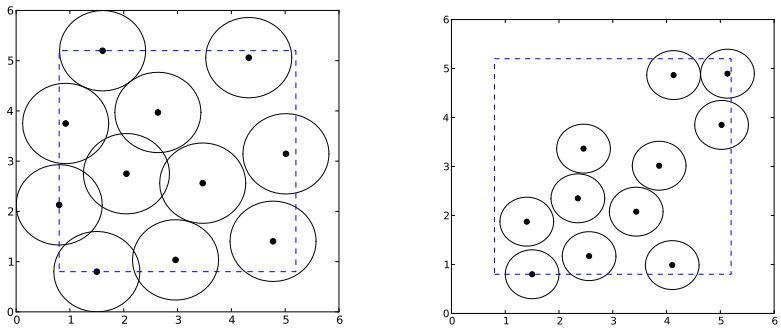
In designing an experiment, a model is fitted to data given by evaluating the experiment at a limited set of points X sampled from a domain Ω. One wishes to fit a model to the data that approximates the real event in all of Ω with as small error as possible. Therefore great care has to be taken when choosing the points X and it is important that, among other things, they are well spread out, ensuring that every portion of Ω is sampled.

In [SdHSV03] two main criteria are discussed that are used to measure the quality of the sampled points: *space-fillingness* and *non-collapsingness* where the first criterion is just what has been described earlier. For the second criterion, when a parameter (dimension) has no influence on the response of the output, two points that differ only in that dimension collapse, meaning that they can be seen as the same point evaluated twice. With a limited amount of evaluations, that should of course be avoided.

2.3.1 Random sampling

The simplest way of finding a set of n points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ in Ω is to randomly pick n points with a uniform probability distribution in Ω. However, this procedure cannot ensure that the points are well distributed in Ω so some method is needed to keep them apart.

One simple method for doing so is illustrated in Figure 1. Points are picked at random from Ω where a point is kept and added to X if it does not lie within any circle of radius R centered round points in X. If R or n are large enough then this may be impossible but then points are chosen that minimizes the overlap of the circles. In Figure 1(a) where $R = 0.8$ it is apparent that two points exist such that their circles overlap with other circles. In Figure 1(b) where $R = 0.5$ none of them overlap.



(a) Random sampling design with $R = 0.8$

(b) Random sampling design with $R = 0.5$

Figure 1: Illustration of a random sampling design where intersection of the circles centered at the points in X should be avoided.

This method gives a well spread out set of points if the radius R is estimated correctly where the radius R has to be estimated depending on the domain Ω and the number of points n. In generalizing to higher dimensions, an estimation of R requires the relation of the volume of Ω to the volume of the hyper-sphere. The volume is given by

$$V_d(R) = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}R^d = \begin{cases} \dfrac{\pi^{d/2}}{(\frac{d}{2})!}R^d, & d \text{ even}, \\ 2^{(d+1)/2}\dfrac{\pi^{(d-1)/2}}{d!!}R^d, & d \text{ odd}, \end{cases} \tag{30}$$

where $d!!$ is the double factorial, $d!! = \prod_{i;0\leq 2i<d}(d-2i)$. From (30) it is apparent that the coefficient depending on d vanishes as $d \to \infty$ due to the factorial denominator and hence with a fixed number of points then R must increase if d increases in order to ensure that the points are well spread out. This can be illustrated in Figure 1(b) where R is too small and hence the upper left corner is not covered. The estimation of R is the main drawback of this method.

2.3.2 Maximin

Suppose we want to sample n points in Ω, preferably with a good spatial distribution. This can be done by maximizing the minimum distance between the n points. This can be formulated as

finding a set of points X that is given by the following optimization problem

$$\max_{X\subset\Omega}\min_{0\leq i<j\leq n}\|\mathbf{x}_i-\mathbf{x}_j\| \iff \begin{cases} \max\limits_{\mathbf{x}} & d, \\ s.t. & d\leq\|\mathbf{x}_i-\mathbf{x}_j\|,\ \forall i,j:0\leq i<j\leq n, \\ & \mathbf{x}_j\in\Omega,\ j=1,2,\ldots,n. \end{cases}$$

Stinstra et.al. [SdHSV03] propose a method for solving this optimization problem by excluding one point at a time and solving a sequence of smaller problems in order to find a point to add. Also several points, preferably in proximity of one another, can be excluded simultaneously but then the optimization gets more complicated. The most commonly used norms are l^1, l^2 and l^∞ and the maximin problem for the l^2 norm can be illustrated in a similar way as in Figure 1. From the nature of the l^1 and l^∞ norms, boxes are used instead of circles when the distribution for those norms are illustrated as in Figure 1.

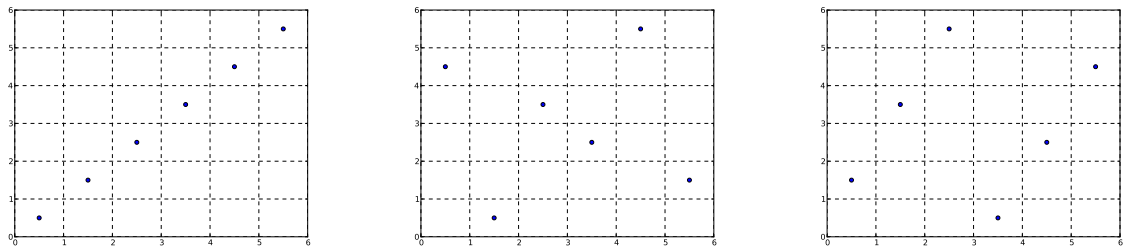
2.3.3 Latin hypercube design

The Latin Hypercube design was proposed by McKay et.al. [MBC79] and it can be viewed as a d-dimensional extension of the Latin square sampling ([Raj68, BHH87]). As before n points should be sampled from Ω in a "good" manner. We introduce the notion of stratified sampling, meaning that parts of the domain are sampled individually. In stratified sampling all areas of the space Ω are represented by input values. Ω is partitioned into I disjoint strata $\Omega^i,\ i=1,2,\ldots,I$ and random samples $z_{ij}, j=1,2,\ldots,n_i$ are obtained from every Ω^i. Then $\sum n_i = n$ and if $I=1$ then random sampling over all of Ω is obtained.

With the Latin Hypercube Design (LHD) this is taken even further ensuring that each dimension has all portions of its distribution represented by input values. The range of each dimension is divided into n parts each of equal marginal probability, denote this sample $X^k = \{z_{jk}, j=1,2,\ldots n\}$ for dimension k. Then for $k=1,2,\ldots,d$ a sample $z_{j_k k}$ is picked at random without replacement from X^k creating a point $\mathbf{x}_j = (z_{j_1 1}, z_{j_2 2}, \ldots, z_{j_d d})$. This is done for $j=1,2,\ldots,n$ creating $X=\{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_n\}$.

The LHD has the non-collapsingness property meaning that projecting an n-point design onto any factor, n different levels are given for that factor. If the output is dominated by only a few factors this method ensures that those components are represented in a stratified manner no matter what factors turn out to be important.

The LHD itself does not provide a set of well spread-out points as figure 2(a) shows. In this figure the correlation is $\rho=1$ and not only are the points not well spread out, they are also non-unisolvent. In figure 2(b) a set of points X is shown where the correlation is close to zero, $\rho=-0.03$ but the distance between the two points in the middle of the figure is not very large.



(a) A LHD in worst case scenario, $\rho=1$.

(b) A LHD with a small correlation, $\rho=-0.03$.

(c) A Maximin LHD, $\rho=0.2$.

Figure 2: The Latin Hypercube design with $n=6$ points and $\Omega=[0,6]\times[0,6]\subset\mathbb{R}^2$.

There are procedures for finding good LHDs by minimizing the pairwise correlations or by maximizing the inter-site distances. With a pure random sampling in all of Ω a small correlation says nothing about the distribution of X in Ω but it does for the LHD since it also uses stratification. Although ρ is small the smallest inter-site distance need not be very large as seen in Figure 2(b). In maximizing the inter-site distances the so-called maximin LHD [dHMHvD07] is obtained which combines the maximin problem discussed earlier with the features of the LHD. The maximin design in Figure 2(c) is taken from `www.spacefillingdesigns.nl`. The minimum distance is maximized although the correlation is not very small, $\rho = 0.2$.

From Figures 2(b) and 2(c) it seems that there could be a trade-off between minimizing correlation and maximizing inter-site distance or at least that the two are not always entirely correlated. That observation turns out to be correct. Roshan and Ying [VH08] showed that minimizing the pairwise correlations and maximizing the inter-site distances need not be in agreement and instead suggest a multi-objective optimization approach providing good results in terms of both the correlation and distance criteria.

Despite this, the method of just minimizing correlations provides good enough results, in particular if additional points are sampled later.

2.4 DIRECT algorithm

In order to find the global minimum or maximum of a function over a domain, a global Optimization solver is needed. The DIRECT (DIviding RECTangles) algorithm is such an algorithm that was first introduced in [PJS93] motivated by a modification to Lipschitzian optimization. It was created to solve difficult global optimization problems with bound constraints and a real-valued objective function. The Lipschitzian optimization methods are also global optimization methods although the largest drawback is that a Lipschitz constant has to be estimated which can be hard or even impossible if the function is not Lipschitz continuous. The DIRECT algorithm requires no knowledge of the Lipschitz constant or for the objective function to even be Lipschitz continuous. It is a sampling method meaning that the progress of the algorithm is governed only by evaluations of the objective function.

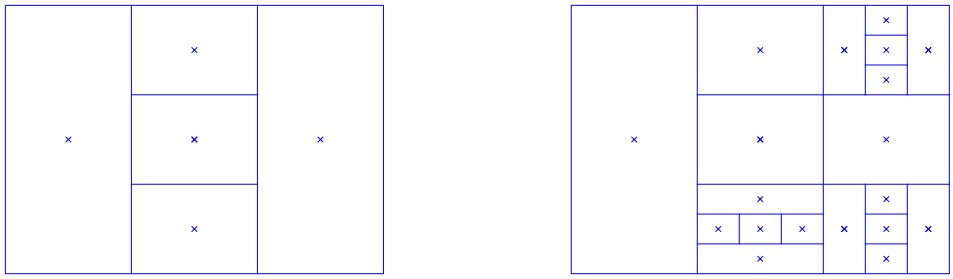
2.4.1 Initial step

DIRECT begins by transforming the domain to the unit hypercube $\hat{\Omega} = \{\mathbf{x} : \mathbf{x} \in [0,1]^d\}$ where the algorithm operates except for when evaluating the function. The center of the space is c_1 and the algorithm commences by finding $f(c_1)$. The function is then evaluated in $c_1 \pm \delta \mathbb{e}_i,\ i = 1,2,\ldots,d$ where δ is one third the side length in the cube and $\mathbb{e}_i$ is the unit vector in the i^{th} direction. During the initial step the DIRECT algorithm chooses the best function value in the largest space. Therefore the following is defined

$$w_i = \min\left(f(c_1 - \delta\mathbb{e}_i), f(c_1 + \delta\mathbb{e}_i)\right),\ 1 \leq i \leq d \tag{31}$$

and the dimension with the smallest w_i is divided into thirds so $c_1 \pm \delta\mathbb{e}_i$ are the centers of the new hyper-rectangles. This is done for all dimensions on the "center hyper-rectangle" continuing with the dimension which has the next smallest w_i. The initial division step is illustrated in Figure 3(a).

After the initial step, the algorithm begins its loop of identifying potentially optimal hyper-rectangles, sampling at their center and dividing them into thirds.



(a) First step of DIRECT.

(b) The divisions after a few iterations.

Figure 3: Illustration of the DIRECT division algorithm.

2.4.2 Potentially optimal hyper-rectangles

The algorithm searches globally and locally by dividing all hyper-rectangles that meet the criteria in (32). Basically, the hyper-rectangles are divided into groups of the same size and for each group the algorithm considers dividing the rectangle with the smallest value of f at the center. Not all such hyper-rectangles are divided though; also an estimation of the Lipschitz constant is used to decide whether division should take place. The argument used for DIRECT is hence basically a Pareto efficiency argument.

Definition (Potentially Optimal Hyper-rectangle) [Fin03]
Let $\epsilon > 0$ be a positive constant and let f_{min} be the current best function value. A hyper-rectangle j is then said to be potentially optimal if $\exists \hat{K} > 0$ such that

$$\begin{aligned} f(c_j) - \hat{K}d_j &\leq f(c_i) - \hat{K}d_i, \quad \forall i, \text{ and} \\ f(c_j) - \hat{K}d_j &\leq f_{min} - \epsilon|f_{min}|. \end{aligned} \tag{32}$$

Here c_j denotes the center of hyper-rectangle j and d_j is a measure of this hyper-rectangle. Jones et.al. [PJS93] chose the distance from c_j to its vertices as a measure whereas Kelley et.al. [GK01] used the longest side in the hyper-rectangle. The parameter ϵ is a balance parameter giving the user control over the relation between global and local search [FK06].

The potentially optimal hyper-rectangles found are then divided in the next iteration step. They are divided along the direction corresponding to the longest side of the hyper-rectangle ensuring that they will shrink in every dimension. If there is a tie for the longest side then all those sides are divided in a fashion similar to the initial step where the order of division is determined by (31) but centered around the center of the hyper-rectangle in question instead of c_1. In Figure 3(b) the division algorithm is illustrated as a pattern of division after a few iterations.

Generally, potentially optimal hyper-rectangles either have low function values at their centers or are large enough to be good targets for a global search, which Figure 4 illustrates. In the figure each point corresponds to a group of hyper-rectangles having equal size and equal function values. The hyper-rectangles illustrated by the lower convex hull in the figure all satisfy (32). It is worth noting the role of the parameter ϵ: the point $(0, f_{min} - \epsilon|f_{min}|)$ alters the convex hull so that points with low function values but in small hyper boxes are not necessarily potentially optimal. This shifts sampling somewhat towards larger hyper-rectangles and unexplored parts of the domain.

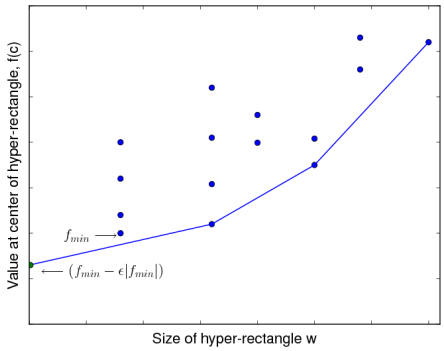


Figure 4: A graph showing grouping by hyper-rectangle size and function value where the potentially optimal hyperrectangles lie on the line.

2.4.3 Norms

In this thesis a few different norms are used. From now on the norm $\|\cdot\|$ denotes the 2-norm $\|\cdot\|_2$ where $\|\mathbf{x}-\mathbf{y}\|_2$ describes the Euclidean distance between points $\mathbf{x}$ and $\mathbf{y}$.

If different dimensions of the search space have different length scales or if it is of interest to rescale the units of the domain, then a weighted norm can be used. In particular it is advantageous to use a weighted norm if one wants the relative difference in each dimension to have equal weight in the norm. In this thesis the weighted norm for the domain Γ is denoted $\|\cdot\|_\Gamma$ and it is defined by

$$\|\mathbf{v}\|_\Gamma \equiv \|W(\mathbf{v}-\mathbf{v}^{min})\| = \left[\sum_{i=1}^{d}\left(\frac{v_i - v_i^{min}}{w_i}\right)^2\right]^{1/2}, \tag{33}$$

where

$$W_{ij} = \frac{1}{w_i}\delta_{ij}, \qquad w_i = v_i^{max} - v_i^{min} = \max_{v_i\in\Gamma} v_i - \min_{v_i\in\Gamma} v_i,$$

where δ_{ij} is the Kronecker delta function, hence $\delta_{ij}=0,\ i\neq j$ and $\delta_{ij}=1,\ i=j$. The vector given by $W(\mathbf{v}-\mathbf{v}^{min})$ takes values in the hyper-square $[0,1]^d$ and hence each dimension has equal impact on the norm. This norm is only defined for box-constrained domains such as Ω where $v_i^{min}=l_i^v$ and $v_i^{max}=u_i^v$ in accordance with previous notation. To be strict $\|\mathbf{v}\|_\Gamma$ is not a norm by itself since none of the required properties are satisfied, unless $\mathbf{v}^{min}=\mathbf{0}^d$. But $\|W(\mathbf{v}-\mathbf{v}^{min})\|$ is a norm and defining $\|\mathbf{v}\|_\Gamma \equiv \|W(\mathbf{v}-\mathbf{v}^{min})\|$ then $\|\mathbf{v}\|_\Gamma$ is a norm per definition.

The norm $\|\mathbf{v}\|_\Gamma$ has the same effect as using the inverse of the linear transformation $\mathcal{T}(v_i) = v_i(v_i^{max}-v_i^{min})+v_i^{min}$ on each element of $\mathbf{v}=(v_1,v_2,\ldots,v_d)$ and then use the Euclidean 2-norm.

2.5 Multiobjective optimization

In single objective optimization the search space is often well defined. As soon as there are several contradicting objectives to be optimized simultaneously, there is no longer a single optimal solution but rather a whole set of possible solutions of equal quality. Consider κ objective functions $f_i(\mathbf{x}),\ i=1,2,\ldots,\kappa$ and otherwise the same problem as in (2) but without integrality.

The problem can then be stated as finding

$$\begin{aligned} & \mathcal{P}^* = \arg\min F(\mathbf{x}) \equiv (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_\kappa(\mathbf{x})), \\ \text{subject to} \quad & \\ & l_i \leq g_i(\mathbf{x}) \leq u_i, \ \ i \in \mathcal{I} \\ & \mathbf{x} \in \Omega \end{aligned} \tag{34}$$

A solution to a multiobjective problem minimizes the components of a vector $F(\mathbf{x})$ in a Pareto sense, where $\mathbf{x} \in \Omega$ is the decision vector. In order to quantify this, the notions of Pareto optimality and Pareto dominance are introduced [CL05, vVL00].

Pareto Dominance
A vector $\mathbf{u} = (u_1, \ldots, u_\kappa)$ *is said to dominate* $\mathbf{v} = (v_1, \ldots, v_\kappa)$ *(denoted by* $\mathbf{u} \preceq \mathbf{v}$*) if and only if* $\mathbf{u}$ *is partially less than* $\mathbf{v}$ *i.e.,* $\forall i \in \{1, \ldots, \kappa\}$*,* $u_i \leq v_i$ *and* $\exists i \in \{1, \ldots, \kappa\}$*:* $u_i < v_i$*.*

Pareto Optimiality
A solution $\mathbf{x} \in \Omega$ *is said to be Pareto optimal with respect to* Ω *if and only if there is no* $\mathbf{x}' \in \Omega$ *for which* $\mathbf{v} = F(\mathbf{x}') = (f_1(\mathbf{x}'), f_2(\mathbf{x}'), \ldots, f_\kappa(\mathbf{x}'))$ *dominates* $\mathbf{u} = F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_\kappa(\mathbf{x}))$*.*

From these definitions the notion of a Pareto Optimal set can be introduced

$$\mathcal{P}^* = \{\mathbf{x} \in \Omega : \nexists \mathbf{x}' \in \Omega \ \text{ s.t. } \ F(\mathbf{x}') \preceq F(\mathbf{x})\}$$

which is the set of Pareto optimal points and the Pareto front

$$\mathcal{PF}^* = \{F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_\kappa(\mathbf{x})) : \mathbf{x} \in \mathcal{P}^*\}$$

which is the set of objective vectors corresponding to the Pareto optimal set. From these definitions it follows that $\mathcal{P}^* \subset \Omega$ and that $\mathcal{PF}^* \subset F(\Omega)$ is the range of $F(\mathbf{x})$ restricted to $\mathcal{P}^*$.

In solving problem (34) the aim is to find the Pareto optimal set $\mathcal{P}^*$. In determining which solution from the Pareto optimal set to use, it is necessary to look at the Pareto front and decide which trade-off between the objectives that is the best for the particular problem. There is hence no simple answer to which solution to choose. If there were, then either there would be no trade-off between the objectives and hence the Pareto optimal set is a singleton or it is known exactly how important each objective is.

In the latter case it may be advantageous to instead use a so called aggregated objective function. If it is known beforehand how big of an impact each objective should have, hence their relative importance has been ranked e.g. by weights: w_i for objective $f_i(\mathbf{x})$, then the multiobjective problem can be transformed to a single objective problem with objective function $\sum_{i=1}^{\kappa} = w_i f_i(\mathbf{x})$.

Variation of the parameters w_i can also be used to find Pareto optimal points, however the weighted sum approach can only capture convex Pareto frontiers [MMS00]. The sum can be modified to get around this as suggested in [MMS00] where for a two objective case the following aggregated form is used $f_1(\mathbf{x})^s + b f_2(\mathbf{x})^s$, with $b > 0$ and $s \in \mathbb{N}$.

3 The optimization algorithm

The black box optimization algorithm is based on surrogate modeling, described in Section 2.1, and the optimization of a merit function, described in Section 2.2. In this section, the different parts of the optimization algorithm are theoretically presented and described using the theory in the previous section. Several methods for surrogate modeling, merit functions and experimental designs have been discussed and here the methods best suited for the given requirements are chosen to create an algorithm. This Section describes the methods used whereas the practical implementation is described in Section 4.

3.1 Creating a surrogate model

In Section 2.1 it was shown that in choosing a conditionally positive definite function ϕ of order m and a unisolvent set of points $X \subset \Omega$, the interpolation problem (35) is uniquely solvable.

$$\begin{aligned} S(\mathbf{x}) &= \sum_{j=1}^{n} \lambda_j \phi(||\mathbf{x} - \mathbf{x}_j||_\Omega) + p(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d \\ S(\mathbf{x}_i) &= f_i, \quad i = 1,2,\ldots,n \end{aligned} \tag{35}$$

The norm $\|\cdot\|_\Omega$ is the weighted norm allowing each dimension to have equal relative impact. Further, choosing a form of the radial basis function from Table 1 e.g. $\phi(r) = r^3$ and $p(\mathbf{x}) = \hat{\mathbf{a}}^T\mathbf{x} + \hat{b}$ ($\mu = (\hat{\mathbf{a}}^T, \hat{b})^T$) it has been shown that ϕ is conditionally positive definite of order $m = 1$. Therefore the matrix equation

$$\begin{pmatrix} \Phi & P \\ P^T & \mathbf{0}^{\hat{m}\times\hat{m}} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0}^{\hat{m}} \end{pmatrix} \tag{36}$$

has a unique solution where $\Phi_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_\Omega^3$ and

$$P = \begin{pmatrix} \mathbf{x}_1^T & 1 \\ \mathbf{x}_2^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_n^T & 1 \end{pmatrix}, \quad \lambda = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{pmatrix}, \quad \mu = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{\hat{m}-1} \\ b \end{pmatrix}, \quad \mathbf{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}. \tag{37}$$

For the uniqueness to hold, the set of points X needs to be unisolvent. If X is $\Pi_m(\mathbb{R}^d)$ unisolvent then $X \cup \mathbf{x}_{n+1}$ is also unisolvent hence unisolvence has to be established for an initial set of point after which it holds for any addition of points to the set. In order for unisolvence to hold there must be at least $d + 1$ points where d is the dimension of the search space.

The surrogate function is then given by

$$S(\mathbf{x}) = \sum_{j=1}^{n} \lambda_j \phi(||\mathbf{x} - \mathbf{x}_j||_\Omega) + \mu^T(\mathbf{x}^T, 1)^T. \tag{38}$$

The RBF method and the Kriging methods are similar in performance but the Kriging method requires maximizing the likelihood function whereas the BRF only requires solving a system of linear equations. The simplicity of the RBF surrogate method was preferred in this case. In using a surrogate based model, the prediction model part of this thesis followed naturally from the parameter tuning part; requiring less code and making it easier to comprehend.

3.2 Choosing initial points

As described in Section 2.3.3 the Latin Hypercube design has several good properties where non-collapsingness is one and another is the fact that a low correlation will ensure that the set of points is rather well spread out, which is not the case for random sampling. Therefore the Latin Hypercube Design is used where minimizing the modulus of the correlation ensures a good enough spread out set of point.

Initially n_{init} points are chosen using the scheme described in Section 2.3.3. The result is a set of points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n_{init}}\}$ that is unisolvent with outmost certainty hence the interpolation can be done uniquely. With additional points drawn from Ω and added to X the initial configuration is not as important as it would have been if no additional points were chosen, although with a good initial set of points the surrogate will hopefully approximate the black box function in a better way.

3.3 Finding new points to evaluate

In creating a surrogate function S from a set of points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and corresponding function values $\{f_1, \ldots, f_n\}$, the real function $f(\mathbf{x})$ is modeled by $S(\mathbf{x})$. Since $S(\mathbf{x})$ is an interpolation of observed data, the surrogate function will only be an approximation of the real function. Increasing the number of data points renders a more trustable model, especially so if the data points are well spread out and with a more realistic model, the minima can be determined with higher accuracy. Although, only making the model more reliable will not provide the sought minima. Therefore we wish to weight in a point's space filling properties with its value's proximity to the model's minima. It would also be advantageous to easily be able to change the impact that the space filling and the proximity to the model's minima has when determining a new point to evaluate. The (black-box) output constraints also have to be incorporated into the method, especially considering the possibly large error in the estimation of the output constraints.

Given a surrogate function constructed using a set of points X and corresponding function values $\mathbf{f}$, one wishes to find new points to evaluate. This can be done using merit functions as described in Section 2.2. We wish to construct a merit function that has the properties mentioned above and that can easily be extended to the multiobjective case. Therefore the form of the merit function chosen is the Quality function $Q(\mathbf{y})$ suggested in [JPRW09] and briefly discussed in Section 2.2.4. It has almost all the desired properties discussed and can be modified to fit the rest as will be described in the following sections.

3.4 Quality function

In [JPRW09] a quality function of the following form is suggested:

$$Q(\mathbf{y}) = \int_{\Omega} \left(U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x})\right) \omega(S(\mathbf{x}))d\mathbf{x}, \tag{39}$$

where $Q(\mathbf{y})$ is maximized to find a new point to evaluate. Hence we seek $\mathbf{y}^*$ such that

$$\mathbf{y}^* = \arg\max_{\mathbf{y}\in\Omega} Q(\mathbf{y}). \tag{40}$$

This form of $Q(\mathbf{y})$ is in this thesis denoted the *neighborhood quality function* (NQF). The term $U_X(\mathbf{x})$ is defined as the shortest Euclidean distance from a point $\mathbf{x}$ to any of the points in the set X scaled to unit size:

$$U_X(\mathbf{x}) = \min_{\mathbf{x}_k\in X} \|\mathbf{x}_k - \mathbf{x}\|_{\Omega}. \tag{41}$$

Therefore the term $(U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x}))$ measures the prospective model uncertainty reduction achieved through adding a point $\mathbf{y}$ to X. In maximizing only this term, or setting $\omega(S(\mathbf{x})) = 1$, the new point will seek to create a well spread out set of points.

The surrogate interpolation as well as the uncertainty function $U_X(\mathbf{x})$ is depicted in Figure 5. Starting with the leftmost picture and going to the right, one additional point has been added to X in each figure. The filled line corresponds to the surrogate model passing through each data point. The dashed-dotted line shows the shape of the surrogate before the point was added, hence it corresponds to the surrogate in the figure to its left. At the bottom the uncertainty function $U_X(\mathbf{x})$ is plotted and it can be seen that where the new points are added, the uncertainty is reduced. Note that the uncertainty function has been rescaled and should only be compared relative to itself. The horizontal axis shows the x-values and the vertical axis shows the function values.

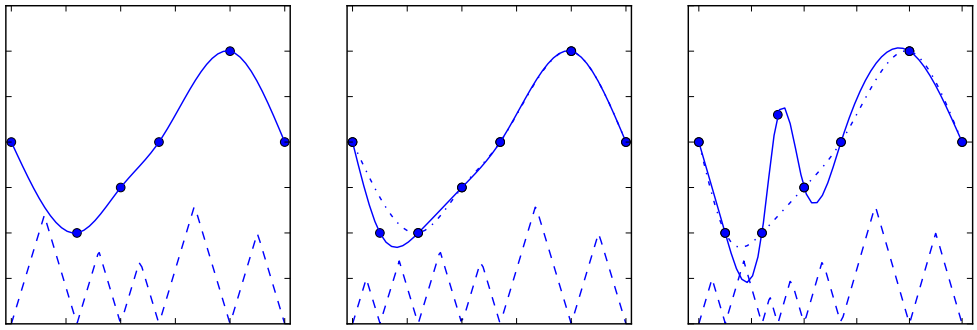


Figure 5: Interpolation of data using the surrogate model. Going from the left to the right, one additional point has been added in each figure. The dash-dotted line shows the surrogate without the new point and the dashed line at the bottom shows the uncertainty function $U_X(\mathbf{x})$ which has been rescaled in these figures.

The term $\omega(S(\mathbf{x}))$ depends on the surrogate model and it is call the weight function for the integral. It weights the space filling properties of a point $\mathbf{y}$ as described before with the proximity to the surrogate's minimum value.

Another form of the weight function investigated in this thesis is a modification of (39) which has been named the *non-neighborhood quality function* (NNQF). It has the following form:

$$Q(\mathbf{y}) = U_X(\mathbf{y})\omega(S(\mathbf{y})) \tag{42}$$

and it is hence less computationally heavy. The two versions of the quality function are named in this way, in this thesis, since the quality of a point $\mathbf{y}$ for NQF depends not only on the point itself but also on a neighborhood of that point. The NNQF depends only on the point in which it is evaluated (as well as X) and it is therefore more prone to select points close to the boundary of the set Ω as discussed in Section 2.2.4.

3.5 Weight function

The weight function $Q(\mathbf{y})$ is maximized and hence the weight function $\omega(S(\mathbf{x}))$ should be comparably small for points where in some sense it is less favorable to be. In which sense depends on the number of objective functions, hence the weight function has different expressions for single objective and multiobjective optimization.

3.5.1 Single objective

With one objective, the aim is to find as small value as possible for the black box function, $f(\mathbf{x})$. Therefore in maximizing $Q(\mathbf{y})$ in (39) or in (42), the weight function $\omega(S(\mathbf{x}))$ should have relatively low values where the surrogate's values are far away from the surrogates minima. The weight function's relative spatial dependence on $S(\mathbf{x})$, given by $U_X(\mathbf{x})$, should be tunable to account for the uncertainty in the surrogate model and hence the following form of ω is used, suggested in [JPRW09]

$$\omega(S(\mathbf{x})) = \exp\left(-\sigma \frac{S(\mathbf{x}) - S_{min}}{S_{max} - S_{min}}\right). \tag{43}$$

The parameter $\sigma \geq 0$ is the tuning parameter that accounts for the relation between spatial dependence and the proximity to the surrogate's minima and S_{min} and S_{max} are the minimum and maximum of the surrogate model respectively. From this it is easy to see that the maximum value is $\omega(S(\mathbf{x})) = 1$ for $S(\mathbf{x}) = S_{min}$ and the minimum value is $\omega(S(\mathbf{x})) = e^{-\sigma}$ for $S(\mathbf{x}) = S_{max}$. In this way the weight function is kept bounded at all times and the bounds are dependent only on the parameter σ. So with $\sigma = 0$ then $\omega(S(\mathbf{x})) = 1$ independent of the value of the surrogate, meaning a space filling search whereas increasing σ will favor regions where the surrogate is close to S_{min}.

3.5.2 Multiobjective

For multiobjective optimization with κ objectives, one surrogate model is created for each objective creating the vector of surrogates $\mathbf{S}(\mathbf{x}) = (S_1(\mathbf{x}), S_2(\mathbf{x}), \ldots, S_\kappa(\mathbf{x}))$ where $S_i, i = 1,2,\ldots,\kappa$ is determined by solving (16) with $\mathbf{f}$ given by the values for the i^{th} objective.

The aim of multiobjective optimization is to find the Pareto front as described in Section 2.5. Therefore the weight function should favor points close to the Pareto front and hence the Euclidean distance from the surrogate's value at a point to the Pareto front, scaled to unit size, is used as a measure of the proximity to the Pareto front. The set of points constituting the Pareto front of $\mathbf{S}(\mathbf{x})$ as defined in Section 2.5 is denoted $\mathbf{S}^*$ and the distance (naturally the minimal distance) is given as

$$\text{dist}(\mathbf{S}(\mathbf{x})) = \min_{\mathbf{s}^* \in \mathbf{S}^*} \|\mathbf{S}(\mathbf{x}) - \mathbf{s}^*\|_{\mathbf{S}(\Omega)} \tag{44}$$

where the scaled norm is used to make every dimension equally important. In analogy with the single objective case, the weight function has the following form

$$\omega(\mathbf{S}(\mathbf{x})) = \exp\left(-\sigma \frac{\text{dist}(\mathbf{S}(\mathbf{x}))}{\text{dist}(\mathbf{S}(\tilde{\mathbf{x}}))}\right) \tag{45}$$

where $\tilde{\mathbf{x}}$ is given by

$$\tilde{\mathbf{x}} = \arg\max_{\mathbf{x} \in \Omega} \text{dist}(\mathbf{S}(\mathbf{x})) \tag{46}$$

hence $\text{dist}(\mathbf{S}(\tilde{\mathbf{x}}))$ is the maximum distance to the Pareto front in the set $\mathbf{S}(\Omega)$. As in the single objective case the following holds $e^{-\sigma} \leq \omega(\mathbf{S}(\mathbf{x})) \leq 1$ where the upper bound is reached when $\mathbf{S}(\mathbf{x})$ is at the Pareto front and ω is decreasing with increasing distance to the Pareto front, reaching the lower bound when $\mathbf{x} = \tilde{\mathbf{x}}$.

3.6 Monte Carlo integration over Ω

The expression for the quality function as given in (39) involves integration over the set Ω. There are many methods of integration available where e.g. the trapezoidal quadrature rule is

simple, easy to use and provides fairly good results when the integrand is flat enough. However, increasing the dimension of $\Omega \subset \mathbb{R}^d$ will render trapezoidal integration useless since the number of points increase exponentially with the dimension.

Another method that can be extended to higher dimensions without an exponential increase in calculations is the Monte Carlo integration method. Monte Carlo is a class of computational methods relying on randomly sampled numbers. For Monte Carlo integration when increasing the number of samples the law of large numbers will ensure convergence if the integrand is "nice" enough [SdM01].

Denote the integrand in (39) by $I(\mathbf{x},\mathbf{y})$: $I(\mathbf{x},\mathbf{y}) = (U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x}))\,\omega(S(\mathbf{x}))$. Choose randomly a set of points E from an even distribution in Ω, then

$$\int_\Omega (U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x}))\,\omega(S(\mathbf{x}))d\mathbf{x} \approx V(\Omega)\frac{1}{|E|}\sum_{\mathbf{x}\in E} I(\mathbf{x},\mathbf{y}) \tag{47}$$

where $V(\Omega)$ is the volume of Ω and as $|E| \to \infty$ the error in (47) goes to zero, in fact the error decreases as $1/\sqrt{|E|}$ [SdM01]. Therefore the maximization problem can be reformulated as

$$\mathbf{y}^* = \arg\max_{\mathbf{y}\in\Omega} Q(\mathbf{y}) \equiv V(\Omega)\frac{1}{|E|}\sum_{\mathbf{x}\in E\subset\Omega} I(\mathbf{x},\mathbf{y}). \tag{48}$$

3.6.1 Reducing the domain of integration

In $I(\mathbf{x},\mathbf{y})$ the spatial term $(U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x}))$ describes the prospective gain in space filling from adding a point $\mathbf{y}$ to the set X. Consider a fixed $\mathbf{y} \in \Omega$ and a given set of points $X \subset \Omega$. The term $U_X(\mathbf{x})$ is the minimum distance from $\mathbf{x}$ to points in X hence when adding $\mathbf{y}$ to X the following will hold $U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x}) \neq 0$ only for the points $\bar{\mathbf{x}} \in \Omega$ that has $\mathbf{y} = \arg\min_{\mathbf{x}\in X\cup\mathbf{y}} \|\bar{\mathbf{x}} - \mathbf{x}\|$ hence the points that are closer to $\mathbf{y}$ than any point in X. With this in mind it is clear that the domain of integration Ω can be reduced and this is illustrated in Figure 6.

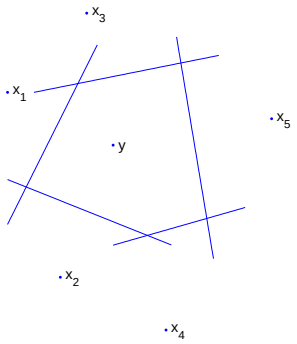


Figure 6: A polyhedron created by determining the region where $(U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x}))$ is non-zero in the bounded domain Ω.

The region where $I(\mathbf{x},\mathbf{y})$ is non-zero is a bounded polyhedron in $\Omega \subset \mathbb{R}^d$ and hence it can be described by a system of linear equations of the form $D\mathbf{x} \leq \mathbf{e}$ where $\mathbf{x} \in \Omega$ is bounded. The polyhedron is not necessarily bounded if Ω is not bounded. The polyhedron depends on $\mathbf{y}$ and it is here denoted $\Omega_y = \{\mathbf{x} \in \Omega : D\mathbf{x} \leq \mathbf{e}\}$. Given a set of points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n\}$ the matrix

D and vector $\mathbf{e}$ are explicitly written as [RW07]:

$$\begin{aligned} D_{\cdot k} &= (\mathbf{x}_k - \mathbf{y})^T \\ e_k &= \tfrac{1}{2}(\mathbf{x}_k - \mathbf{y})^T(\mathbf{x}_k + \mathbf{y}). \end{aligned} \tag{49}$$

The quality function has hence been reduced to

$$Q(\mathbf{y}) = \int_{\Omega_y} \left(U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x})\right) \omega(S(\mathbf{x}))d\mathbf{x} \tag{50}$$

and the expression for $Q(\mathbf{y})$ after Monte Carlo integration is as in (48) with Ω replaced by Ω_y.

3.7 Output constraints

Consider the case where there are several output functions among which there are one or more objectives. Besides the objectives there are output functions that there is no interest in minimizing or maximizing but only keeping within some bounds, so called output constraints. This can be stated as the following minimization problem:

Given an objective function $f(\mathbf{x})$ and η output functions $g_i(\mathbf{x}), i \in \mathcal{I}$, find $\mathbf{x}^*$ such that

$$\begin{aligned} \mathbf{x}^* = \quad & \arg\min f(\mathbf{x}) \\ s.t. \quad & l_i \leq g_i(\mathbf{x}) \leq u_i, \;\; i \in \mathcal{I} \\ & \mathbf{x} \in \Omega \end{aligned} \tag{51}$$

In incorporating this into the model, the quality function $Q(\mathbf{y})$ must be modified in some way. The function $f(\mathbf{x})$ is an unknown black box functions and hence surrogate modeling as described in Section 3.1 can be applied. The same is true for the functions $g_i(\mathbf{x}), i \in \mathcal{I}$.

For every output constraint $g_i(\mathbf{x})$ a surrogate model $S_{g_i}(\mathbf{x})$ can be created that approximates the real function. Denote the vector of surrogates by $\mathbf{S}_g(\mathbf{x}) = (S_{g_1}(\mathbf{x}), \ldots, S_{g_\eta}(\mathbf{x}))$. In the algorithm it should be disadvantageous to select points where the output constraints are outside the bounds. This functionality is placed in the function $\varphi(\mathbf{S}_g(\mathbf{y}))$. A simple form like the indicator function $1_{\mathcal{D}}(\mathbf{S}_g(\mathbf{x}))$ where $\mathcal{D}$ is as given in (5) is not favorable since the uncertainty in $\mathbf{S}_g(\mathbf{x})$ might exclude feasible points.

It would be favorable to be able to alter the function $\varphi(\mathbf{S}_g(\mathbf{y}))$ depending on the level of trust that can be put in the model. This leads us to consider a form similar to the weight function $\omega(S(\mathbf{x}))$ namely

$$\varphi(\mathbf{S}_g(\mathbf{y})) = \prod_{i=1}^{\eta} \min\left\{1, e^{-\sigma_c(S_{g_i}(\mathbf{y})-u_i)}\right\} \min\left\{1, e^{-\sigma_c(l_i - S_{g_i}(\mathbf{y}))}\right\} \tag{52}$$

where σ_c has a similar functionality as σ has in the weight function $\omega(S(x))$. The function's dependence on $\mathbf{S}_g(\mathbf{y})$ and σ_c can be seen in Figure 7.

With the addition of the output constraints function $\varphi(\mathbf{S}_g(\mathbf{y}))$ the quality function is given as follows

$$Q(\mathbf{y}) = \varphi(\mathbf{S}_g(\mathbf{y})) \int_{\Omega} \left(U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x})\right) \omega(S(\mathbf{x}))d\mathbf{x}. \tag{53}$$

where Ω and Ω_y are interchangeable.

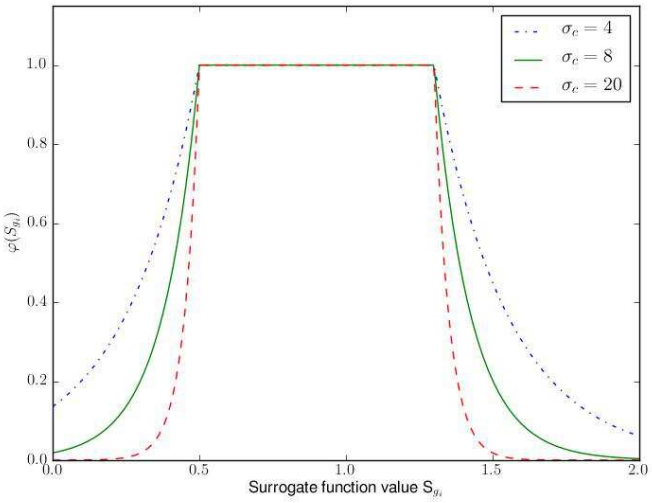


Figure 7: The function φ as described in (52) as a function of S_{g_1} (seen in one dimension), with $l_1 = 0.5$ and $u_1 = 1.3$.

3.8 Integrality

Recall from Section 1.4 the definition of $\mathbb{I}$: the set of indices for which the parameters are integer valued. Including this into the model gives a mixed integer non-linear problem (MINLP) which is NP-hard in general and in fact MINLPs are worse than NP-hard [Let09].

The simplest way by which integrality can be ensured is rounding to nearest feasible integer. This may provide results that are far from optimal, especially if the function in question varies considerably. There are methods for MINLPs which are hard to apply to this case. Therefore rounding together with an integer weight function $\gamma(\mathbf{y})$ is considered.

As with the output constraints function φ, the simple choice $\gamma(\mathbf{y}) = 1_G(\mathbf{y})$ where $G = \{\mathbf{v} : v_j \in \mathbb{Z}, j \in \mathbb{I}\}$ will not work in practice. In order to guide the algorithm toward integer values in the dimensions given by $\mathbb{I}$ the following function can be used

$$\gamma(\mathbf{y}) = \prod_{j \in \mathbb{I}} \min\left\{1, e^{-\sigma_{int}(\Delta_j(\mathbf{y}) - \delta)}\right\}$$

where (54)

$$\Delta_j(\mathbf{y}) = \min_{\tilde{y} \in \mathbb{Z}} |\tilde{y} - y_j| = \min\{\lceil y_j \rceil - y_j, y_j - \lfloor y_j \rfloor\}$$

and $\mathbf{y} = (y_1, y_2, \ldots, y_d)$. The function is depicted in Figure 8 where the steepness of the function is determined by σ_{int} in analogy with σ_c and σ and the constant δ determines the size of the neighborhood around every integer where $\gamma(\mathbf{y}) = 1$. The final forms of the quality functions are then given by

$$\begin{aligned} Q(\mathbf{y}) &= \gamma(\mathbf{y})\varphi(\mathbf{S}_g(\mathbf{y})) \int_\Omega \left(U_X(\mathbf{x}) - U_{X \cup \mathbf{y}}(\mathbf{x})\right) \omega(S(\mathbf{x}))d\mathbf{x} \\ \text{and} \\ Q(\mathbf{y}) &= \gamma(\mathbf{y})\varphi(\mathbf{S}_g(\mathbf{y}))U_X(\mathbf{y})\omega(S(\mathbf{y})) \end{aligned} \tag{55}$$

where Ω can be replaced by Ω_y as described earlier.

The functions φ and γ are functions of $\mathbf{y}$ only and hence are not integrated over. The integral is used to account for not only the characteristics at the point $\mathbf{y}$ but also a neighborhood Ω_y. If

the integrand would be multiplied with $\varphi(\mathbf{S}(\mathbf{x}))$ then points close to the boundary of the output constraints would be picked with less likelihood. This, since then characteristics of φ will most likely make the integrand small in a part of Ω_y leading to a lower value of the quality function and hence making it a less favorable point. This behavior is not advantageous since optimum is in general likely to occur where one or more of the constraints are fulfilled with equality.

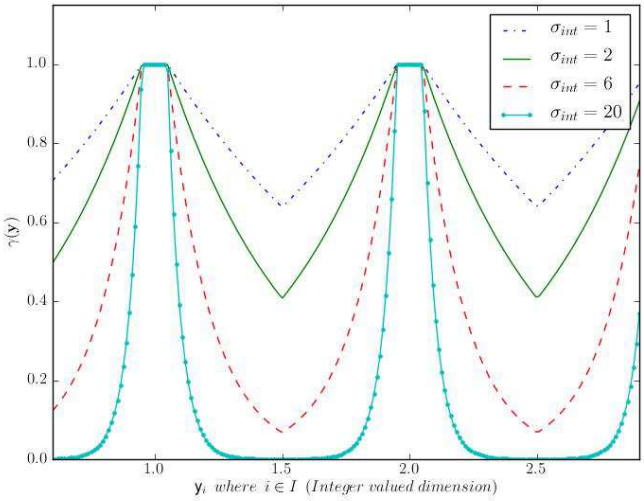


Figure 8: The term γ as described in (54) as a function of $\mathbf{y} \in \Omega$ (seen in one dimension). In the figure $\sigma_{int} = 1,2,6,20$ has been used and $\delta = 0.05$.

4 Implementation of general algorithm

The methods presented in Section 3 are in this section used to construct a working algorithm. The algorithm is still general and can be used on a wide range of problems. First the algorithm is described in general with pseudo code for the single objective case. Further, in Section 4.2.1 the implementation of the quality function is discussed for the single objective case and in Section 4.2.4 the topic is implementation of integration and domain reduction. In Section 4.3 the adaptations necessary to extend the algorithm to the multiobjective case are discussed followed by a section talking about the DIRECT solver used.

The optimization problem of interest in this report is the optimization problem (2), introduced in Section 1.4, with decision variables $\chi_1, \chi_2, \ldots, \chi_d$. Recall also the notation $\boldsymbol{\chi} = (\chi_1, \chi_2, \ldots, \chi_d)$ given by (3). Further, in Section 2 and Section 3 some theory were presented using general points $\mathbf{x} \in \mathbb{R}^d$. Here in Section 4, we put $\mathbf{x} = \boldsymbol{\chi}$ in order to be able to easily compare the parts of the algorithm with the theory presented in the previous two sections.

4.1 Single objective algorithm

The algorithm presented here is general and in Section 5 the necessary changes to make it work with the Carmen crew optimizers are described. The pseudo code for the single objective algorithm is presented in Algorithm 4.1.

Algorithm 4.1: SINGLE OBJECTIVE ()

Choose search domain, Ω
Set dimensions which are integer valued, $\mathbb{I}$
Set range for output constraints, $\mathcal{D}$
Set nr. initial pts n_{init}, nr. parallel jobs N_p and nr. batches of jobs N_b
Create initial set of points X or retrieve points from earlier optimization
Calculate values for output constraints and the objective: $(\mathbf{g}, \mathbf{f}) = send\_jobs(X)$
for 1,2 $\ldots, N_b$ **do**
    Calculate surrogate function $S(\mathbf{x})$ for objective function
    Calculate $S_{min} = \min_{\mathbf{x}\in\Omega} S(\mathbf{x})$
    Calculate $S_{max} = \max_{\mathbf{x}\in\Omega} S(\mathbf{x})$
    Calculate surrogate functions $\mathbf{S}_g(\mathbf{x}) = (S_{g_1}(\mathbf{x}), \ldots, S_{g_n}(\mathbf{x}))$ for output constraints
    Chose σ, σ_c
    Set $X^0 = \emptyset$
    for 1,2, $\ldots, N_p$ **do**
        $\mathbf{y}^* = find\_new\_point(S_{min}, S_{max}, S(\mathbf{x}), \mathbf{S}_g(\mathbf{x}), \sigma, \sigma_c, X)$
        Ensure that $\|\mathbf{y}^* - \mathbf{x}\| > \delta \ \forall \mathbf{x} \in X$ and some δ, and that $y_i^* \in \mathbb{Z}$ for $i \in \mathbb{I}$
        Add $\mathbf{y}^*$ to X^0: $X^0 = X^0 \cup \{\mathbf{y}^*\}$
    end for
    Calculate new values for output constraints and the objective: $(\mathbf{g}^0, \mathbf{f}^0) = send\_jobs(X^0)$
    Add the new points to $\mathbf{g}, \mathbf{f}$: $\mathbf{g} = \mathbf{g} \cup \mathbf{g}^0$, $\mathbf{f} = \mathbf{f} \cup \mathbf{f}^0$
    Add X^0 to X: $X = X \cup X^0$
end for
Set $X_f = \{\mathbf{x} \in X : S_g(\mathbf{x}) \in \mathcal{D}\}$: the set of feasible decision vectors
Find $\bar{\mathbf{x}} = \arg\min_{\mathbf{x}\in X_f} \{f(\mathbf{x})\}$
return $\bar{\mathbf{x}}$

Initially, the problem specific data has to be specified. A box-domain $\Omega \subset \mathbb{R}^d$ is chosen over which optimization is conducted where the dimensions $\mathbb{I}$ are integer valued. The range for the

output constraints $\mathcal{D}$ is set, also given by a hyper-box. The number of initial points n_{init} as well as the number of parallel computations N_p and the number of batches of point N_b are set where $n_{init} > d$ and d is the dimensionality of Ω.

An initial set X of n_{init} points is created using the Latin Hypercube Design described in Section 3.2. The algorithm also supports the possibility for a warm start: utilizing a set of points determined by the user. The pseudo code for the generation of initial points can be seen in Algorithm 4.2. Recall that the domain Ω is a hyper-box with upper and lower bounds in dimension j given by u_j^v and l_j^v respectively.

Along each dimension $k = 1,2,\ldots,d$, n evenly distributed coordinates are chosen: $X^k = \{z_{1k},z_{2k},\ldots,z_{nk}\}$ such that $l_k^v = z_{1k} < z_{2k} < \ldots < z_{nk} = u_k^v$. Then decision vectors $\mathbf{x}_i, i = 1,\ldots,n$ are created by randomly picking an element from each set X^k without replacement until the sets are empty. Then a set of points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ is obtained which is evenly distributed in a sense but it can not be guaranteed that that X is unisolvent. In order to create a set X that is unisolvent (with an outmost probability) a correlation test is performed. The maximum absolute value of any off-diagonal element in the correlations matrix given by X provides a good measure of how well spread out the set of points are. The correlations matrix is given by $\{\rho_{ij}\}$ and after a fixed number of tries the set of points will be chosen which has the lowest value of $\max_{i\neq j} |\rho_{ij}|$.

Algorithm 4.2: CREATE INITIAL SET OF POINTS (Ω, n_{init})

Set number of retries
Set $\rho_{min/max} = 1$ and $X = \emptyset$ and denote $n = n_{init}$
for loop index = 1,2,..., number of retries **do**
  for $k = 1,2,\ldots,d$ **do**
    Select n evenly spaced coordinates along the k^{th} dimension in Ω:
    $X^k = \{z_{1k},z_{2k},\ldots,z_{nk}\}$ such that z_{1k}, z_{nk} are at, or close to, the boundary of the k^{th} dimension of Ω.
  end for
  for $j = 1,2,\ldots,n$ **do**
    Randomly pick a coordinate j_k from each set $X^k, k \leq d \Rightarrow \mathbf{x}_j = (z_{j_1 1},\ldots,z_{j_d d})$
    Each chosen coordinate $z_{j_k k} \in X^k$ is removed from X^k: $X^k = X^k \backslash \{z_{j_k k}\}, k = 1,\ldots,d$
  end for
  Calculate correlation matrix $\mathcal{P} = \{\rho_{ij}\}_{i,j=1,\ldots,n}$ from $X_0 = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$:
  if $\rho_0 \equiv \max_{i\neq j} |\rho_{ij}| < \rho_{min/max}$ **do**
    Set $\rho_{min/max} = \rho_0$
    Set $X = X_0$
  end if
end for
return X

The evaluation of the expensive black box function is done using the *send_jobs* function. This function is problem dependent and in using test functions such as the Branin function, see (59), it merely returns a simple function value. The implementation of this function for the Carmen crew and fleet optimizers is discussed in Section 5.3 where the pseudo code is presented in Algorithm 5.2.

The values for the objective function $\mathbf{f}$ and the output constraints $\mathbf{g}$ received from *send_jobs* are used to find new points to evaluate. This is done as many times as set by the number of batches of jobs N_b. How the selection of new points is done is discussed in more detail in Section 4.2.

The algorithm returns the point $\bar{\mathbf{x}}$ which has the lowest calculated value $f(\bar{\mathbf{x}})$ while the output constraints $\mathbf{S}_g(\bar{\mathbf{x}})$ are within the set bounds $\mathcal{D}$. At every point where the function has been evaluated, the value of the surrogate model corresponds to the objective value since $S(\mathbf{x})$ has been created through interpolation. The same is true for the output constraints.

4.2 Finding new points: Implementation

The selection of new points is based on the parts of the algorithm presented in Section 3. First the surrogate function $S(\mathbf{x})$ is calculated, or basically the coefficients $(\lambda, \mu)^T$ are calculated from (36) using the objective values $\mathbf{f}$ and the surrogate model is then given by (38). The surrogate models for the output constraints $\mathbf{S}_g(\mathbf{x})$ are calculated in the same manner using the output constraint values $\mathbf{g} = \{\mathbf{g}_1, \ldots, \mathbf{g}_\eta\}$. The minimum and maximum values of the surrogate function, S_{min} and S_{max} respectively are calculated using the DIRECT algorithm and values for σ and σ_c are chosen. The choice of σ and σ_c depends on N.

For every batch of jobs, hence for every batch of points to evaluate, a number of points N_p are chosen to be processed in parallel. This is done in the function *find_new_point* whose pseudo code is presented in Algorithm 4.3.

Algorithm 4.3: FIND_NEW_POINT $(S_{min}, S_{max}, S(\mathbf{x}), \mathbf{S}_g(\mathbf{x}), \sigma, \sigma_c, X)$

except the first run **do**
  Calculate surrogate function $S(\mathbf{x})$, $\mathbf{x} \in \Omega$ for objective function
  Calculate surrogate functions $\mathbf{S}_g(\mathbf{x}) = (S_{g_1}(\mathbf{x}), \ldots, S_{g_\eta}(\mathbf{x}))$ for output constraints
end except
//Choose and construct merit function:
if (Merit function = NQF) **do**
  Ω_y=Reduce_domain(Ω)
  Ω_y^B=Solve_LP(Ω_y), (see Section 4.2.4)
  Set $Q(\mathbf{y}) = \gamma(\mathbf{y})\varphi(\mathbf{S}_g(\mathbf{y})) \int_{\Omega_y^B} (U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x}))\, \omega(S(\mathbf{x}))d\mathbf{x}$
else
  // Merit function = NNQF
  Set $Q(\mathbf{y}) = \gamma(\mathbf{y})\varphi(\mathbf{S}_g(\mathbf{y}))U_X(\mathbf{y})\omega(S(\mathbf{y}))$
end if
Calculate $\mathbf{y}^* = \arg\max_{\mathbf{y}\in\Omega} Q(\mathbf{y})$
return $\mathbf{y}^*$

The first time this function is called ($N_b = 1, N_p = 1$) the surrogate functions for the objective function and output constraints that were created initially are still valid. If not the first time then the surrogates have to be recalculated. Then the form of the merit function (quality function) is chosen, either the neighborhood method or the non-neighborhood method. For both methods, the following factors have to be calculated: $\gamma(\mathbf{y})$ for integer constraints, $\varphi(\mathbf{y})$ for output constraints, $U_X(\mathbf{x}) - U_{X\cup\mathbf{y}}(\mathbf{x})$ or $U_X(\mathbf{x})$ for spatial dependence and $\omega(\mathbf{x})$ for dependence on surrogate function value. Additionally, for the integral method the domain is reduced from Ω to Ω_y and further to Ω_y^B, see Section 4.2.4, and the integral has to be evaluated.

The implementation of these factors and their dependence on the choice of σ, σ_c are discussed in the following sections. The function then calls the DIRECT global optimization solver to find $\mathbf{y}^* = \arg\max_{\mathbf{y}\in\Omega} Q(\mathbf{y})$ which is then returned.

As seen in Algorithm 4.1, the selected point $\mathbf{y}^*$ is then checked for feasibility according to some criteria. First it cannot be closer than δ to any point in X where δ is set to $\delta = d^{1/2}/120$ to

ensure that if $\mathbf{y}^*$ is given in hours then rounding $\mathbf{y}^*$ to minutes will still ensure $\mathbf{y}^* \neq \mathbf{x}_k \ \ \forall \mathbf{x}_k \in X$. Also the dimensions which are set as integer valued must be ensured to be so also for $\mathbf{y}^*$. This is further discussed in Section 4.2.3.

4.2.1 Weight function

As discussed earlier the weight function has the following form

$$\omega(S(\mathbf{x})) = \exp\left(-\sigma \frac{S(\mathbf{x}) - S_{min}}{S_{max} - S_{min}}\right).$$

The range of $\omega(S(\mathbf{x}))$ is $[e^{-\sigma}, 1]$ where the lower bound is reached for $S(\mathbf{x}) = S_{max}$ and the upper bound is reached for $S(\mathbf{x}) = S_{min}$. Seen from another perspective, say that σ is very large and fixed. In maximizing $Q(\mathbf{y})$, points where $\omega(S(\mathbf{x}))$ is large are obtained, hence points will be found whose surrogate values are close to S_{min}. Similarly, with a fixed low value of σ then $\omega(\mathbf{x})$ is very flat and hence the value of $S(\mathbf{x})$ has no real importance. Therefore, with low values of σ, a space filling search will be conducted whereas with σ large the minimum of the surrogate will be found. This is illustrated in Figure 9 where the surrogate function $S(\mathbf{x})$ is shown as well as the weight function for different values of σ. It can be seen that the weight function gets increasingly localized at the minimum as σ increases.

This fact is utilized in the algorithm when choosing the value of σ. The surrogate model is created from a set of points X and at an early stage of the procedure, when there are few data points available, the uncertainty in the model will be large. At this point a local search, hence finding the minimum of the surrogate, will in most cases not prove advantageous due to the uncertainty of the model. Instead a global space filling search is preferred in order to increase the trust that can be placed in the model and hence decrease the uncertainty. This corresponds to first using $\sigma = 0$ where the surrogate's value is disregarded and only a space filling search is conducted in order to increase the accuracy of the model. Then σ is gradually increased, creating a more accurate model only where the model predicts low values, and finally when σ is very high the lowest predicted value is found.

As seen in Algorithm 4.1, σ is chosen for every N_b where N_b is the number of batches of jobs. Therefore $\sigma = \sigma(N_b)$ and in selecting σ, discrete levels are used. Namely, certain values of σ are chosen, {0,5,10,20,50}, as well as the percentage of times they are supposed to be used, say {20,15,25,25,15}. Then a vector is created where for $N_b = 10$ batches the vector would have the form $\sigma(10) = [0, 0, 5, 5, 10, 10, 10, 20, 20, 50]$. For the case where the interest is not to find the minimum value but only increase the reliability of the model (e.g. for the Prediction Model), then σ is set very low, e.g. $\sigma(10) = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$. With this setting, a space filling search is conducted.

In working with black-box functions the parameters σ cannot be changed depending on the problem but has to be selected using the method described above where the set of values and the corresponding percentages are set heuristically. The heuristically chosen values are set as above.

4.2.2 Output constraints function

The output constraints function has the following form

$$\varphi(\mathbf{S}_g(\mathbf{y})) = \prod_{i=1}^{\eta} \min\left\{1, e^{-\sigma_c(S_{g_i}(\mathbf{y}) - u_i)}\right\} \min\left\{1, e^{-\sigma_c(l_i - S_{g_i}(\mathbf{y}))}\right\} \tag{56}$$

and is created in analogy with the weight function. The parameter σ_c also has the similar functionality as σ has for the weight function although here the uncertainty requiring σ_c to be

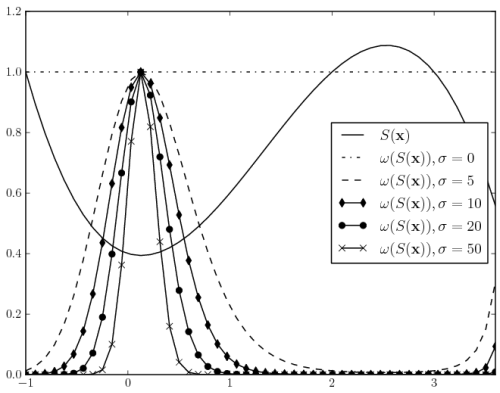


Figure 9: A surrogate function $S(\mathbf{x})$ as well as the weight function $\omega(S(\mathbf{x}))$ (in one dimension) for different values of σ.

changed depends not only on the uncertainty of the surrogate model of the objective function but also on the uncertainty of the surrogate model of the output constraints.

In Figure 10 a function is multiplied with the output constraints function (56) for different σ_c with $\eta = 1$. The output constraints function is bounded above by u_1 in such a way that $x < 3$ is infeasible. From (56) and Figure 10 it can be seen that using $\sigma_c = 0$ the output constraints are entirely neglected hence the top graph in the figure with $\sigma_c = 0$ corresponds to the function itself. From the figure it is clear that increasing σ_c will create a steeper barrier, which is the sought property. Although, since the surrogate is an approximation of the real function, an error in the surrogate could place the real optimal point on the wrong side of the barrier and hence not be chosen. Therefore the value of σ_c is gradually increased, accounting for the uncertainties in the surrogate models. This is done using a list of values and a list with frequency of occurrence in analogy with the weight function where an example is $\sigma_c = [0, 0, 2, 2, 2, 4, 4, 7, 7, 10]$.

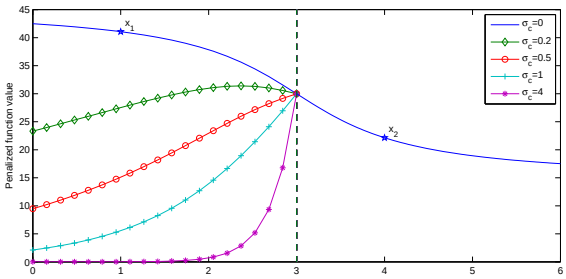


Figure 10: A function multiplied by the output constraints function $\varphi(\mathbf{S}_g(\mathbf{y}))$ (in one dimension) for different values of σ_c where the bounds on $\mathcal{D}$ are such that $x < 3$ is infeasible.

The output constraints function is not integrated over since the feasibility of the surrounding points does not necessary say anything about the feasibility of the point itself, where feasibility means $\mathbf{S}_g(\mathbf{y}) \in \mathcal{D}$. Moreover, in using the neighborhood quality function with the output constraints function as a part of the integrand, the integration would be done over parts that are penalized e.g. for the point $x = 3$ in Figure 10 one would integrate over $[2, 3.5]$ (half the distance to any other point, see Figure 6) where the value has been made lower in $[2, 3]$. Therefore, that point would not have been picked even though it is the most promising feasible point.

4.2.3 Integrality constraints function

The integrality constraints function is not a constraint per se but is used to try to gently push the optimizer towards integer valued coordinates in some dimensions. These dimensions are given by $\mathbb{I}$ as discussed before and the function used for this purpose is

$$\gamma(\mathbf{y}) = \prod_{j \in \mathbb{I}} \min\left\{1, e^{-\sigma_{int}(\Delta_j(\mathbf{y})-\delta)}\right\}$$

where

$$\Delta_j(\mathbf{y}) = \min_{\tilde{y} \in \mathbb{Z}} |\tilde{y} - y_j| = \min\{\lceil y_j \rceil - y_j, y_j - \lfloor y_j \rfloor\}$$

and whose graph is given in Figure 8. Here $\mathbf{y} = (y_1, y_2, \ldots, y_d)$ and δ is the neighborhood around the integer where $\gamma(\mathbf{y}) = 1$. In Figure 11 a function is multiplied with the integrality constraints function for different values of σ_{int}. From the figure it is clear that increasing σ_{int} too much will give a largely oscillating quality function which is not favorable since it is hard to optimize. Instead with a lower σ_{int}, locally around every integer the maximum will be pushed towards the integer while at the same time keeping a moderately oscillating quality function.

One setting for σ_{int} will obviously not work for all cases but for cases where the function (integral over spatial term and weight function) changes moderately, the setting $\sigma_{int} = 1$ provides good results. This setting is kept for all cases since too much oscillation will do more harm than good.

Also, if the range of integer values for one dimension is large enough it can be argued that that particular dimension could be relaxed to be seen as continuous. In fact, for dimensions j where the range of values is large, then j has temporarily been removed from $\mathbb{I}$ when creating $\gamma(\mathbf{y})$.

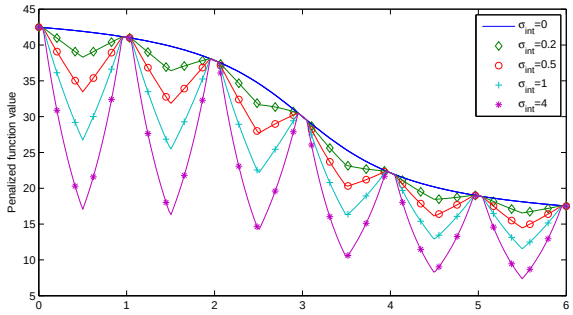


Figure 11: A function multiplied by the integrality constraints function $\gamma(\mathbf{y})$ for different values of σ_{int}.

4.2.4 Spatial function: Integration

The spatial function $U_X(\mathbf{x}) - U_{X \cup \mathbf{y}}(\mathbf{x})$ aims to put distance between points and it has been shown that it is non-zero in a bounded polyhedron given by $\Omega_y = \{\mathbf{x} : D\mathbf{x} \leq \mathbf{e}, \mathbf{x} \in \Omega\}$ where $D_{\cdot j} = (\mathbf{x}_j - \mathbf{y})^T$ and $e_j = \frac{1}{2}(\mathbf{x}_j - \mathbf{y})^T(\mathbf{x}_j + \mathbf{y})$.

Using this, the integral part of the quality function can be expressed as

$$\int_{\Omega_y} \left(U_X(\mathbf{x}) - U_{X \cup \mathbf{y}}(\mathbf{x})\right) \omega(S(\mathbf{x})) d\mathbf{x}$$

but this also brings up a problem. The integral over Ω can be computed using Mote Carlo integration according to (47) which is particularly easy since Ω is a hyper-box and it is easy to

pick random points inside a hyper-box as well as to calculate its volume. However, neither are simple for the polyhedron.

The reduction of domain from Ω to Ω_y is done mainly in order to increase the accuracy of the numeric integration. It also has the favorable feature that Ω_y decreases as the number of points in X increases hence the accuracy increases as the algorithm advances.

It would be preferred to integrate over a box while at the same time keep the higher accuracy achieved through reducing to Ω_y. However, integration over Ω_y with the addition of some parts of $\Omega \backslash \Omega_y$ will only worsen the accuracy slightly. Therefore in creating the smallest possible box that contains Ω_y both the accuracy of Ω_y and the simplicity of the box will be kept.

It is more practical not to create the smallest box possible that contains Ω_y but the smallest box containing Ω_y that is aligned with the axes of the Euclidean coordinate system. In doing so the max and min coordinates of the polyhedron in every dimension has to be calculated; denote them $q_j^{(+1)}$ and $q_j^{(-1)}$ respectively for dimension j. For each $j \in \mathcal{J}$, that problem can be formulated as the following simple LP

$$\begin{aligned} & q_j^{(\xi)} = \arg\max \ \ \xi \, \mathbb{e}_j \cdot \mathbf{x} \equiv \xi \chi_j \\ & \text{subject to} \\ & \qquad D\mathbf{x} \leq \mathbf{e} \\ & \qquad \mathbf{x} \in \Omega. \end{aligned} \tag{57}$$

where $\mathbb{e}_j$ is the unit vector in the j^{th} dimension, hence e.g. $\mathbb{e}_3 = (0, 0, 1, 0, \ldots, 0)$ and ξ is $+1$ or -1. Using this, the box is given as $\Omega_y^B = \{\mathbf{x} = (\chi_1, \ldots, \chi_d) : q_j^{(-1)} \leq \chi_j \leq q_j^{(+1)}, j \in \mathcal{J}\}$.

Finding Ω_y^B requires solving a number of $2d$ LP problems hence solving (57) for $\xi = \pm 1$ for $j \in \mathcal{J}$. For this the commercial LP solver *Xpress* is used where an *mps* file is automatically written for every problem.

4.3 Adaptation to multiobjective

In adapting the single objective algorithm, Algorithm 4.1, to multiobjective optimization, little has to be done in terms of algorithm structure. In algorithm 4.4 pseudo code for the multiobjective algorithm is shown.

Algorithm 4.4: MULTIOBJECTIVE ()

Set/Choose $\Omega, I, \mathcal{D}, n_{init}, N_b, N_p, X$ as in the single objective case
Calculate values for output constraints and the objectives: $(\mathbf{g}, \mathbf{f}) = send\_jobs(X)$
for 1,2 $\ldots, N_b$ **do**
  Calculate surrogate functions $\mathbf{S}(\mathbf{x})$ for objective functions
  Calculate surrogate functions $\mathbf{S}_g(\mathbf{x}) = (S_{g_1}(\mathbf{x}), \ldots, S_{g_\eta}(\mathbf{x}))$ for output constraints
  Find the Pareto front $\mathbf{S}^*$, make it denser and extend it
  Find the max distance to the Pareto front, dist$(\mathbf{S}(\tilde{\mathbf{x}}))$: $\tilde{\mathbf{x}} = \arg\max_{\mathbf{x}\in\Omega} \text{dist}(\mathbf{S}(\mathbf{x}))$
  Chose σ, σ_c and set $X^0 = \emptyset$
  for 1,2,$\ldots$,N_p **do**
    $\mathbf{y}^* = find\_new\_point(\tilde{\mathbf{x}}, \mathbf{S}^*, \mathbf{S}(\mathbf{x}), \mathbf{S}_g(\mathbf{x}), \sigma, \sigma_c, X)$
    Ensure that $\|\mathbf{y}^* - \mathbf{x}\| > \delta \ \forall \mathbf{x} \in X$ and some δ, and that $y_i^* \in \mathbb{Z}$ for $i \in \mathbb{I}$
    Add $\mathbf{y}^*$ to X^0: $X^0 = X^0 \cup \{\mathbf{y}^*\}$
  end for
  Calculate new values for output constraints and the objectives: $(\mathbf{g}^0, \mathbf{f}^0) = send\_jobs(X^0)$
  Add the new points to $\mathbf{g}, \mathbf{f}$: $\mathbf{g} = \mathbf{g} \cup \mathbf{g}^0$, $\mathbf{f} = \mathbf{f} \cup \mathbf{f}^0$
  Add X^0 to X: $X = X \cup X^0$
end for
Set $X_f = \{\mathbf{x} \in X : \mathbf{S}_g(\mathbf{x}) \in \mathcal{D}\}$: the set of feasible decision vectors
return evaluated Pareto points $\{\mathbf{x}_l \in X_f : \nexists \mathbf{x}_k \in X_f \ s.t. \ f(\mathbf{x}_k) \preceq f(\mathbf{x}_l)\}$

The only major difference from the single objective case is the weight function, which is why the quality function was chosen as the merit function to begin with. As given in Section 3.5.2, the weight function has the following form

$$\omega(\mathbf{S}(\mathbf{x})) = \exp\left(-\sigma \frac{\text{dist}(\mathbf{S}(\mathbf{x}))}{\text{dist}(\mathbf{S}(\tilde{\mathbf{x}}))}\right)$$

where (58)

$$\text{dist}(\mathbf{S}(\mathbf{x})) = \min_{\mathbf{s}^* \in \mathbf{S}^*} \|\mathbf{S}(\mathbf{x}) - \mathbf{s}^*\|_{\mathbf{S}(\Omega)}$$

and where $\mathbf{S}^*$ is the Pareto front and hence dist$(\mathbf{S}(\mathbf{x}))$ is the distance from $\mathbf{S}(\mathbf{x})$ to the Pareto front. Also dist$(\mathbf{S}(\tilde{\mathbf{x}}))$ is the maximum distance from any point in the range of $\mathbf{S}(\mathbf{x})$ to the Pareto front. The Pareto front is found using a Python implementation of MOEA, Multi Objective Evolutionary Algorithm. More exactly, points $\mathbf{T}^* \subset \mathbf{S}^*$ are found that approximate the Pareto front, see Figure 12(a) which is the output from the MOEA solver. In order to get a more dense and more well spread out set of points, additional points from $\mathbf{S}^*$ are added to $\mathbf{T}^*$ through linear interpolation between the already existing points in $\mathbf{T}^*$. From the nature of the Pareto front this is a well defined action and the new points will either approximate the Pareto front well or lay outside the set, as can be seen in Figure 12(b). Using the extended set $\mathbf{T}^*$, the distance function can easily be calculated as

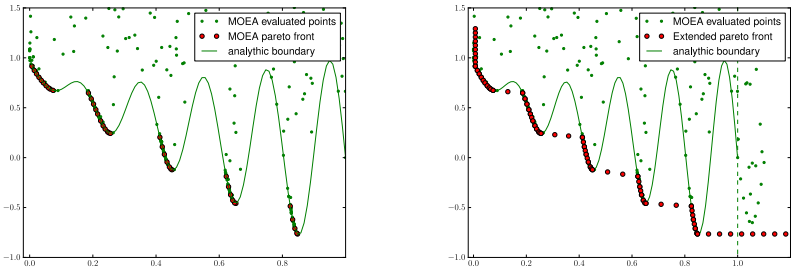
$$\text{dist}(\mathbf{S}(\mathbf{x})) = \min_{\mathbf{s}^* \in \mathbf{T}^*} \|\mathbf{S}(\mathbf{x}) - \mathbf{s}^*\|_{\mathbf{S}(\Omega)}$$

hence a minimum over a finite set $\mathbf{T}^*$ instead of a non-linear minimization problem. The Pareto front is extended upwards and to the right due to points $\mathbf{x}$ with high values for one output surrogate $S_1(\mathbf{x})$ but whose value for the other output surrogate $S_2(\mathbf{x})$ is only slightly dominated. This pair $(S_1(\mathbf{x}), S_2(\mathbf{x}))$ will then be far from the Pareto front even though a slight reduction of $S_2(\mathbf{x})$, say for $\mathbf{x} = \mathbf{x}_{new}$ would place $(S_1(\mathbf{x}_{new}), S_2(\mathbf{x}_{new}))$ at the Pareto front.

In Figure 12(a) the MOEA solver is used on the ZDT3 function, see Appendix E [ZDT00], showing the Pareto front found by MOEA as well as the rest of the MOEA evaluated points and the analytical boundary. The analytical Pareto front is of course a part of the analytical

boundary. In Figure 12(b) the Pareto front, as given by the MOEA solver, is extended and has been made more dense. Also, in Figure 12(b) the ZDT3 has been modified to the right of the dashed vertical line in order to illustrate the necessity of an extended Pareto front as mentioned earlier. The sampled points at the bottom right corner are close to the extended Pareto front but far from the Pareto front found by MOEA.

If the multiobjective problem in question does not have a trade-off between the objectives, hence they are highly (positively) correlated, then the Pareto front may consist of a singleton and the problem is reduced to finding a minimum in analogy with the single objective case.



(a) Pareto front as given by the MOEA solver.

(b) The more dense and extended pareto front.

Figure 12: The Pareto front, found by the MOEA solver, compared to the extended and more dense Pareto front where the points evaluated in MOEA and the boundary, and hence the analytic Pareto front, are shown.

4.4 DIRECT solver

As mentioned in Section 4.2 the global optimization algorithm used to find $\mathbf{y}^* = \arg\max_{\mathbf{y}\in\Omega} Q(\mathbf{y})$ as well as S_{min} and S_{max} is the DIRECT method. It divides the sample space Ω into hyper-rectangles, samples at their centers and divides if the criteria for potential optimality in (32) is satisfied. In order to be independent of commercial software, an own version of the DIRECT solver has been implemented in Python. In doing so a Matlab implementation by Finkel [Fin03] was used as a template. The scheme for the DIRECT method is described in Section 2.4 but the pseudo code is presented here in Algorithm 4.5 together with a short description.

Algorithm 4.5: DIRECT $(f(\mathbf{x}), bounds, options)$

Normalize the domain to be $\hat{\Omega} = \{\mathbf{x} = (\chi_1, \ldots, \chi_d) : 0 \leq \chi_j \leq 1, j \in \mathcal{J}\}$ with center c_1
Find $f(c_1), f_{min} = f(c_1), i_{ts} = 1, e_{val} = 1$
Evaluate $f(c_1 \pm \delta e_j), j \in \mathcal{J}$, and divide hyper-cube
while $i_{ts} \leq maxits$ and $e_{val} \leq maxevals$ **do**
    Identify the set $\mathcal{H}$ of all potentially optimal hyper-rectangles
    for all $\mathcal{H}_k \in \mathcal{H}$
        Identify the longest side(s) of hyper-rectangle $\mathcal{H}_k$
        Evaluate f at centers c_k of new rectangles, divide $\mathcal{H}_k$ into smaller hyper-rectangles
        Update f_{min}, $xatmin$ and e_{val}
    end for
    $i_{ts} = i_{ts} + 1$
end while
return $xatmin$

In short words, the DIRECT algorithm begins by normalizing the space to be the unit cube $\hat{\Omega}$ with center in c_1. Further, function values: $f(c_1)$ and $f(c_1 \pm \delta e_j), j \in \mathcal{J}$ are found and the hyper-cube is divided as described in Section 2.4.

The maximum number of iterations and the maximum number of evaluations are set in *options*, and while they have not been exceeded, the following is done: all potentially optimal hyper-rectangles $\mathcal{H}$ are located and for each $\mathcal{H}_k \in \mathcal{H}$ the longest side d_k is found, the function is evaluated at the center c_k of hyper-rectangle $\mathcal{H}_k$ giving $f(c_k)$ and then $\mathcal{H}_k$ is divided into smaller hyper-rectangles. The sampled point $\bar{\mathbf{x}} = xatmin$ which has the lowest function value $f(\bar{\mathbf{x}})$ is then returned.

5 Implementation in RUSA

The constructed algorithm itself is fairly general and can be used for almost every black box optimization problem. In order to use it on Jeppesen's crew optimizers it was necessary to among other things be able to open optimization jobs, choose parameters and in an automated way send and run jobs and receive data from jobs. There is a program RUSA (Rule Sensitivity Analysis), a thesis work in 2008 by Joel Driessen [Dri08], that takes care of most of these things.

So, through RUSA there is support for sending and rececing jobs and visualizing the results which will be discussed in more detail in Section 5.1. This thesis work adds some more logic to RUSA, namely trying to figure out which jobs that are worth evaluating in order to find the best solution or to minimize the over-all error for a prediction model. The requirements for the additions to the RUSA GUI are discussed in Section 5.2.

5.1 Original RUSA

The original RUSA [Dri08] consists of tabs 1,3,4,5,6 in Figures 13 and 14, hence the ones that are not shown in the figures. The original RUSA, consisting only of tabs 1,3,4,5,6 (Planner, Jobs, KPI Table, Visualization, KPI Correlations) works as follows:

0. ***Menu***: Open experiment/ create new experiment
1. ***Planner***: Choose parameters & rules and set the values they shall be evaluated at. Then the jobs are generated as an outer product of the values chosen, hence d parameters each with m_j values $j = 1,2,\ldots,d$ will generate $\prod_{j=1}^{d} m_j$ jobs.
3. ***Jobs*** : The individual jobs can be viewed and removed before starting the optimization and during the run the number of solutions can be presented.
4. ***KPI Table***: For each job, the KPIs can be viewed
5. ***Visualization***: The results can be plotted
6. ***KPI Correlations***: The correlations between the KPIs can be calculated.

The RUSA software is divided into two parts: the GUI (Graphical User Interface) and the application. The application part handles the data and logic whereas the GUI part handles the user interface. A thorough description of RUSA and its structure is given in [Dri08].

5.2 Additions to RUSA GUI

RUSA was created from a specific requirement specification but the addition of the Parameter Tuning puts other requirements on the program. The program's structure therefore does not provide simple support for all the additions required. The requirements for the GUI with the addition of Automated Optimization and Prediction Model are summarized as follows:

- Setting parameters and domain: min, max
- Selecting KPIs as objectives/constraints and setting bounds on such constraints
- Choosing number of jobs to run
- Resume optimization (*would be good*)
- While algorithm is running:

  - Jobs are shown in the *Jobs* tab
  - When jobs are finished the KPIs are shown in *KPI Table*
  - Solution status updated in *Jobs* tab (*would be good*)
- The plotting tools in *Visualization* work as before
- Correlations can be calculated as before
- Solutions are saved continuously
- The prediction model: choosing a set of parameters and calculating the predicted KPIs
- The possibility to run the job corresponding to the estimation in the prediction model. (*would be good*)
- After loading an experiment, all of the above shall function as well
- The original RUSA must work

In adding the Parameter Optimization functionality as well as the Prediction Model, while at the same time keeping the original RUSA working, two new tabs were created: *Automated Optimization* (Figure 13), and *Prediction Model* (Figure 14).

The original tabs in RUSA are intact but also used for the parameters in Automated Optimization. If intending to use the Automated Optimization, then when setting the values of each parameter[5] in the *Planning* tab, the range of those values is set at the bounds for that parameter. They constitute the domain of the search and can be integer valued, Boolean as well as Reltime (a measure of time rounded to minutes).

5.2.1 Automated optimization

In the upper left corner of Figure 13 a preliminary list of the available KPIs is found; a full list can only be found after running a job with the original RUSA. From this list, KPIs can be selected to the *Selection* area to the right by clicking *Add Selected KPI* and they can be set as objectives or constraints using the check button. There can be at most two objectives and there must be at least one. The upper and lower bounds for the KPI constraints are set to 0 and 10^{10} as default and they can be changed by clicking at them.

The *Settings* at the bottom left corner are used to govern the algorithm. Choosing *General Settings* the user sets how many jobs the algorithm shall run in total whereas *Detailed Settings* allows the user to choose the number of initial points n_{init}, the number of jobs to run simultaneously N_p, constituting one batch of jobs, and the number of (serial) batches to run N_b. For a given optimization problem, it is natural that the number of batches is the setting that has the largest impact on the duration of the algorithm.

For each batch of jobs to run, the parameter σ is increased according to an empiric rule; increasing σ will give a more local search and hence trusting the surrogate more. For the prediction model case it might be more important to minimize the overall error, not just close to the minima. In checking *Do Only Prediction Model* the parameter σ will be set to zero and hence a space filling search is conducted. Checking *Run Express Calculation* will use the non-neighborhood method for finding new points instead of the neighborhood method which is chosen as default.

Before running the optimization, the chosen jobs can be viewed by pressing *Show Jobs*. The jobs will then be presented in the *Jobs* tab. *Run Optimization* will run the automated

[5]RUSA also supports turning rules on and off but it is a functionality of RUSA that is not used in this thesis

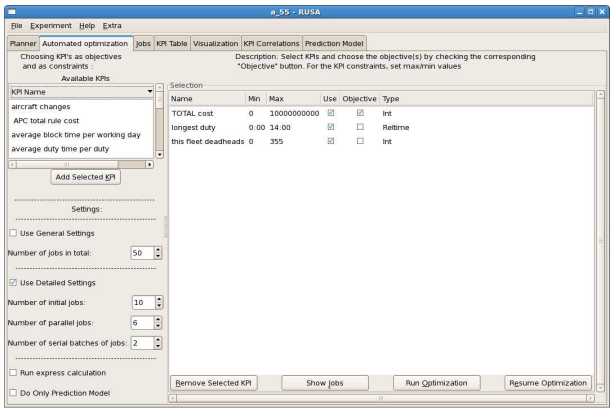


Figure 13: The new tab *Automated Optimization* in RUSA

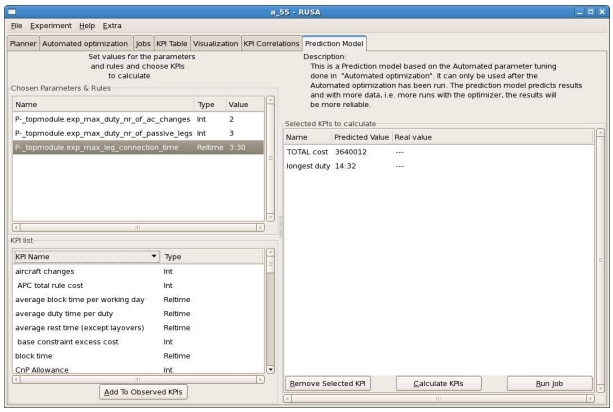


Figure 14: The new tab *Prediction Model* in RUSA

optimization after asking which optimizer (i.e. pairing optimizer or rostering optimizer) and which queue to use. The jobs will appear in *Jobs* and the number of solutions will be updated automatically. When a batch of jobs has finished, the KPI values are shown in *KPI Table* and the algorithm continues for as long as prescribed.

If the algorithm has terminated, hence running the number of jobs set by the user, and it is of interest to continue finding better solutions or a more reliable prediction model, then the user can do so by using *Resume Optimization*. Every setting in the current tab can be changed in between optimizations although for the optimization to be resumed, the choice of parameters in *Planner* cannot be altered.

5.2.2 Prediction model

To the left in Figure 14, the Parameters chosen in *Planner* are shown. Their values can be changed by clicking at them, giving a parameter setting that one wishes to estimate. On the right the selected KPIs are shown, hence the ones one wishes to investigate. The KPIs selected for the optimization as objectives and constraints are set as default but more can be added from the list of KPIs at the bottom left.

In pressing *Calculate KPIs*, the Prediction Model estimates the KPI values by evaluating the surrogate function for the given parameter setting and presents them in the column *Predicted Value*. To run this particular job; to see what the real KPI values are, just press *Run Job* and the values will appear in the column *Real values* when the job is finished. If the job corresponding to the chosen combination of parameters has already been run, *Run Job* can safely be pressed and without running the job, the values will be presented. Note that at the points evaluated, the surrogate model will always provide an exact answer since the surrogate is an interpolation of the data given by the jobs that have been run.

5.3 GUI - Algorithm interface

In order to utilize the optimization algorithm discussed in the previous chapter on the Carmen crew optimizers, parameters, KPI values and other settings that are set in the GUI are needed. In order to keep the GUI user friendly and account for all the requirements discussed earlier, this concerns a fairly large amount of code and a high-level algorithm connecting the RUSA GUI to the solver can be seen in Algorithm 5.1.

Algorithm 5.1: GUI-Algorithm connection ()

IN RUSA GUI:
    Choose parameters and set bounds for them: $[l_j^v, u_j^v]$, $j \in \mathcal{J}$
    Choose KPIs as objective(s) and constraints
    Set bounds for constraints: $[l_i, u_i]$, $i \in \mathcal{I}$
    Select algorithm (NNQF or NQF) and job settings
if resume optimization **then**
    Load $X = x_{pts}$ and (**f**,**g**) $= KPI_{vals}$
else Create initial points $X = x_{pts}$
end if
if number of objectives $= 1$ **then**
    Run single objective optimization (Algorithm 4.1)
else Run multiobjective optimization (Algorithm 4.4)
end if
return

Algorithm 5.1 shows the connection between the GUI and the algorithm. First, the input data given by the user in the GUI is extracted and formatted to suit the algorithm. If resuming the optimization then data is loaded and if not an initial set of points is created. Then, depending on the number of objective KPIs, the single objective or multiobjective algorithm is called. Here *run single objective optimization* and *run multiobjective optimization* are given by the pseudo code in algorithm 4.1 and 4.4 respectively. The only interaction that the code for the single objective and multiobjective optimization algorithms have with the main RUSA system and its GUI, is the function *send jobs* given in algorithm 5.2. It is a function that sends the jobs given by *x_pts* and returns the KPI values.

Algorithm 5.2: send_jobs ($x\_pts = X_0, all\_x\_pts = X, T$)

Generate Jobs from *x_pts*
Write generated jobs to file and execute in Studio
Generate all jobs from *all_x_pts* (for *Jobs* tab)
while Jobs **not** Done
    Wait T seconds
    Update Solution status in GUI
    if Some jobs are not working
        Cancel those Studio jobs
    end if
end while
Extracting KPIs from all jobs (for *KPI Table* tab)
Getting the KPI values for the chosen KPIs
Updating the GUI
return KPI values (objectives, constraints)

RUSA is a fairly complex program with many functions dealing with the connection to Studio, creating jobsets from parameters and rules etc. The old RUSA had only one thread requiring all actions to be fast in order for the GUI not to freeze. For the implementation in this project, the requirements for the GUI renders this impossible, hence multithreading had to be implemented. Due to the single threading nature of RUSA there was no support for making calls from the Application side to the GUI side, e.g. to update the GUI. This had to be done in order to fulfill the requirements of the GUI. The connections to Studio needed not be changed

but the already existing modules for that could be slightly altered to fit the new requirements. The *send jobs* algorithm in Algorithm 5.2 uses just this.

In Algorithm 5.2, jobs are generated from the new points $x\_pts = X_0$ and they are written to file and executed in Studio. RUSA already contained modules for generating jobs, writing jobs to file and executing jobs; these modules were slightly modified during the new project. Then jobs are generated from $all\_x\_pts = X$ since then they will all be shown in the *Jobs* tab. The jobs are monitored using an *apc_lock*-file and while there exists at least one unfinished job, the thread waits a prescribed time and then updates the GUI. There are cases where the jobs do not run properly or some of them end up in a long queue while others finish quickly. In order not to wait for the jobs in long queues or for the program to stop working due to non-functioning jobs, a heuristic is implemented to cancel the jobs that pose trouble.

When the jobs are done, the KPI values are extracted, showing them in the *KPI Table* after the signal has been given for the GUI to update. Then the monitored KPI objectives and KPI constraints are returned to the single/multiobjective optimization algorithm.

6 Evaluating the algorithm

In this section the single objective and multiobjective algorithms are tested on a small benchmark problem. This is done in order to make sure that the algorithms are working as planned before continuing with the Carmen crew problems. The single objective algorithm is tested in Section 6.1, the KPI constraints are tested in Section 6.2 and the multiobjective algorithm is tested in Section 6.3.

6.1 Single objective algorithm

In order to evaluate the single objective algorithm, Algorithm 4.1, it is tested on the Branin test function [Bra72]

$$f(\mathbf{x}) = \left(x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10 \tag{59}$$

which is defined on $x_1 \in [-5, 10], x_2 \in [0{,}15]$. This function has no local minima except for three global minima, $\mathbf{x}^* = (-\pi, 12.275), (\pi, 2.275), (9.425, 2.475)$ with $f(\mathbf{x}^*) = 0.398$.

The level curves for (59) are shown in Figure 15(a) and the other figures in Figure 15 are level curves for the surrogate functions. In Figures 15(b)-15(d) 10 initial points are used and 3 batches of 5 points are selected by the algorithm, a total of 25 points. The initial points are marked with stars and the subsequent points are marked by circles.

Using $\sigma = \{0, 10, 50\}$ for the three batches and using the NQF and NNQF methods yield Figures 15(b), 15(c) and for space filling only, i.e. $\sigma = \{0, 0, 0\}$ Figures 15(d) and 15(e) are given. In comparing the NQF and NNQF methods with the level curves of the exact Branin function, the NQF method captures more essential traits of the function than the NNQF method. However increasing the number of points to evaluate to 3 batches of 10, a total of 40 points, see Figure 15(f), provides results more similar to the exact solution for the NNQF method. The NQF method also provides better results when it comes to the minimum value. All three global minima are found and their values are not far away from the analytic minima. If a point close to the minima is found, then points too close to that one with possibly lower values can be hard to find due to the spatial part of the weight function. Although increasing σ reduces that problem. The NNQF method provides inferior results also when it comes to finding the lowest objective value and with the space filling method you are only lucky if a really low function value is found.

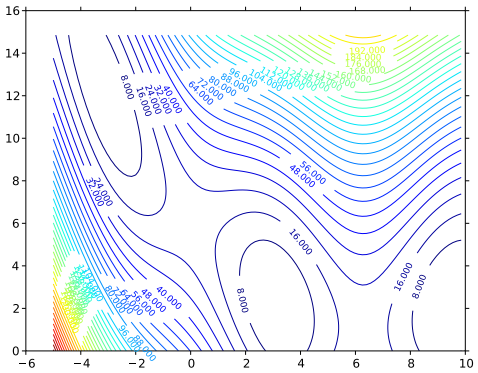
6.2 Output constraints

The output constraints part $\varphi(\mathbf{S}_g(\mathbf{y}))$ is tested on the Branin function with the constraints[6] given by

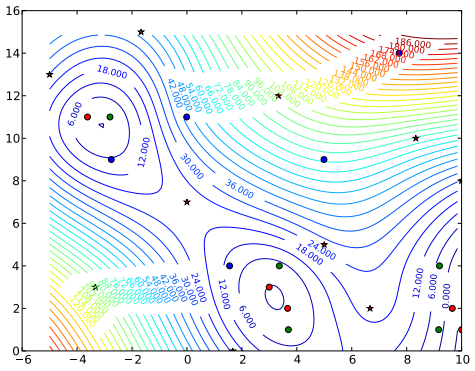
$$\begin{cases} g_1(\mathbf{x}) = x_2 - \frac{1}{2}(x_1 - 1)^2 \\ g_2(\mathbf{x}) = -x_2 - 3x_1/2 + 10 \end{cases} \tag{60}$$

and $g_1(\mathbf{x}), g_2(\mathbf{x}) \geq 0$. The analytical constraints are shown in Figure 16 but the shape of the output constraints surrogate functions are not shown. In the figure the same evaluation is done as without the constraints: 10 initial points followed by 3 batches of 5 points, a total of 25 points. As can be seen in the figure, besides the initial points all the points but one are sampled within the area $\mathcal{D}$ given by the constraints. There is one sampled outside the feasible region in the upper left corner and that is probably due to that the surrogate model $S_{g_1}(\mathbf{x})$ does not entirely mimic the shape of the output constraint $g_1(\mathbf{x})$.

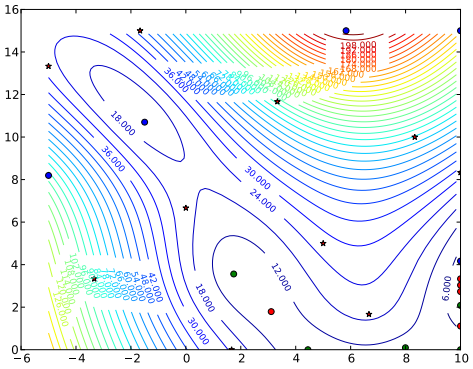
[6] The constraints are not part of any benchmark test or taken from any published reference, but chosen by the author.



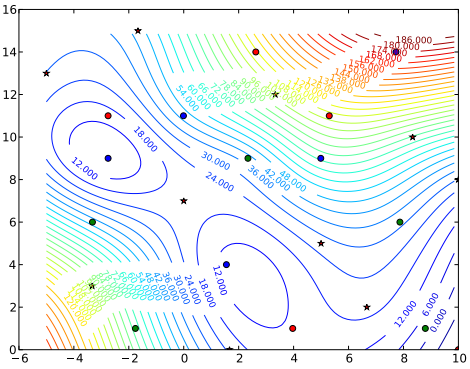
(a) The exact level curves to the Branin test function



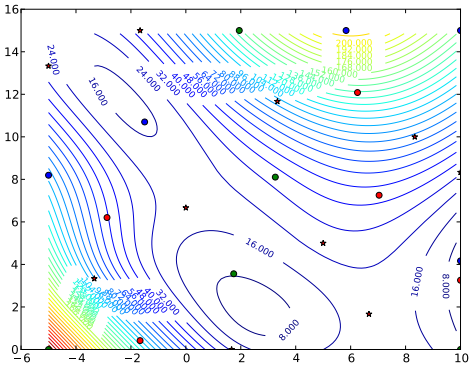
(b) Using the neighborhood (NQF) method



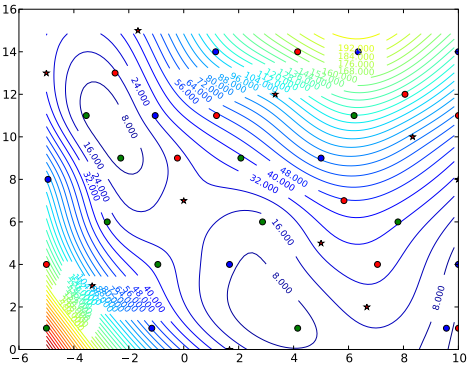
(c) Using the non-neighborhood (NNQF) method



(d) Using the NQF method with only space filling.



(e) Using the NNQF method with only space filling.



(f) Using the NNQF space filling method with a total of 40 points: 10 initial and 3 batches of 10 points.

Figure 15: The surrogate functions level curves with initial points (stars) and chosen points (circles) with 10 initial points and 3 batches of 5 points, a total of 25 points for all figures but 15(f). The parameter σ was set $\sigma = \{0, 10, 50\}$ except where space filling was used, then $\sigma = \{0, 0, 0\}$.

The best points selected give good objective values, even better than without the constraints. With the constraints, the feasible search domain has been reduced leading to a more dense set of points and hence a better approximation to the real objective function in the parts of the domain that are of interest.

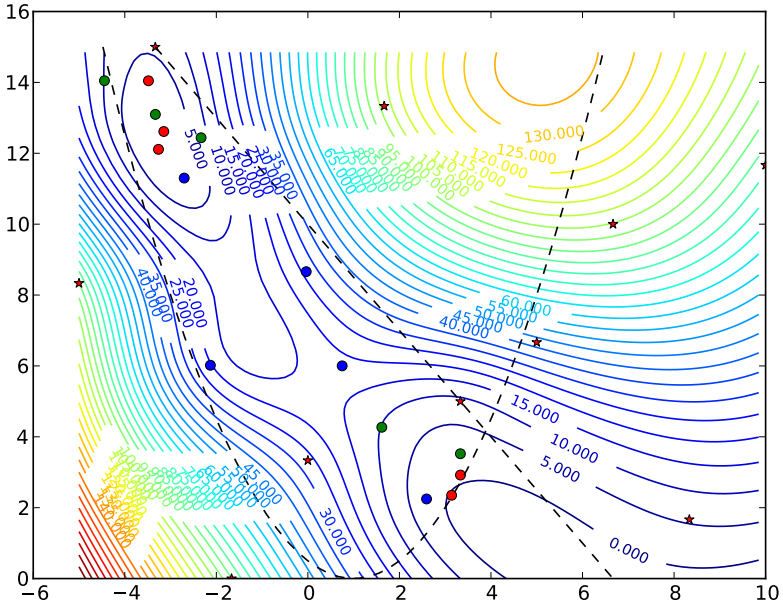
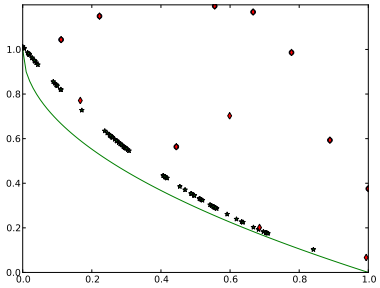


Figure 16: Running the algorithm with 10 initial points (stars) and 3 batches of 5 points (circles) using the constraints (60) given by the dotted lines.

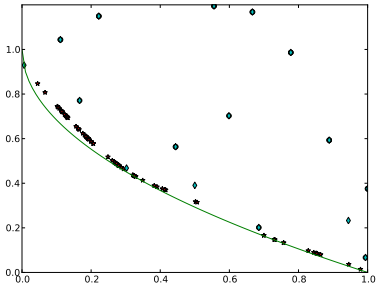
6.3 Multiobjective

To test the multiobjective algorithm, test functions ZDT1 and ZDT2 were used, see Appendix E. In their original form the parameter space has dimension 30 and domain $[0,1]^{30}$ but here the tests are performed with dimensions 3 and 5 hence domains $[0,1]^3$ and $[0,1]^5$ respectively. Function ZDT1 is convex and ZDT2 is concave as seen in Figure 17. Unless otherwise stated, 10 initial points are used followed by a number of batches N_b each containing 5 points $(10+N_b\times 5)$. In figures 17(a)-17(g) the filled line shows the analytical Pareto front and the stars show the Pareto front given by the algorithm. The diamonds shown are evaluated solutions.

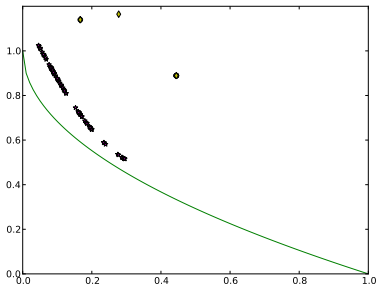
In figures 17(a)-17(d) ZDT1 is used; in the first two figures with $d=3$ and the following two with $d=5$. With $d=3$ it is sufficient with 20 points to capture large parts of the Pareto front in a good way whereas for $d=5$ that is not the case even for 75 points. In figures 17(e)-17(g) ZDT2 is used with $d=3$. It can be seen that 20 points provide a good approximation of the Pareto front and that 4 points have been sampled with objective values close to the Pareto front. In increasing the number of points to 100, as in Figure 17(g), more points are sampled with objective values close to the front.



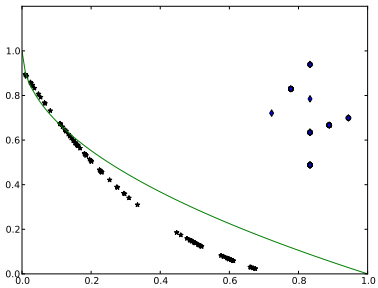
(a) ZDT1: $d = 3$, 15 points



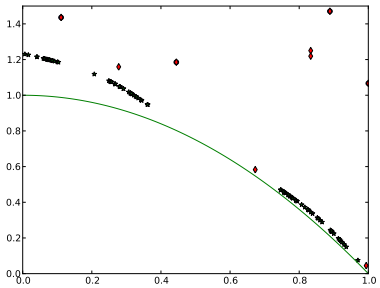
(b) ZDT1: $d = 3$, 20 points



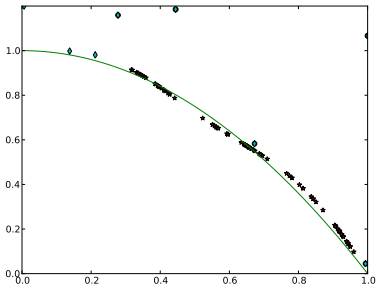
(c) ZDT1: $d = 5$, 30 points



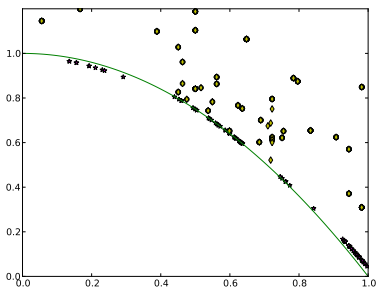
(d) ZDT1: $d = 5$, 75 points



(e) ZDT2: $d = 3$, 15 points



(f) ZDT2: $d = 3$, 20 points



(g) ZDT2: $d = 3$, 100 points

Figure 17: The multiobjective algorithm applied to the ZDT1 and ZDT2 functions with $d = 3$ and $d = 5$ parameters (dimensions) and different number of points (function evaluations). The filled line is the analytical Pareto front, the stars correspond to the Pareto front given by the algorithm and the diamonds are evaluated solutions.

7 Results: Evaluating Additions to RUSA

In evaluating the additions to the RUSA program: *Parameter Optimization* and *Prediction Model*, a few test cases are considered. In particular, for the Parameter Optimization there were three *Pairing* test cases available during this project, all of which have one objective, five parameters and five output constraints. The difference between the cases is the range of the output constraints, hence the set $\mathcal{D}$ is different. The test settings for the parameters and output constraints are presented in Appendix A and the results are presented in Section 7.1 and Appendix C.1.

For the Prediction Model, the same test case is used but with $\mathcal{D} = \mathbb{R}^5$, hence no output constraints. The results are presented in 7.2

Since the *Rostering* and *Pairing* problems are quite different in structure, RUSA is also tested for a *Rostering* optimization problem. The test description and parameters are presented in Section 7.3 together with the results for the rostering problem. The best found solutions are presented in Appendix C.2. For the rostering problem the domain consists of five parameters where three of them are Boolean and two are integer valued, with no KPI (output) constraints present.

7.1 RUSA automated optimization

The following Pairing tests consist of one objective KPI and five KPI constraints as seen in Appendix A. However, in Section 7.1.1 all, or some, KPI constraints are relaxed in order to analyze how the solver handles the structure of the problem as well as for reference. In Section 7.1.2 the results for the three main test cases, seen in Appendix A, are presented and lastly, in Section 7.1.3, multiobjectivity is tested.

The three main test cases are denoted $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ respectively and if more than one test is run for a particular test case, then those are enumerated by super scripts ($\mathcal{T}_1^1$, $\mathcal{T}_1^2$). The relaxed problems are denoted $\mathcal{T}_{0.1}$, $\mathcal{T}_{0.2}$.

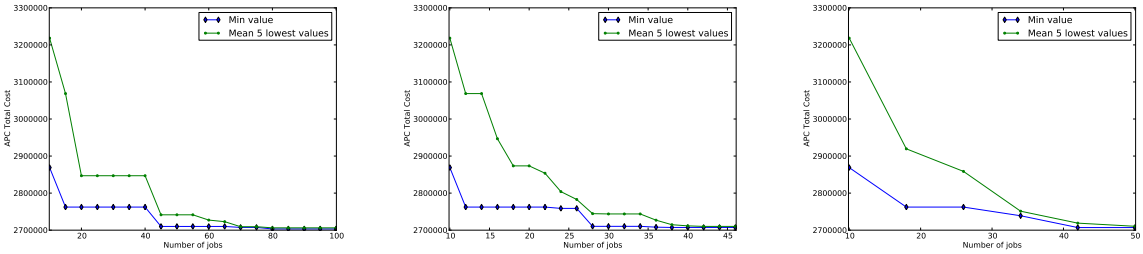
Unless stated otherwise, the results are produced using 10 initial points ($n_{init} = 10$) followed by 18 batches of 5 points ($N_b = 18$, $N_p = 5$) which is abbreviated $10 + 18 \times 5$. Also, the setting used for σ is the one given in Section 4.2.1, hence for 10 runs the following is used (0,0,5,5,5,10,10,20,20,50) unless stated otherwise.

7.1.1 Single objective: Relaxing some KPI constraints

Two tests are performed with relaxed KPI constraints. First, for reference, no KPI constraints are used to get a feeling of what the best possible solution is.

$\mathcal{T}_{0.1}$ All KPI Constraints Relaxed.

With no KPI constraints there are no further restrictions on the domain Ω except for integrality in some dimensions. Comparing the results for $\mathcal{T}_{0.1}$ in Table 2 with the three test cases in Table 6 it is notable that if this test was done for test cases 1 and 2 then all KPI values except for *aircraft changes* would have been within the bounds $\mathcal{D}$. However, *aircraft changes* is exceeded by quite a large amount and all the solutions with low *APC Total Rule Cost* have a high value for *aircraft changes* as seen in Figure 25(a). Also a large *duty time per working day* is better for the *APC Total Rule Cost* as seen in Figure 25(b). These two figures have correlations very close to -1 hence comparing *duty time per working day* and *aircraft changes* as done in Figure 25(c) they are highly positively correlated. Since there is a lower bound on *duty time per working day* (for Cases 1,2) and an upper bound on *aircraft changes*, these are conflicting and hence there must be a trade-off. With a high correlation, the dependencies are easier to deal with if they



(a) 10 initial runs and 18 batches of 5 runs.

(b) 10 initial runs and 18 batches of 2 runs.

(c) 10 initial runs and 5 batches of 8 runs.

Figure 18: The development of the minimum value of *APC Total Rule Cost* as a function of the number of jobs. Also the mean of the five lowest values is presented.

would have been investigated by hand. Although, looking at how *APC Total Rule Cost* depends on *Number of 4-day trips* in Figure 25(d) the dependence is more complicated. Also *Number of 4-day trips* and *aircraft changes* have a non-trivial dependence, see Figure 25(e). In reducing the upper bound of both these KPIs, which is done in Test Case 2 and 3, there are very few of the parameter settings in test $\mathcal{T}_{0.1}$ that will be feasible, as seen in Figure 25(e).

Table 2: Consider the two relaxed optimization problems $\mathcal{T}_{0.1}$ and $\mathcal{T}_{0.2}$. For each of these two optimization problems, the table shows KPI values for the best feasible solutions found.

KPI	$\mathcal{T}_{0.1}$	$\mathcal{T}_{0.2}$
APC total rule cost	2 703 312	2 710 072
average block time per working day	4:24	4:24
deadhead time	44:35	46:10
aircraft change	308	299
duty time p.w.d	8:01	8:02
Number of 4-day trips	75	77

For the results in Table 2, 10 initial points have been used followed by 18 batches of 5 points. It is of interest how fast the lowest value is found and this is done for test $\mathcal{T}_{0.1}$ with three different settings: 10 initial points and then 18 batches of 5 points ($10 + 18 \times 5$) as well as $10 + 18 \times 2$ and $10 + 5 \times 8$ points. Figure 18 shows the results. The minimum objective function value is shown together with the mean of the five lowest objective function values. In Figure 18(a), with 18 batches of 5 jobs, the achieved minimum objective function value (2 703 312) is reached after 70 jobs and when σ increases, the mean of the five lowest objective function values decreases and is very close to the minimum value. In figure 18(b) with 18 batches of 2 jobs the achieved minimum objective function value (2 707 299) is reached after 28 jobs. Also here the mean is catching up with the minimum objective function value when σ is increased. In figure 18(c), with 5 batches of 8 jobs, the achieved minimum objective function value (2 707 051) is reached after 42 jobs.

The NNQF method was used to produce the results in Table 2. The reason for that is discussed near the end of Section 7.1.2. The parameter settings for those results can be found in Appendix C.1. Since *aircraft changes* is the main bottleneck, at least for Test Cases 1 and 2, test $\mathcal{T}_{0.2}$ is run.

$\mathcal{T}_{0.2}$ Relaxing the KPI Constraint *aircraft changes* for Test Case 1

As seen in Table 2, the feasible solution to the relaxed problem $\mathcal{T}_{0.2}$ with the lowest objective value, is 0.25% higher than the feasible solution to the entirely relaxed problem $\mathcal{T}_{0.1}$ with the lowest objective value. There is also a solution whose objective value is 0.05% lower than the best found in $\mathcal{T}_{0.1}$, however for that, *deadhead time* is above the upper bound, 50:00. There are many jobs run with *deadhead time* close to 50:00, both above and below, which could be attributed to the error in the surrogate modeling. So, limiting the domain through KPI constraints that do not impose much constraint on the best parts of the domain, can actually help guiding the solver towards better minima. This was also seen when evaluating the algorithm in Section 6.2.

7.1.2 Single objective: Using all KPI constraints

With the relaxed tests in Section 7.1.1 as reference and with some knowledge about the trade-offs in the problem, the three main test cases are run:

$\mathcal{T}_1$ Test Case 1

For test case 1, two tests ($\mathcal{T}_1^1$, $\mathcal{T}_1^2$) are performed, for reasons that will be explained below. Table 3 shows that test $\mathcal{T}_1^1$ gives a best objective value that is 2.2% higher than the best found in the relaxed test $\mathcal{T}_{0.2}$. As a reference, the best obtained objective value is also 2.2% higher than the best for $\mathcal{T}_{0.1}$.

The toughest constraint, *aircraft changes*, is not fulfilled since there are 25 aircraft changes too many but it is a lot closer to feasibility than before. Not a single solution is entirely feasible when it comes to KPI constraints although some are very close. When running an additional 20 batches of 5 jobs, several feasible solutions arise and the best one is $\mathcal{T}_1^2$ which has an objective value that is 4.1% larger than that in $\mathcal{T}_{0.1}$. With 10 initial runs followed by 5 batches of 8 jobs the best feasible solution found has an objective value that is 2.3% higher than the objective value for the best feasible solution found for test $\mathcal{T}_1$.

$\mathcal{T}_2$ Test Case 2

In test $\mathcal{T}_2$ the same solution as in $\mathcal{T}_1^1$ is found although it also finds a feasible solution which is the one presented in Table 3. The objective value for this solution is 6.2% higher than the best for the entirely relaxed problem $\mathcal{T}_{0.1}$. The KPI *aircraft changes* is in this case the constraint that puts most restriction on the feasible domain close to the optimum. With 10 initial runs followed by 5 batches of 8 jobs the best feasible solution found has an objective value that is 0.9% higher than the objective value for the best feasible solution found for test $\mathcal{T}_2$.

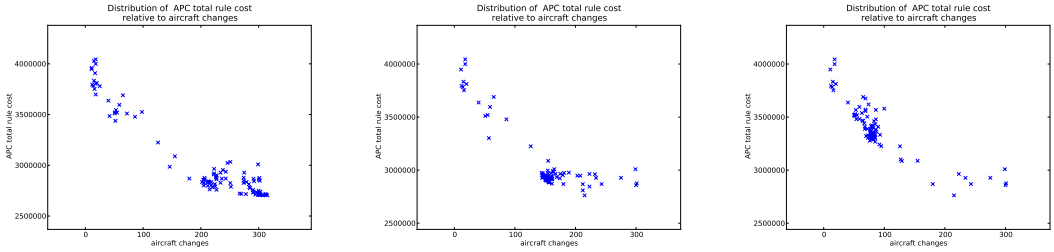
$\mathcal{T}_3$ Test Case 3

For test $\mathcal{T}_3$ the best feasible solution found is presented in Table 3 and the corresponding objective value is 19.4% higher than the best objective value found for $\mathcal{T}_{0.1}$. It is notable that 70% of the sampled points are feasible where most non-feasible points are sampled when $\sigma_c = 0$, hence when the KPI constraints are not used. The KPI constraint *aircraft changes* is active but also *Number of 4-day trips* is close to its bound. With 10 initial runs followed by 5 batches of 8 jobs, the best feasible objective value found was 2.2% higher than the objective value for the best feasible solution found for test $\mathcal{T}_3$.

In Figure 19 the *aircraft changes* vs *APC Total Rule Cost* is plotted for tests $\mathcal{T}_{0.1}$, $\mathcal{T}_1$ and $\mathcal{T}_3$. From having a large emphasis on the interval 200-300 in Figure 19(a) the plot has been shifted to the left as the upper limit on *aircraft changes* is lowered in the subsequent figures. In

Table 3: KPI values for tests $\mathcal{T}_1^1$, $\mathcal{T}_1^2$, $\mathcal{T}_2$ and $\mathcal{T}_3$ for the *Automated Optimization* where $\mathcal{T}_1^1$ and $\mathcal{T}_1^2$ are two tests instances for test $\mathcal{T}_1$.

KPI	$\mathcal{T}_1^1$	$\mathcal{T}_1^2$	$\mathcal{T}_2$	$\mathcal{T}_3$
APC total rule cost	2 762 307	2 812 856	2 861 166	3 226 121
average block time p.w.d	4:14	4:06	4:02	3:27
deadhead time	44:40	40:30	44:00	40:15
aircraft change	215	180	164	95
duty time p.w.d	7:45	7:31	7:25	6:22
Number of 4-day trips	68	74	64	27



(a) *APC Total Rule Cost* vs *aircraft changes* for $\mathcal{T}_{0.1}$ (b) *APC Total Rule Cost* vs *aircraft changes* for $\mathcal{T}_1$. (c) *APC Total Rule Cost* vs *aircraft changes* for $\mathcal{T}_3$.

Figure 19: *APC Total Rule Cost* vs *aircraft changes* for no KPI constraints in a), Test Case 1 in b) and Test Case 3 in c).

Figure 19(b), for $\mathcal{T}_1^1$, the upper limit on the KPI constraint in question is 190 and the interval which is sampled most frequently is 150-220 and in particular 150-180. In Figure 19(c), for $\mathcal{T}_3$, the upper limit on the KPI constraint is 100 and a clear shift to the left can be noted with the most prominent region being 60-90.

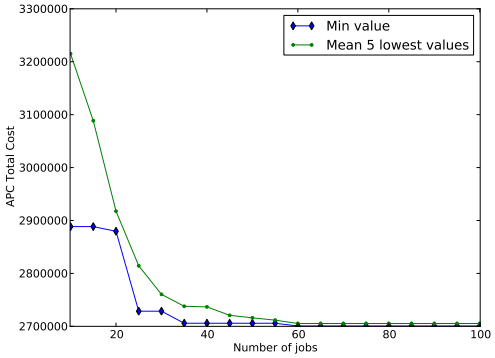
As noted earlier these results have been produced using the NNQF method. This is due to that fact that the NQF has produced inferior results for these test cases. In fact the results with the NQF were in average 2% higher than those using the NNQF method. In some cases they were the same but never better.

Changing the values of σ in a different way e.g. cycling using the vector (0,5,10,10,10,20, 20,20,50,50) instead of (0,0,5,5,5,10,10,20,20,50) provided a new best found solution with objective value 2 698 552 for test $\mathcal{T}_{0.1}$ but for other tests the results were slightly worse. The development of the minimum value as a function of the number of evaluations is seen in Figure 20(a) where the rapid convergence should be noted.

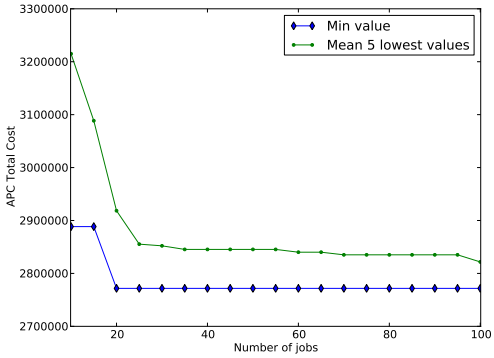
A space filling test was also performed with $\sigma = (0,0,0,0,0,0,0,0,0,0)$. The best objective value was more than 2.5% higher (2 771 731) than the best found and the following four best solutions are significantly higher. In Figure 20(b) the difference between the minimum value and the mean of the five lowest values shows just this.

7.1.3 Multiobjective optimization

The multiobjective algorithm is tested on the same main problem as earlier. From the figures in Figure 25 the most interesting multiobjective cases, amongst those combinations of KPIs shown in the figures, are Figures 25(d) and 25(e). The other plots are too highly correlated. Using *Number of 4-day trips* and *aircraft changes* as objectives the results can be seen in Figure 21.



(a) Test with a different σ cycling.



(b) Space filling test.

Figure 20: The development of the minimum value of *APC Total Rule Cost* as a function of the number of jobs. Also the mean of the five lowest obtained values is presented. Both (*a*) and (*b*) correspond to test $\mathcal{T}_{0.1}$ which has no KPI constraints.

The reference case with *APC Total Rule cost* in 21(a) (the same as 25(e)) can be compared to the multiobjective case in Figure 21(b). The multiobjective plot captures what seems to be the Pareto front in a good way although it is not very well spread out. In Figures 21(c) and 21(d) the same is done for the KPIs *Number of 4-day trips* and *APC Total Rule cost*. The algorithm focused solely on one part of the Pareto front namely in the lower left corner.

7.2 RUSA prediction model

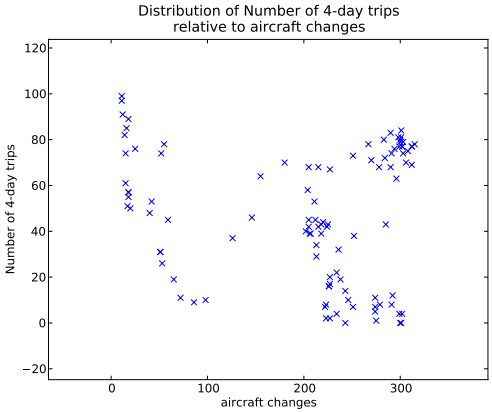
The prediction model aims at, as accurately as possible predict the behavior of KPIs for any given combination of parameters. Therefore the error in the model is observed: comparing the surrogate and the real function, as the algorithm progresses. The KPI for which the error is calculated is *APC Total Rule Cost*. The error $e(\mathbf{x})$ is calculated as the difference between the value of an evaluated point and what the prediction model predicted beforehand. It is also normalized in some way according to one of $e_1(\mathbf{x})$ and $e_2(\mathbf{x})$ given in (61) where f_{min} and f_{max} are the minimum and maximum of the evaluated objective values respectively.

$$e_1(\mathbf{x}) = \frac{|S(\mathbf{x}) - f(\mathbf{x})|}{f(\mathbf{x})}, \quad e_2(\mathbf{x}) = \frac{|S(\mathbf{x}) - f(\mathbf{x})|}{f_{max} - f_{min}} \tag{61}$$

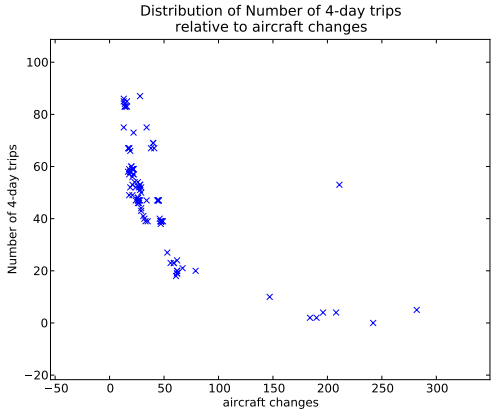
Recall from Algorithm 4.1 the definition of X^0 as the set of points constituting a batch of jobs to be run simultaneously. Running N_p jobs simultaneously ($X^0 = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_p}\}$) and calculating the error for each point as described earlier, then for each batch of jobs, the vector of errors is given by $\mathbf{e}_X = (e_k(\mathbf{x}_1), e_k(\mathbf{x}_2), \ldots, e_k(\mathbf{x}_{N_p}))$ for some $k \in \{1,2\}$.

As a measure of the total error, the norms $\|\mathbf{e}_X\|_t$ are used with $t \in \{1,2,\infty\}$. These norms are calculated for each batch of N_p jobs and plotted against number of iterations as a measure of how the error in the model changes. With no KPI constraints, the error, as described above, is shown in Figure 22 where also the trends are shown by the dashed lines.

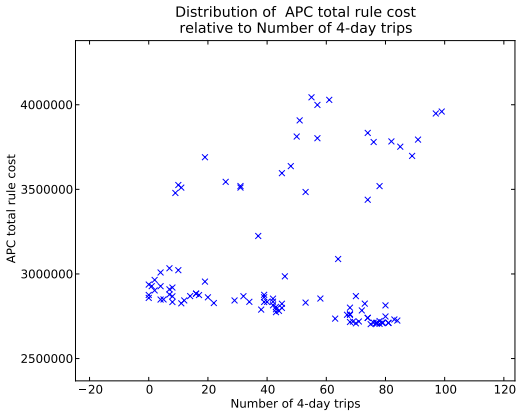
A setting for the prediction model is *space filling* and with that setting the plots in Figure 26(a) in Appendix D are created. Both measures of error are decreasing as the number of jobs increase. The error for the non-space filling method is slightly lower than for the space filling method when comparing Figures 22 and 26(a) although the error varies between batch. These tests are resumed and used in Figures 26(b) and 26(c) to investigate the error if adding 10 additional batches of 5 points in a space filling manner. In Figure 26(b), the space filling test in



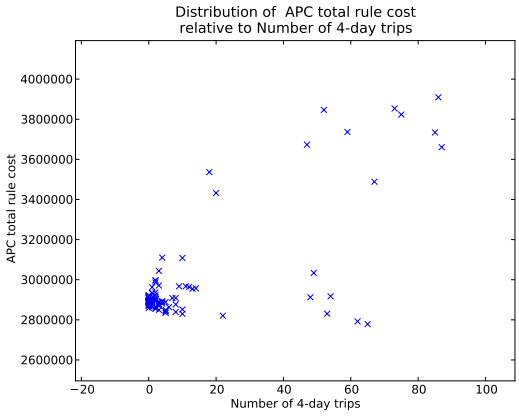
(a) *Number of 4-day trips* vs *aircraft changes* with *APC Total Rule cost* as objective.



(b) Multiobjective case with *number of 4-day trips* and *aircraft changes* as objectives.



(c) *Number of 4-day trips* vs *APT Total Rule cost* with the latter as objective.



(d) Multiobjective case with *number of 4-day trips* and *APC Total Rule cost* as objectives.

Figure 21: Comparing the KPIs *Number of 4-day trips* and *aircraft changes* in the case of no KPI constraints and *APC Total Rule cost* as objective in (a) and the multiobjective case with the two KPIs as objectives in (b). Plots (c) and (d) are created in analogy with the two previous plots but for KPIs *Number of 4-day trips* and *APC Total Rule cost*.

Figure 26(a) has been resumed, and it is clear that the trend is a continuing decrease in error. In Figure 26(c), which is a continuation of Figure 22, it is apparent that the error increases by a substantial amount, at least initially when adding the extra points.

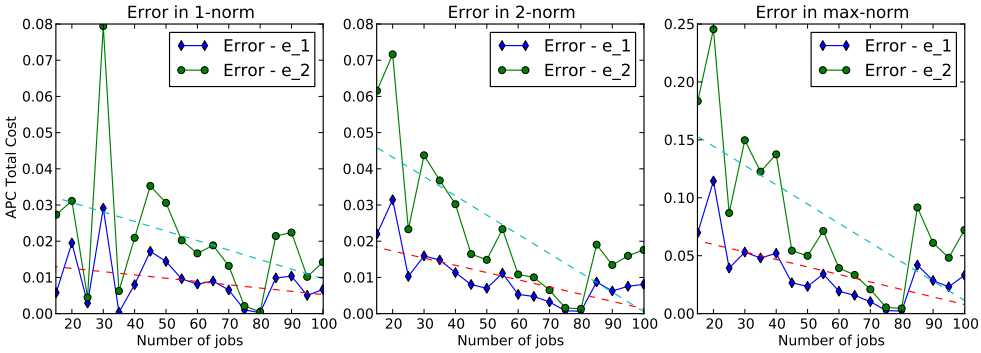


Figure 22: For no KPI constraints, hence test $\mathcal{T}_{0.1}$: the measures of error e_1 and e_2 given in (61) for 10 initial points followed by 18 batches of 5 points. The dashed lines show the linear regression trends.

7.3 RUSA: Rostering test case

The rostering test case used in this thesis work was set up in such a way that only five of all the algorithm parameters were allowed to change values; these five parameters are presented in Table 4 together with their ranges. The parameters are anonymous and named *Rostering_parameter1-Rostering_parameter5*. All five parameters are integer valued where three are Boolean and one of the other two has a range that is large enough to allow for that variable to be treated as a continuous variable. There are no KPI constraints and the goal is to find the solution with the lowest KPI *Cost*.

Table 4: Parameters used and their ranges

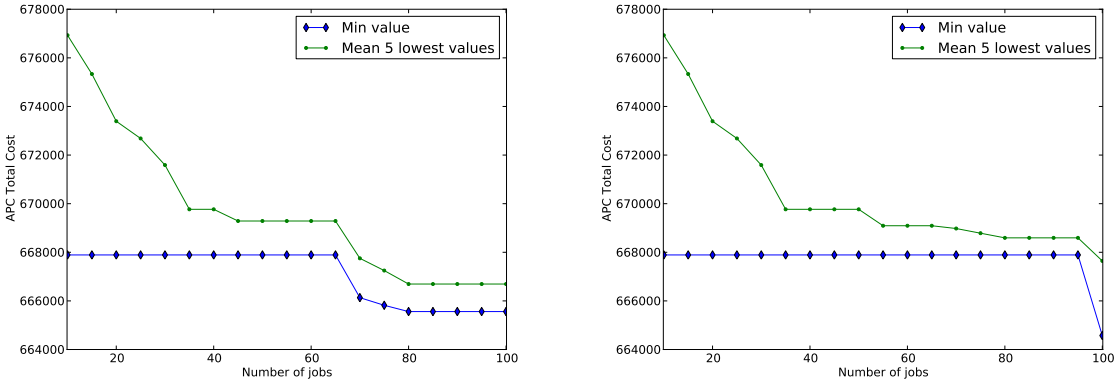
Parameter	Min	Max	Type
Rostering_parameter1	0	100	Int
Rostering_parameter2	False	True	Bool
Rostering_parameter3	False	True	Bool
Rostering_parameter4	False	True	Bool
Rostering_parameter5	1	5	Int

The jobs took 40-100 minutes to complete and the problem was larger than the down-scaled pairing problem discussed earlier. The lowest objective value found was 664574 and it was found both with the usual σ-cycling method and the space filling method. However, for the space filling method it is found in the last batch as can be seen in Figure 23. The parameter settings for these solutions can be found in Appendix C.2. The solutions differ in the value for *Rostering_parameter5*, however, the parameter *Rostering_parameter4* has been set to "False" causing *Rostering_parameter5* to have no effect on the results.

The best solution found can be compared to the results using the standard settings for the matador script: the so called matador standard script. The cost corresponding to the matador standard is 693878 which is 4.4% higher than the best found using RUSA. In Figure 27 in Appendix F the progress of the cost during the optimization can be seen for the matador

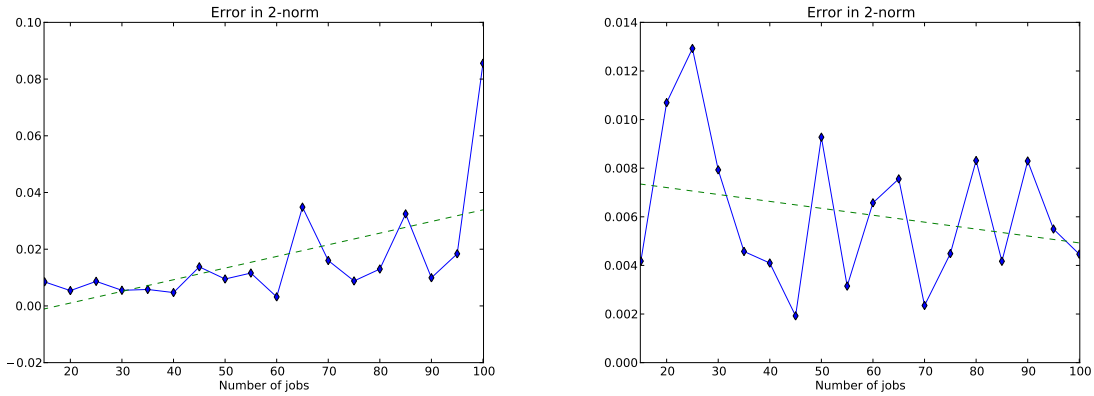
standard as well as for the best found solution. The matador standard script finished after 60 minutes and hence it was run twice in order to be able to make a fair comparison with the best found solution using RUSA, which ran for 110 minutes.

The error for these runs is presented in Figure 24 together with the linear regression trend. For the space filling, the error decreases whereas it increases for the non-space filling test. It should be noted that the rostering optimizer is based on a local search principle and is quite heuristic in nature. The objective value obtained can often vary quite a lot depending on small changes in input for the optimizer. Examples of such small changes are a small change for the value of an algorithm parameter or just a change of the random seed used for the random generator that is used in the rostering optimizer. This is further discussed at the end of Section 8.



(a) The usual σ-cycling. (b) Space filling.

Figure 23: The development of the minimum value of *Cost* for the rostering test case as a function of the number of jobs where also the mean of the five lowest values is presented. There are 10 initial runs followed by 18 batches of 5 runs.



(a) The usual σ-cycling. (b) Space filling.

Figure 24: The error $e_1(\mathbf{x})$ as described in (61) for each batch of jobs for the rostering test case, as a function of the number of jobs. There are five jobs in each batch

8 Discussion and future research

The goal of this project was to create and implement a functioning parameter optimization algorithm, as well as a prediction model tool, into a GUI to be used on the pairing and rostering optimizers at Jeppesen Systems. The requirements in particular have been discussed earlier and concern among other things KPI constraints, integrality constraints and several objectives. The RUSA GUI that was used, and to which the *Automated Optimization* and *Prediction Model* parts were added, was written in Python. The necessary additions needed to be very entangled into the already existing code hence it was decided that the project should be written in Python.

This section will discuss the results presented in the previous section and the algorithm in general, namely the algorithm construction in Section 8.1, the algorithm results in Section 8.2 and the RUSA results in Section 8.3.

8.1 Algorithm construction

The Quality function was chosen for its flexibility, especially when it comes to extending to the multiobjective case. The cycling between global and local search is easily done using a parameter σ which is also a measure of trust that can be placed in the surrogate model. This is also done in most other methods but in different ways. Increasing σ too fast can lead to premature convergence with too much focus on local search whereas increasing it too slow can cause the algorithm to focus extensively on global search.

In order to deal with the output constraints, a multiplicative penalty function was chosen instead of a more traditional additive penalty function as in (8). This decision removed the choice of penalty parameters but instead introduced σ_c to account for the trust that can be placed in the KPI constraint surrogate. By implementing the constraints in this way, a constraint cannot be penalized individually but all constraints are penalized by the same amount, which could be seen as a draw-back. Individual values for σ_c for each constraint could be implemented, depending on some measure of trust that can be placed in each surrogate, hence making it more adaptive. Another method for adaptivity would be to place more points where the function varies a lot. It could be implemented in a function similar to $\varphi(\cdot)$ and depend on $S(\mathbf{x})$ and X. There are adaptive methods whose adaptive features perhaps could be fitted to this problem e.g. the ARBF algorithm [HQE08, Qut09].

Not allowing connections to TOMLAB, which is a MATLAB module that has a variety of global optimization solvers and even some black-box solvers, due to licensing reasons rendered the necessity for some other stand alone Global Optimization Solver. The simplest solution was to code one in Python and the DIRECT method was fairly simple to understand an implement. The performance of this global optimization solver is therefore not as good as for commercial solvers but it proved sufficiently good for some initial tests. The problem with the whole quality function method combined with the DIRECT solver is that when increasing σ the weight function $\omega(S(\mathbf{x}))$ gets increasingly narrow around the surrogates minima. This would be a problem for any Global optimization solver and in particular this one. When finding the $\mathbf{x}$ corresponding to the minimum of the surrogate it is therefore better to minimize $S(\mathbf{x})$ than $\omega(S(\mathbf{x}))$ with $\sigma = \infty$.

In incorporating the integrality constraints into the algorithm it would have been favorable to pass them along to a Global optimization solver that handles integrality constraints (as well as non-linear constraints). However, the algorithm at hand was the DIRECT algorithm which has no such features and hence a penalty function was introduced to push the points in the right direction, followed by rounding to the nearest integer. The problem with an overly oscillating function to maximize is eminent hence caution has to be taken. For parameters which are integer valued but have a large range in values they are seen as continuous. This is done in order to help the DIRECT solver deal with fewer oscillations and since in most cases that continuous

relaxation is a good approximation.

8.2 Algorithm evaluation

In testing only the algorithm on the Branin function, as in Section 6.1, the results were good. The minimum objective values were close to the analytical one and points were sampled close to all three global minima of the function. This test was primarily made to make sure that the algorithm was working as expected, not for comparison. However, comparing with the results given in [JPRW09] their results gave a faster convergence when it comes to the number of points sampled and a higher accuracy. This may to some extent be attributed to the DIRECT solver used but also that in this project several points are chosen to be processed in parallel which does not allow for the surrogate model to gradually converge. However, running several parallel jobs, the same amount of time will allow considerably more data to be collected, therefore the tests are not really comparable. The NQF provided better results than the NNQF as can be seen in Figure 15, however increasing the number of points resulted in a decreased difference.

The output constraints were added in Section 6.2 and they reduced the feasible domain very efficiently making the algorithm sample points only in, or close to the boundary of the feasible domain. The idea of the output constraints function seems to be working well, although the constraints were not very complicated.

The multiobjective case was not the main use case in this project and the scope was limited to two objectives. The algorithm was tested on test functions as seen in Section 6.3. For the lower dimensional cases, but still two objectives, the algorithm provided results that were close to the analytical Pareto front. For higher parameter dimensions the algorithm had a harder time capturing the Pareto front. Since TOMLAB was not available, a Python MOEA solver was used to find the Pareto front of any given function. The solver worked well as seen in figure 12(a) but it is not as good as commercial solvers such as those in the TOMLAB package.

When dealing with the task of finding or approximating the Pareto front, hence multiobjective optimization, it is usually of interest to find a well spread out set of solutions along the Pareto front. This is not directly considered in the algorithm but is considered indirectly through $U_X(\mathbf{x})$ if the Pareto optimal set of points is clustered in Ω. Consider a case where the Pareto front is not approximated as continuous, even though it is. Instead the Pareto front is approximated as disjoint clusters; this is the case in most figures in Figure 17. In such a situation there is a higher probability that points with objective values close to the non-continuous Pareto front are picked by the algorithm. That does in general not lead to a well spread out set of Pareto optimal points. Making the Pareto front more dense and extending it in between points fixes this to some extent but in e.g. Figure 17(c) it does not. Also, extending it upwards and to the right is a good idea in principle although as can be seen in Figure 17(c), it does not always work. In that case, extending it along the tangent of the Pareto front would have proven more effective.

8.3 Evaluation of additions to RUSA

In order to evaluate the additions to RUSA, several tests were considered where the Pairing tests are discussed first. The unconstrained case was assumed to provide the best result when it comes to the KPI objective *APC*[7] *Total Rule cost*, at least mathematically. However, adding constraints that do not impose restrictions on the parts of the domain where the best solutions exist seem to help the solver towards better solutions. This was also seen when testing on the Branin function. The addition of non-active constraints only caused the objective value corresponding to the best solution to improve by a fraction of a percent. The seed for the

[7] APC=Automatic Paring Construction

random number generator used when selecting the initial set of points also had an effect on the result that was of the same, or slightly lower, magnitude.

For these runs 10 initial points followed by 18 batches of 5 points was used ($10+18\times5$). This may be too many to use in some cases hence it is of interest to see how the algorithm performs with a different number of points. In Figure 18 the development of the minimum objective value is shown, depending on the number of jobs: $10 + 18 \times 5$ as well as $10 + 18 \times 2$ and $10 + 5 \times 8$. All three provide similar results and the first one which has twice the amount of points is only a fraction of a percent better than the other two. The fraction of global to local search is the same for the three different sampling settings hence with more batches the global search is conducted for more points. Therefore a solution with 50 points can not be compared with the value after 50 points for a run with 100 points: the local search has simply not started yet. It seems that fewer points do not provide much worse results and also small batches give a faster convergence when it comes to the number of points. However, in using smaller batches for a fixed number of points, the whole algorithm run takes longer to complete.

The KPI constraint that appears to be the constraint that puts most restrictions on the feasible region close to the optimum was *aircraft changes*. That KPI constraint is active[8], or almost active, in all three major test cases seen in Appendix A and in each test case the upper bound for *aircraft changes* is lowered. Judging from Figure 25(a), this fact should apriori mean that the *APC Total Rule cost* gets higher and the results confirm that hypothesis. From Figure 19 it is clear that the KPI constraints function is working as it should with many sampled points with the KPI *aircraft changes* below its upper bound. When using $10 + 5 \times 8$ points the results were $1 - 2\%$ worse than when using $10 + 18 \times 5$ points which probably stems from the fact that the solver does not have time to sufficiently refine the surface in order to accurately capture the parts of the domain where the good solutions are. With no KPI constraints, the way the sampling was done did not have an equally large effect on the results, as mentioned earlier.

It seems as if the algorithm finds a KPI value of 156 where 190 is the maximum limit (as in $\mathcal{T}_1$) then many other points will be sampled with KPI value around or equal to 156. Finding something closer to 190 has been proven difficult. It may be the surrogate model that oscillates or the fact that there are not enough sampled points where the result would show a value of 190 and so the surrogate falsely predicts that the value is too high and hence penalizes that region. There will always be an uncertainty in the model and hence using the KPI constraints it can never be ensured that no feasible points are penalized. In cycling σ_c this is taken into account but the problem is still there.

Using a different cycling scheme for the weight function, hence changing σ in a different way enabled the solver to find a slightly better solution although the same discussion as earlier concerning the seed is also true for this case. Also, the setting for σ that provided better results for the case with no KPI constraints did not do so in the other test cases; then the results were instead slightly higher which could be attributed to too rapid convergence in the wrong areas. The KPI constraints obviously make the problem significantly harder than the box-constrained case with no KPI constraints. An improvement for the GUI and the algorithm would be to allow the user the option of changing the way that σ is cycled. Also, the current construction does not allow for a KPI to be both a constraint and an objective, something that may be of use. That is however not too hard to implement.

As noted on earlier, the NNQF provided superior results in comparison with the NQF for the RUSA cases but for the Branin tests it was the other way around. In addition, the NQF requires more CPU time since it involves numerical integration and solving LP problems, making it even less favorable. The most plausible explanation for the difference in performance between the

[8] An active constraint is an inequality constraint that holds with equality (at a point). The set of all active constraints generally includes equality constraints too.

Branin and RUSA cases is the fact that the Branin test function is a smooth function whereas the Carmen test cases have a few complicating features. One of those is that the Studio jobs, hence the function evaluations, do not run until optimality is reached. As a consequence, sometimes a slight change in parameter values might trigger a better solution to be found. This is a larger factor for Rostering than for Pairing but it can still have an impact on these results, e.g. for an integer valued parameter with range 20000, a change from 5 to 6, or 5 to 4, creates a change in output value (*APC Total Rule cost*) that cannot be seen as smooth. This causes the surrogate to oscillate in a non-preferable way. To account for this, the interpolation requirements could have been relaxed allowing the surrogate not to pass through each point. There are methods for that e.g. in [JPRW09]. An initial assumption was that since the Carmen crew optimizers are deterministic they do not contain noise, although judging from the results they can be seen as doing so. A relaxed interpolation requirement would create a smoother surface although since the solutions have proven discontinuous at times, essential traits of the function could be lost. This is a problem, in particular for the *Prediction Model*. Also different transformations of the objective function have been considered although none were implemented in the final RUSA program.

The Prediction Model was evaluated measuring the error in the objective KPI between the predicted value and the outcome. For the space filling search as well as the case with no KPI constraints, the error decreased. For the case of no KPI constraints the results are somewhat misleading since points may be sampled very close to each other when σ increases but for the space filling it is a good measure of the over-all error in the surrogate model. Continuing with 10 batches of 5 points in a space filling manner shows that the test that did not have space filling initially gave a large error whereas the error for the space filling test continued to decrease. This is logical and proves that space filling should be used for the Prediction Model. When using the Prediction Model tab in RUSA, the problem mentioned above with an oscillating surrogate function can be observed. However, the Prediction Model gives an estimate of what the observed KPI values are and there is always the possibility of checking the results by running the job. It would have been informative to test the performance of the Kriging Surrogate method compared with the RBF as well as other radial functions such as thin plate spline for the Carmen test cases. Since this was not assumed to vastly improve the results and there are excellent comparisons in the literature [Jon01], this was not pursued.

The multiobjective algorithm is very similar to the single objective algorithm which is also why the weight function was chosen to begin with. There are no analytical solutions to compare the results in Figure 21(b) with although in comparing with Figure 21(a) the results are promising. Though, the drawback with a non-well spread out set of points as discussed earlier can be seen in Figure 21(d) where there is only a part of the Pareto front sampled. The spatial function $U_X(\mathbf{x})$ aims at putting distance between evaluated points in the domain. In general, the spatial function does not, however, give a well spread out set of points in the co-domain of the objective functions. Therefore, something similar could be constructed for the co-domain of the objective functions and included as a penalty function.

The rostering test case was not as extensive as the Pairing test case which has been rather thoroughly discussed and analyzed above. A few things can be said about the results: most of which concern the structure of the Rostering Optimizer. The error in the space filling test decreases as would be expected although it is increasing for the non-space filling test. An explanation for this could be found analyzing the structure of the Rostering problem and Rostering optimizer. As discussed earlier, a small change in parameter values can trigger the optimizer to find a solution that differs considerably. For the Rostering optimizer this is a large issue, especially for the surrogate modeling since then the "real objective function" is highly discontinuous, much more so than in the Pairing case. This causes an oscillating surrogate function

where points are selected close to each other leading to large errors. In increasing σ, points are usually selected close to each other where the value is predicted as low and hence then the error is large. For the space filling test this does not happen to the same extent which can be reflected in the results. However, the fact that the error is larger in the non-space filling case than in the space filling case does not automatically mean that the end result will be worse. With a high σ, the algorithm samples at a point where the value is predicted low. If the prediction is wrong in such a way that a predicted value is much too low, the real value may still be very good.

Even though there are problems arising because of the structure of the rostering problem, the objective value for the best rostering solution found was considerably better than the best objective value using the matador standard settings. Changing the random seed for the Rostering optimizer gives different results, however in average the best found solution is still better than the one found using matador standard. Also, in changing the seed, the problem changes and there is no way that the best setting for one problem can be guaranteed as being good for a different problem. If there is a different seed, then there is probably another best solution and the algorithm would find solutions close to that one instead.

Matador standard runs only for 60 minutes whereas the best found parameter setting makes the optimizer run for 110 minutes. The script was therefore run twice for matador standard in order to compare the results. As can be seen in Figure 27 in Appendix F, the cost for the tuned solution was also lower than the cost for matador standard after 60 minutes. Moreover, in the interval 60-110 minutes, the tuned solution gives a larger improvement than the matador standard settings. Manual tuning of the matador standard script by experienced engineers would most definitely lower the cost when comparing to matador standard but their results would not necessarily be lower than the best solution found using RUSA. Even if the manual tuning would provide a better solution than the automated search using RUSA, that should not diminish the RUSA results since the largest advantage of the algorithm is that no manual tuning is needed. After the automatic tuning is done, hopefully giving a good solution, manual fine tuning could follow in order to see if even better neighboring solutions exist.

The additions to the RUSA GUI was not the main focus of this thesis but a lot of effort has been spent on making the additions (the *Automated Optimization* and *Prediction Model* tabs) work and making them user friendly. Besides the possible additions mentioned earlier, in particular relaxing the interpolation requirements, and a few non-essential bugs, the subjective view of the author is that there are not that many more features and fixes that the GUI would benefit from in its present state. Hopefully it will be of use and added to if necessary.

Final note from the author: In writing one of the final sentences of this thesis I noticed that in one of the last paragraphs in [JPRW09] the addition of expensive black-box constraints to the algorithm is briefly discussed.

References

[AJD06] Charles Audet and Jr J.E. Dennis. Mesh adaptive direct search algorithms for constrined optimization. *SIAM Journal of Optimization*, 17(1):188–217, 2006.

[AO06] Charles Audet and Dominique Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM Journal of Optimization*, 17(3):642–664, 2006.

[BHH87] George E.P. Box, William G. Hunter, and J. Stuart Hunter. *Statistics for Experimenters: An introduction to Design, Data Analysis, and Model Building*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 1987.

[BP76] Egon Balas and Manfred W. Padberg. Set partitioning: A survey. *SIAM Review*, 18(4):710–760, 1976.

[Bra72] F.H. Branin. A widely convergent method for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research and Development*, 16(5):504–522, September 1072.

[CL05] Carlos A. Coello Coello and Gary B. Lamont, editors. *Applications of multi-objective evolutionary algorithms*. World Scientific Publishing Co Pte Ltd, 2005.

[dHMHvD07] Dick den Hertog, Hans Melissen, Bart Husslage, and Edwin van Dam. Maximin latin hypercube design in two dimensions. *Operations Research*, 55(1):158–169, 2007.

[Dri08] Joel Driessen. The rule sensitivity analysis project. Master's thesis, Uppsala University, 2007-2008.

[Fas03] Greg Fasshauer. Handout: 603 multivariate meshfree approximation, positive definite and completely monotone functions. Depertment of Applied Mathematics, Illiniois Institute of Technology, 2003. *Available at:* http://amadeus.math.iit.edu/∼fass/603_ch2.pdf (2010-07-13).

[Fin03] D. E. Finkel. Direct optimization algorithm user guide. Technical report, North Carolina State University, Center for Research in Scientific Computation, 2003. *Available at:* http://www4.ncsu.edu/∼ctk/Finkel_Direct/DirectUserGuide_pdf.pdf (2010-07-13).

[FK90] Marshall L. Fisher and Pradeep Kedia. Optimal solutions of set covering/partitioning problems usign dual heuristics. *Management Science*, 36(6):674–688, June 1990.

[FK06] D. E. Finkel and C. T. Kelley. Additive scaling and the direct algorithm. *Journal of Global Optimization*, 36(4):597–608, December 2006.

[GK01] J.M. Gablinsky and C.T. Kelley. A locally-biased form of the direct algorithm. *Journal of Global Optimization*, 21:27–37, 2001.

[GS09] Ahmed Ghoniem and Hanif D. Sherali. Complementary column generation and bounding approaches for set partitioning formulations. *Optimization Letters*, 3(1):123–136, January 2009.

[Gus99] Tomas Gustavsson. *A heursitic approach to column generation for airline crew scheduling*. Licentiate thesis, Department of Mathematics, Chalmers Univeristy of Technology and Göteborg University, Göteborg, Sweden, 1999.

[Gus09] Tomas Gustavsson. Orbit. Svenska Operationsanalysföreningen, December 2009.

[Gut99] H.-M. Gutmann. A radial basis function method for global optimization. Technical Report DAMTP 1999/NA22, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 1999.

[Gut01] H.-M. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.

[HQE08] Kenneth Holmström, Nils-Hassan Quttineh, and Marcus M. Edvall. An adaptive radial basis algorithm (arbf) for expensive black-box mixed-integer constrained global optimization. *Optimization in Engineering*, 9:311–339, 2008.

[INF10] Informs prize, 2010. *Available at:* http://www.informs.org/Recognize-Excellence/INFORMS-Prizes-Awards/INFORMS-Prize (2010-08-20).

[Jep] Jeppesen. *Jeppesen Training for Carmen Products: Introduction to Pairing and Pairing I*.

[Jon01] Ronald R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, December 2001.

[JPRW09] S. Jakobsson, M. Patriksson, J. Rudholm, and A. Wojciechowski. A method for simulation based optimization using radial basis functions. *Optimization and Engineering*, pages 1–32, 2009.

[JSW98] D.R. Jones, M. Schonlau, and W.J. Welsh. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:445–492, 1998.

[KK04] Niklas Kohl and Stefan E. Karisch. Airline crew rostering: Problem types, modeling, and optimization. *Annals of Operations Research*, 127:223–257, 2004.

[Let09] Adam N. Letchford. Mixed-integer non-linear programming: a survey. 1st LANCS Workshop on Discrete and Non-Linear Optimisation, Southampton, February 2009. *Available at:* http://www.lancs.ac.uk/staff/letchfoa/talks/MINLP.pdf (2010-06-18).

[MBC79] M.D. McKay, R.J. Beckman, and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21:239–245, 1979.

[Mic86] Charles A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

[MMS00] A. Messac, E. Melachrinoudis, and C. P. Sukam. Aggregate objective functions and pareto frontiers: Required relationships and practical implications. *Optimization and Engineering*, 1(2):171–188, July 2000.

[PJS93] C.D. Perttunen, D.R. Jones, and B.E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Application*, 79(1):157–181, October 1993.

[Pow92] M.J.D. Powel. The theory of radial basis function approximation in 1990. In W. Light, editor, *Advances in Numerical Analysis, Volume 2: Wavelets, Subdivision Algorithms and Radial Basis Functions*, pages 105–210. Oxford University Press, 1992.

[Pow99] M.J.D. Powell. Recent research at cambridge on radial basis functions. Technical Report DAMTP 1998/NA05, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, June 1999.

[Qut09] Nils-Hassan Quttineh. *Algorithms for Costly Global Optimization*. Licentiate thesis, School of Education, Culture and Communication, Division of Applied Mathematics, Mälardalen University, Västerås, Sweden, 2009.

[Raj68] Des Raj. *Sampling Theory*. McGraw-Hill Book Company, 1968.

[RS05] Rommel G. Regis and Christine A. Shoemaker. Constrained global optimization of expensive black box functions using radial basis functions. *Journal of Global Optimization*, 31:153–171, 2005.

[RW07] Johan Rudholm and Adam Wojciechowski. A method for simulation based optimization using radial basis functions. Master's thesis, Chalmers University of Technology and Göteborg University, Göteborg, Sweden, 2007.

[SdHSV03] Erwin Stinstra, Dick den Hertog, Peter Stehouwer, and Arjen Vestjens. Constrained maximin designs for computer experiments. *Technometrics*, 45(4):340–346, Nov 2003.

[SdM01] Alexander Shapiro and Tito Homem de Mello. On rate of convergence of monte carlo approximations of stochastic programs. *SIAM Journal on Optimization*, 11(1):70–86, 2001.

[Tor97] Virginia Torczon. On the convergence of pattern search algorithms. *SIAM Journal of Optimization*, 7(1):1–25, 1997.

[VH08] Roshan Joseph V. and Ying Hung. Orthogonal-maxmin latin hypercube designs. *Statistica Sinica*, 18:171–186, 2008.

[vVL00] David A. van Veldhuizen and Gary B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2):125–147, 2000.

[ZDT00] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

A Test cases

For the evaluation of the *Parameter Optimization* part of RUSA, three pairing test cases were used. There were five parameters constituting the search space as can be seen in Table 5 together with their ranges. The type *Time* is seen as continuous and is written as *hours:minutes*.

Table 5: Parameters used and their ranges

Parameter	Min	Max	Type
_topmodule.exp_max_duty_flight_time	8:00	10:00	Time
_topmodule.exp_max_duty_time	11:00	13:00	Time
_topmodule.exp_max_duty_nr_of_ac_changes	1	3	Int
_topmodule.glc1_pen	1	20'000	Int
_tpomodule.glc2_pen	1	20'000	Int

Parameters 4 and 5 are penalty parameters and hence the Key Performance Indicator *"TOTAL COST"* which includes the penalties cannot be used as an objective. Instead *"APC* [9] *total rule cost"* is used which does not include the penalties. The range of the second penalty parameter has been adjusted to [1,9999] since the name for the job with a parameter setting given by the max parameter values would be *z_a10-00_b13-00_c3_d20000_e20000* which is 32 characters and the maximum name length for Studio jobs is 31 characters.

The KPI constraints used are presented in Table 6 together with the bounds for the different cases.

Table 6: Objective KPI as well as KPI constraints used and their ranges

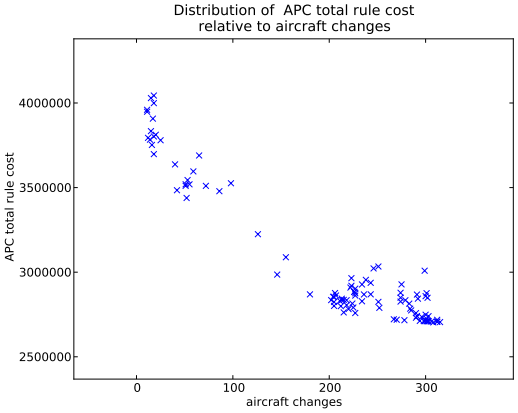
KPI	Type	Case 1	Case 2	Case 3
APC total rule cost	Obj.	- -	- -	- -
average block time per working day	Time	$\geq$ 3:50	$\geq$ 3:20	$\geq$ 3:00
deadhead time	Time	$\leq$ 50:00	$\leq$ 60:00	$\leq$ 100:00
aircraft changes	Int	$\leq$ 190	$\leq$ 170	$\leq$ 100
duty time per working day	Time	$\geq$ 7:30	$\geq$ 6:20	$\leq$ 7:00
Number of 4-day trips	Int	$\leq$ 100	$\leq$ 80	$\leq$ 30

Here "$\geq$ 3:50" means that the KPI should be equal to or larger than 3:50 and vice verse for the opposite inequality. In this case $\geq$ 3:50 means setting the lower bound to 3:50 and the upper bound to something large. It should be noted that "ac" in *_topmodule.exp_max_duty_nr_of_ac_changes* means *aircraft changes* hence that parameter is very much connected to the KPI *aircraft changes*.

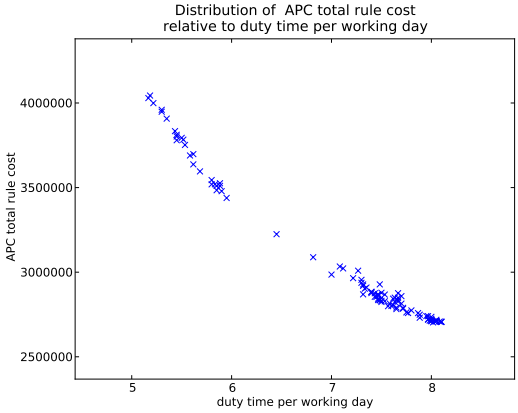
B KPI plots

The following KPI plots, Figure 25, are taken from the RUSA software for the case described in Appendix A but with no KPI constraints.

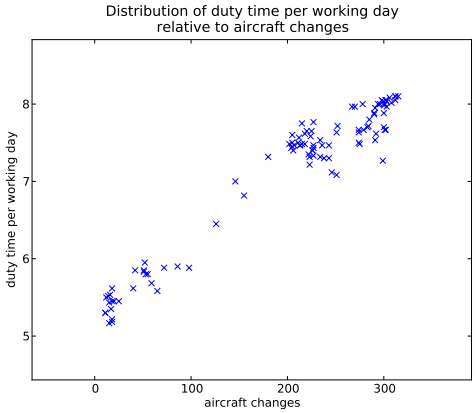
[9] APC=Automatic Pairing Construction



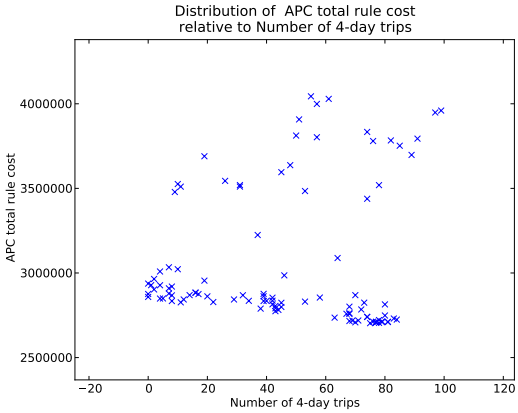
(a) *APC Total Rule Cost* vs *aircraft changes*



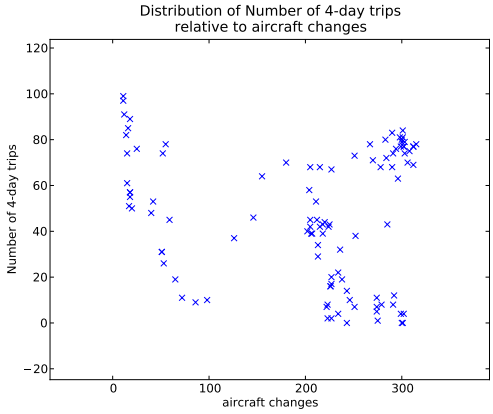
(b) *APC Total Rule Cost* vs *Duty time p.w.d.*



(c) *Duty time p.w.d* vs *aircraft changes*



(d) *APC Total Rule Cost* vs *Number of 4-day trips.*



(e) *Number of 4-day trips* vs *aircraft changes.*

Figure 25: KPI plots for test $\mathcal{T}_{0.1}$ with no KPI constraints. The figures are generated by RUSA.

C Best solutions found

C.1 Pairing

Here, the parameter settings corresponding to the solutions with the lowest objective values are presented for tests $\mathcal{T}_{0.1}$, $\mathcal{T}_{0.2}$, $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ where two tests, $\mathcal{T}_1^1$ and $\mathcal{T}_1^2$, are run for test case $\mathcal{T}_1$ with different numbers of jobs. The test perimeters for $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ are given in Appendix A and tests $\mathcal{T}_{0.1}$ and $\mathcal{T}_{0.2}$ are relaxations of $\mathcal{T}_1$. The results for the relaxed test cases are presented in Section 7.1.1 and the results for the other test cases are given in Section 7.1.2.

Table 7: Parameter values used for the solutions with the lowest objective values for tests $\mathcal{T}_{0.1} - \mathcal{T}_3$.

Parameter	$\mathcal{T}_{0.1}$	$\mathcal{T}_{0.2}$	$\mathcal{T}_1^1$	$\mathcal{T}_1^2$	$\mathcal{T}_2$	$\mathcal{T}_3$
_topmodule.exp_max_duty_flight_time	9:33	9:20	9:40	9:07	9:40	9:06
_topmodule.exp_max_duty_time	13:00	13:00	12:53	13:00	12:53	13:00
_topmodule.exp_max_duty_nr_of_ac_changes	3	3	2	3	3	3
_topmodule.glc1_pen	2	3	1112	2233	2593	4569
_tpomodule.glc2_pen	1	355	556	1	556	4465

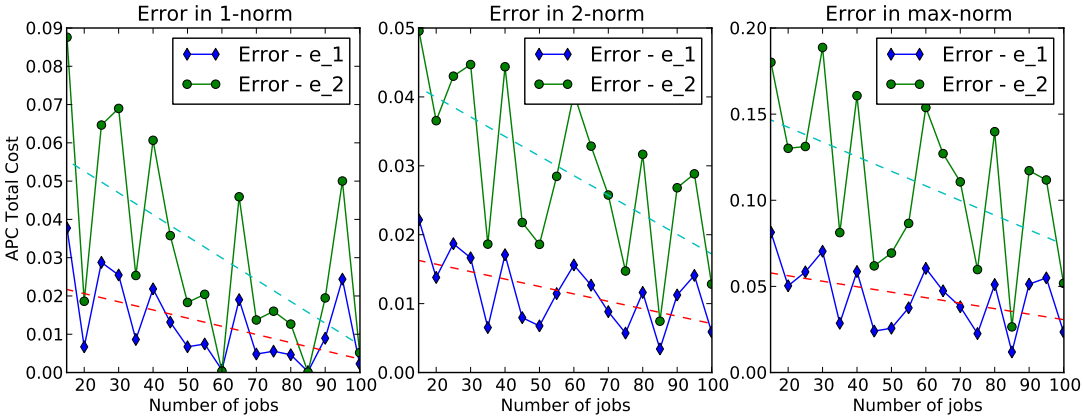
C.2 Rostering

The following parameters correspond to the best solutions found for the Rostering test case. Tests are performed usign the space filling method and the usual σ-cycling.

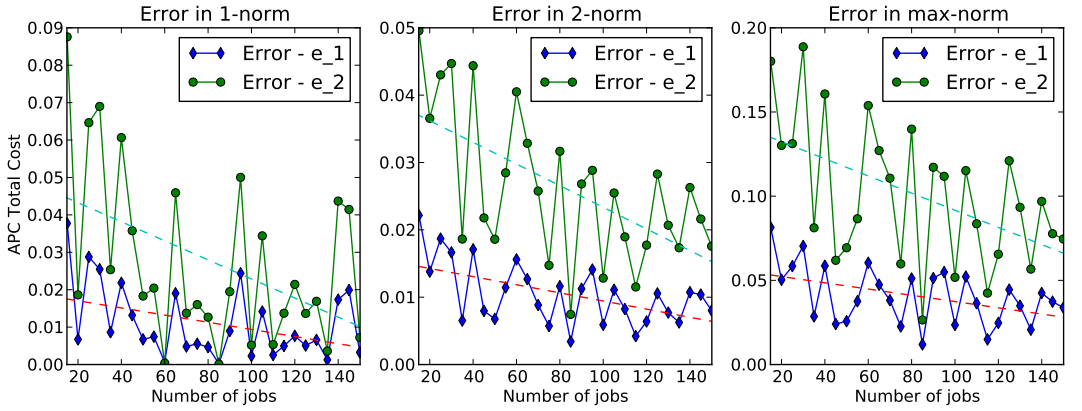
Table 8: Parameter settings for the solutions with the lowest objective values for the Rostering test case.

Parameter	Space filling	Non-Space filling
Rostering_parameter1	71	71
Rostering_parameter2	True	True
Rostering_parameter3	False	False
Rostering_parameter4	False	False
Rostering_parameter5	1	5

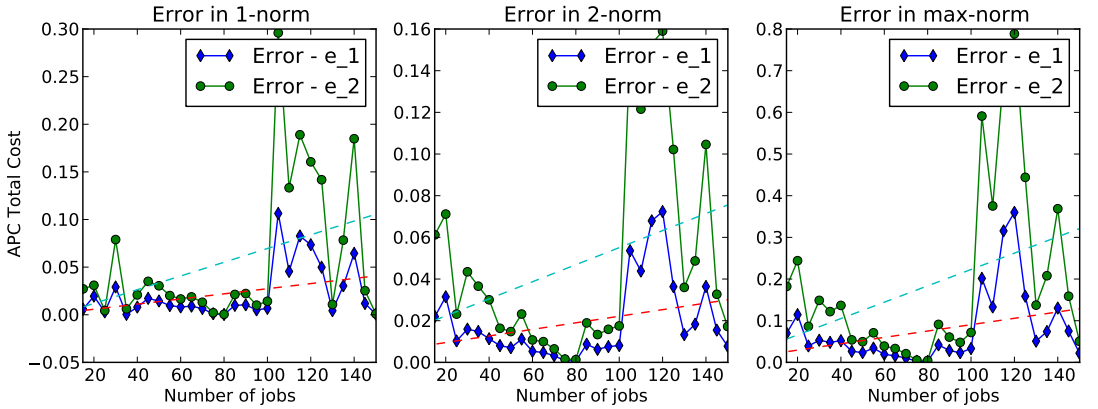
D Plots of Prediction model error



(a) No KPI constraints and using the space filling method.



(b) Continuing from a) with 10 batches of 5 points with space filling method.



(c) Continuing from Figure 22 with 10 batches of 5 points with space filling method.

Figure 26: The measures of error e_1 and e_2 given in (61) for no KPIs as well as space filling. Using 10 initial points followed by 18 batches of 5 points ($10+18\times5$) with space filling in a) and additional 10 batches of 5 points with space filling in b). In c) test $\mathcal{T}_{0.1}$ is performed ($10+18\times5$) with the addition of 10×5 points with space filling. The dashed lines show the linear regression trends.

E ZDT multiobjective test functions

The ZDT test functions [ZDT00] are used to test multiobjective algorithms. There are six functions named ZDT1-ZDT6 but only the three first are presented here. For the three first test functions there are $n = 30$ decision variables $x_1, \ldots, x_n$ although that number is reduced considerably for the tests in this thesis. All decision variables take values in the unit interval: $x_i \in [0,1] \ \forall i$. All of the three test functions defined below are structured in the same way and consist of the three functions f_1, g, h:

$$\begin{array}{lrcl} \text{Minimize} & \mathcal{T}(\mathbf{x}) & = & (f_1(x_1), f_2(\mathbf{x})) \\ \text{subject to} & f_2(\mathbf{x}) & = & g(x_2, \ldots x_n)h(f_1(x_1), g(x_2, \ldots, x_n)) \\ \text{where} & \mathbf{x} & = & (x_1, \ldots, x_n) \end{array} \tag{62}$$

The problems consist of two objectives and n decision variables where function f_1 is a function of the first decision variable only, g is a function of the remaining $n-1$ variables and h is a function of f_1 and g.

- The test function ZDT1 has a convex Pareto front:

$$\begin{array}{lcl} f_1(x_1) & = & x_1 \\ g(x_2, \ldots x_n) & = & 1 + 9 \cdot \sum_{i=2}^{n} x_i/(n-1) \\ h(f_1, g) & = & 1 - \sqrt{f_1/g} \end{array} \tag{63}$$

- The test function ZDT2 is the nonconvex (concave) counterpart to ZDT1:

$$\begin{array}{lcl} f_1(x_1) & = & x_1 \\ g(x_2, \ldots x_n) & = & 1 + 9 \cdot \sum_{i=2}^{n} x_i/(n-1) \\ h(f_1, g) & = & 1 - (f_1/g)^2 \end{array} \tag{64}$$

- The test function ZDT3 has a Pareto front which consists of several non-continuous convex parts:

$$\begin{array}{lcl} f_1(x_1) & = & x_1 \\ g(x_2, \ldots x_n) & = & 1 + 9 \cdot \sum_{i=2}^{n} x_i/(n-1) \\ h(f_1, g) & = & 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1) \end{array} \tag{65}$$

For all three functions, the Pareto front is formed with $g(\mathbf{x}) = 1$.

F Rostering: Comparing best solution with Matador standard

In Figure 27 the progress of the cost for a solution is shown for the matador standard settings as well as for the best solution found for the Rostering test case, shown in Appendix C.2.

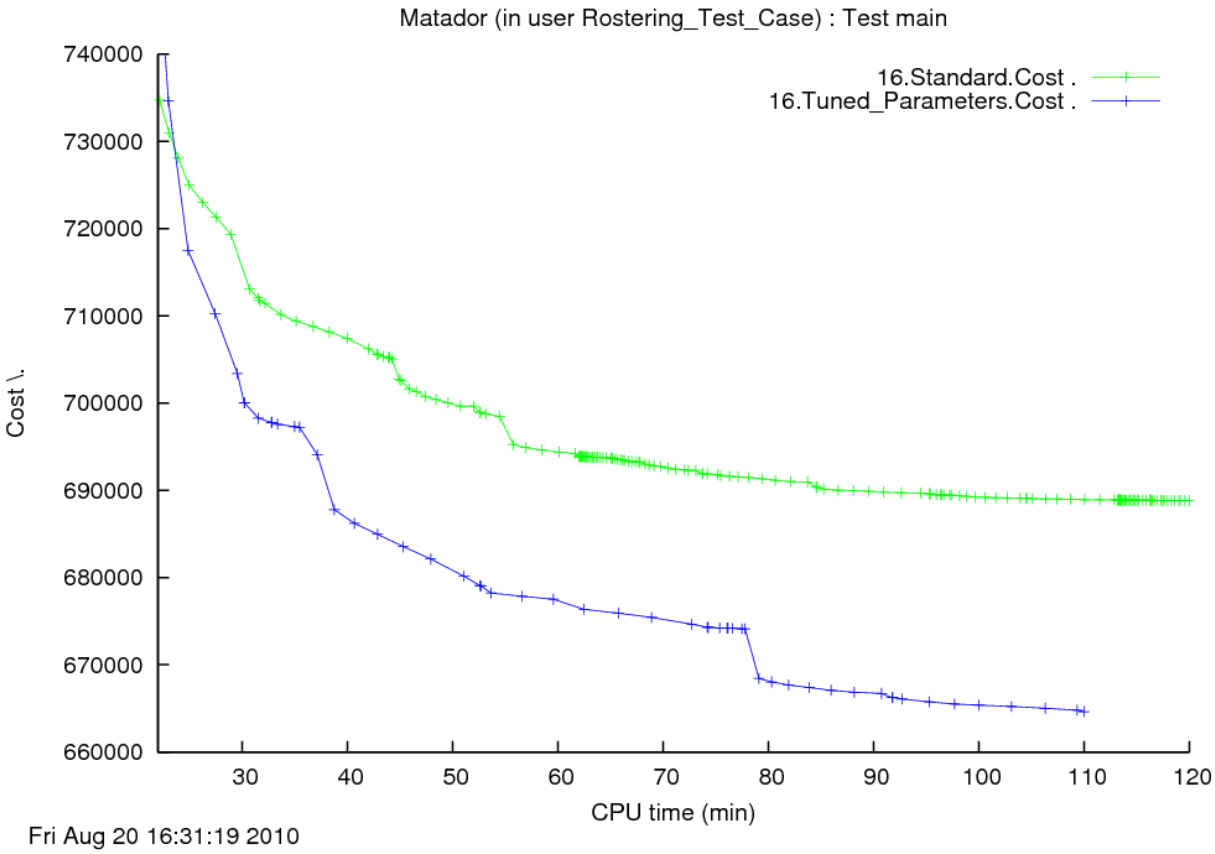


Figure 27: Plot of the development of the *Cost* as a function of the *CPU time* for the Rostering test problem. The figure shows the matador standard run (upper green line) and the best parameter setting found using RUSA (lower blue line). The matador standard ran for only 60 minutes but the script was run again to compare with the results for the tuned settings after 110 minutes.