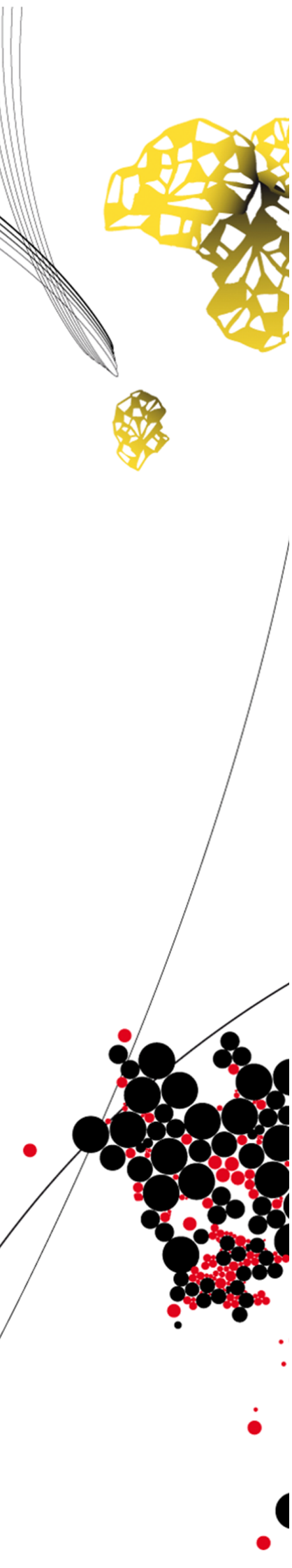


UNIVERSITY OF TWENTE.

Faculty of Behavioural, Management & Social Sciences

Optimization in Diamond

Joris van der Meulen

M.Sc. Thesis
September 2016

Supervisors:
dr. ir. M. R. K. Mes
dr. ir. J. M. J. Schutten
ir. N. Nijenmanting

Production & Logistics Management
(Industrial Engineering & Management)
Faculty of Behavioural,
Management & Social Sciences
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Management Summary

Diamond is a decision making tool that enables users to construct models of the processes that take place in dairy factories and optimize these processes by varying the product mix and technology settings in these models. Differential Evolution (DE), a stochastic optimization method, is implemented in Diamond to perform these optimizations. DE's effectiveness and efficiency depend on the values of several auxiliary optimization parameters. In the current situation, a user of Diamond has to set values for those parameters before performing an optimization. The goal of this research is to find an approach for determining the values of DE's auxiliary optimization parameters in Diamond so that they do not have to be set by the user anymore.

We apply the approaches of parameter selection and meta-optimization to tune the auxiliary optimization parameters of DE in Diamond. Parameter selection comes down to selecting values for the auxiliary optimization parameters relying on conventions and default values. Meta-optimization involves treating the search for good auxiliary parameter values as an optimization problem in its own right. It hence requires the implementation of an optimization method on the meta-level. The meta-level optimizer aims to find good values for the auxiliary parameters of DE, which in turn aims to find good values for the optimization variables of the actual problem in Diamond. We depict this process graphically in Figure 1. We select the Nelder-Mead (NM) method as meta-level optimizer.

We evaluate three different performance aspects for our solution approach: reliability, robustness, and efficiency. An assumption regarding meta-level search spaces based on which we selected the NM method as meta-optimizer does not seem to hold, impeding on the reliability of our solution approach. We applied our solution approach 5 times to 3 different problems and it only yielded consistently good results for one of the problems, and failed to yield good results twice for both other problems. Our solution approach appears to be quite robust against changes in the actual problem, but more tests in this direction have to be performed. Our solution approach seems efficient when we compare it to the strategy of selecting commonly advised auxiliary optimization parameters from literature. However, it barely performs better (on the problem for which our solution approach yielded consistently good results) than two straightforward strategies that use a similar

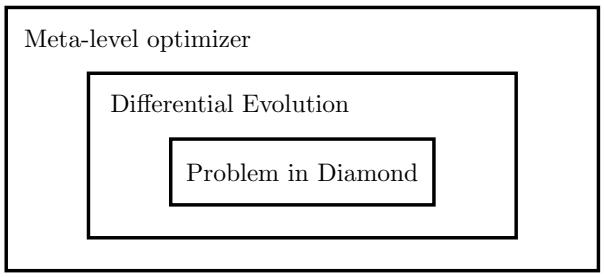


Figure 1: Meta-optimization in Diamond

amount of computation time and that do not rely on any assumptions regarding meta-level search spaces for their reliability.

By applying parameter selection and meta-optimization, we have constructed an approach for determining the values of DE's auxiliary optimization variables so that they do not have to be set by the user anymore before performing an optimization, which was the goal of our research. We however conclude that our solution approach, even though it is successful in tackling the research problem, is not very promising for Diamond. In particular, the NM method is not such a good meta-optimizer. In a more general sense, the practical suitability to Diamond of the combined parameter selection and meta-optimization approach can be questioned. The biggest downside of this approach is that auxiliary parameters resulting from a meta-optimization run cannot easily be generalized to different values for the auxiliary parameters to which parameter selection has been applied.

We recommend further research aimed at improving the speed and quality of our solution approach, such as making use of information that has been obtained in previous meta-optimization runs and parallelizing our solution approach. Our main recommendations are on the deeper levels of Figure 1 though. We recommend further research in the field of (self-)adpative DE, in which feedback from the search progress is used to control the values of the auxiliary optimization parameters. Adaptive DE variants usually introduce new auxiliary parameters whose values the user must decide upon, but these new auxiliary parameters are generally a lot more robust than those of standard DE. It might therefore be possible to determine values for the new auxiliary parameters that can be applied to Diamond in general instead of being suitable for only one problem and perhaps other instances of that problem. Finally, information extracted from the actual problems and similarities between future problems in Diamond can potentially be used to develop an algorithm tailored specifically for Diamond that is more efficient and robust than DE or any other general metaheuristic.

Acknowledgements

I would like to express my gratitude to those who have helped me with my graduation assignment. First of all, I would like to thank Martijn and Marco, my supervisors at the University of Twente, for their guidance and their critique. I would like to thank everybody at Reden and the team at FrieslandCampina, for making me feel welcome and taking the time to discuss my ideas about the project. Special thanks go to Niels, who supervised me at Reden and frequently took the time to explain me something or help me out somewhere. Finally, I would like to thank Sarah, my friends, and my family, for both the support and the distraction they have provided me with over the course of this assignment.

Table of Contents

List of Abbreviations

DE	Differential Evolution
DOE	design of experiments
EA	evolutionary algorithm
FC	FrieslandCampina
GA	genetic algorithm
HJ	Hooke-Jeeves
LJ	Luus-Jaakola
LUS	Local Unimodal Sampling
NFL	no free lunch
NLP	nonlinear programming problem
NM	Nelder-Mead

Chapter 1

Introduction

The research in this thesis revolves around Diamond, a project conducted by Reden and the FrieslandCampina (FC) department milk valorization. In this project, a software solution is developed that is also called Diamond. Diamond is a decision making tool that enables users to construct models of the processes taking place in dairy factories and optimize these processes by varying the product mix or technology settings in these models.

This chapter functions as an introductory chapter to Diamond and the research we conduct. In Section 1.1 we briefly explain how Diamond works. We identify the research problem in Section 1.2 and determine the research goal in Section 1.3. In Section 1.4 we introduce the research questions and describe the set-up of the remainder of this report based on those questions.

1.1 Diamond

Diamond is developed as a tool that can give decision-making support for two types of problems that FC frequently encounters in processes in their factories. The first type of problem is related to raw material sourcing. Such problems arise when several raw materials or waste flows from other factories can be used for a certain process. The decision that has to be made in this type of problem is which raw materials to use in the process and in what volume.

The second type of problem arises when a process consists of multiple steps leading to several end products. In such processes there are variable technology settings that influence the product specifications of the end products and the amounts of products that are produced. The decision that has to be made in this type of problem is what values to select for those technology settings.

In order for Diamond to give decision-making support for these problems, the user has to provide the software with a problem that resembles the process in which the user wants this support. A problem in Diamond is defined by three parts: a datastore, a network, and optimization variables. In this section we

briefly discuss each of these parts.

The network

A user of Diamond has to load or construct a network in Diamond. A network in Diamond is constructed by dragging elements onto a grid and linking them with connectors. Those elements represent the inputs, outputs, and all intermediate steps of the process that is being modelled. In this report the inputs are referred to as raws, the outputs as sales, and the intermediate steps as unit operations. The connectors represent the incoming and outgoing product flows of each element. Raws only have outgoing product flows, sales only have incoming product flows, and unit operations always have both.

We present a screen capture of the interface of Diamond in Figure 1.1. This screen capture is blurred for confidentiality reasons. It displays a network that we have constructed on the grid with elements and connectors. We have assigned a different colour to each element type. Raws are distinguished by their blue colour, sales by their pink colour, and unit operations by their green colour. A list can be distinguished to the left of the grid in which the network is constructed. This list contains the elements that can be used in the network. It is provided by loading a separate datastore in Diamond that defines these elements.

Figure 1.1: Screen capture of Diamond's interface (blurred for confidentiality reasons)

The datastore

A datastore has to be loaded in Diamond to define the elements that can be used for the construction of a network. When a user wants to load an already constructed network in Diamond, the matching datastore in which the elements that have been used for the construction of that network are defined has to be loaded as well. In such a datastore the different elements are defined in the following way.

A raw is defined by a parameter vector in which the product specifications of the raw are stored. With product specifications we mean the amount of each ingredient in the product, the amount of each nutrient in the product, and product properties of interest, such as the viscosity and the water activity of the product. A sale is defined by two parameter vectors that denote lower and upper bounds for its product specifications. A unit operation is defined by its ingoing and outgoing product flows and the transfer functions between them. Since unit operations need to model every processing step that might happen in a dairy factory, there can be a large amount of transfer functions defining one unit operation and these functions can be complex. Transfer functions in a unit operation can depend on variables. We refer to those variables as the technology settings of that unit operation. A user can assign values to the technology settings of a unit operation if that unit operation is used in the network.

The optimization variables

When a datastore and a network have been loaded or constructed in Diamond, optimization variables need to be selected. The technology settings of a unit operation can be optimization variables. The input volume of a raw can also be an optimization variable.

Performing an optimization

When a network has been constructed and optimization variables have been selected, a user can click the optimize button in the lower left part of the interface. When this button is clicked, a dialog pops up in which the user can inspect the technology settings and input volumes that are optimization variables. The user has to set values for several auxiliary optimization parameters in this dialog and can then start an optimization run. An optimization run takes minutes to hours, depending on the size and complexity of the problem at hand. The values selected for the auxiliary optimization parameters also influence the runtime. Upon termination of an optimization run, Diamond displays the best values for the op-

timization variables that have been encountered during the run, the impact that those values have on the process, and the resulting profit. It is now up to the user to adjust the product mix or technology settings in practice so that they match the values of Diamond, or decide not to.

1.2 Problem Identification

A user of Diamond has to provide the software with a datastore and a network of a process in which optimization variables are selected. In turn, after some computation time, Diamond provides the user with hopefully near-optimal values for the optimization variables. In the context of Diamond, optimal values for the optimization variables mean those values that lead to the highest profit that can be obtained in the process that has been modelled. We depict this procedure in Figure 1.2. A part of this figure is enclosed by a green dotted line. This part represents the software solution Diamond. An iterative procedure takes place within it.

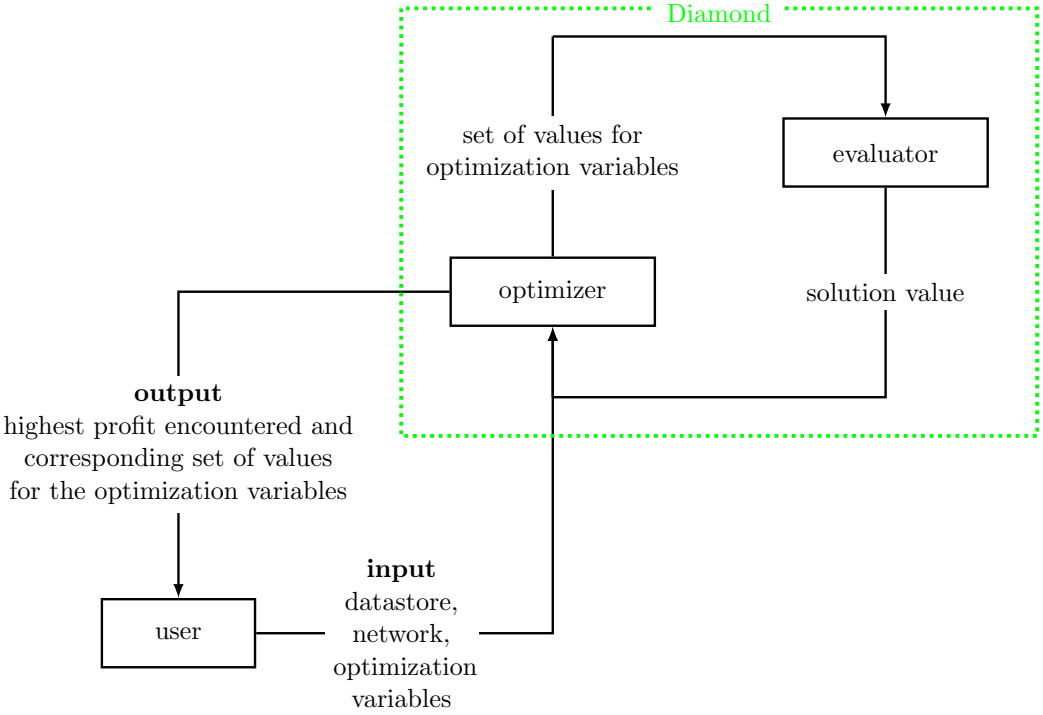


Figure 1.2: How Diamond operates – a rough depiction

Diamond roughly consists of two parts. In one part, which we call the evaluator, a solution value for a specific set of values for the optimization variables

is evaluated. This solution value is related to the profit in such a way that a lower solution value generally corresponds to a higher profit. The other part of Diamond is responsible for determining the sets of values for the optimization variables that have to be evaluated by the evaluator. We call this part the optimizer because it aims to find optimal values for the optimization variables. To this end some of the evaluated solution values and their corresponding sets of values for the optimization variables are temporarily stored in the optimizer. The best solution value that has been encountered is always stored and so is the corresponding set of values for the optimization variables.

The choice has been made by the developers of Diamond to treat the evaluator as a black box, meaning that we can obtain an output from the evaluator for a given input but have no knowledge of its internal workings. In the case of the evaluator, the input is a set of values for the optimization variables and the output is the corresponding solution value, as can be seen in Figure 1.2. As soon as a datastore, a network, and optimization variables are defined, a problem in Diamond can be formulated as a nonlinear programming problem (NLP), because of which the evaluator's internal workings are known. Although there are solution methods for solving specific types of NLPs by making use of certain characteristics of their solution spaces, the NLPs resulting from the problems in Diamond have complex, multimodal[1], non-continuous, non-linear objective functions. Next to soft constraints that lead to a penalty in the objective function if, and based on the extent that, they are not satisfied, the NLPs resulting from the problems in Diamond also have hard constraints. Constrained NLPs with complex, multimodal, non-continuous, non-linear objective functions are generally treated as black boxes because it is difficult to make useful assumptions regarding their solution spaces. Furthermore, users of Diamond have a lot of freedom in determining the datastore, network, and optimization variables, because of which the internal workings of the evaluator vary. These are the reasons for treating the evaluator as a black box.

Treating the evaluator as a black box, and hence ignoring any assumptions that could possibly be made about the solution spaces of the problems in Diamond, brings along a problem. If no assumptions are made regarding a solution space, the no free lunch (NFL) theorem for optimization states that each optimization method is as likely to find a good solution value as any other. Wolpert and Macready (1997) prove this by showing that, for an arbitrary measure of performance, the probability of obtaining a specific sequence of solution values,

[1]Having multiple optima, as opposed to having one which makes a function unimodal.

averaged over all possible functions, is independent from the applied algorithm. In other words, one could not expect to find an optimization method that performs any better than any other optimization method (Jansen, 2013).

There are solution methods that, by incorporating auxiliary optimization parameters, can overcome the implications of the NFL theorem. These solution methods are called metaheuristics and they can be efficient on a wide range of problems provided that they are well parametrized (Luke, 2013). One of those metaheuristics has been implemented in the optimizer in Diamond.

Differential Evolution (DE) is the metaheuristic that has been implemented in the optimizer in Diamond. We explain how this metaheuristic works in Chapter 2. DE has been selected by the developers of Diamond because it is a competitive metaheuristic with relatively few auxiliary optimization parameters. It has been shown that DE is efficient in a wide variety of practical as well as theoretical problems (Das & Suganthan, 2011; Civicioglu & Besdok, 2013; Lampinen, Storn, & Price, 2005, pp. 156-182). Like any metaheuristic though, values have to be selected for several auxiliary optimization parameters in order for DE to perform well. At the moment it is the case that a user of Diamond has to choose values for those parameters before performing an optimization. FC however does not want Diamond to require any optimization-related input from its users other than the datastore, the network, and the optimization variables. This brings us to our problem statement: *Values for the auxiliary optimization parameters of DE have to be set by the user before performing an optimization in Diamond.*

1.3 Research Goal

We formulate our research goal based on the problem statement that we have defined in Section 1.2. We formulate the research goal as follows: *Find a way to determine values for DE's auxiliary optimization parameters so that they do not have to be set by the user before performing an optimization in Diamond.*

1.4 Research Questions

To conduct research in a structured manner, we formulate several research questions. Based on the problem statement that we have formulated in Section 1.2 and the research goal that we have defined in Section 1.3, we formulate the following main research question: *How can we determine values for DE's auxiliary optimization parameters in Diamond?* We formulate several research questions

that support the main research question:

RQ 1. What is the current situation?

- **1(a).** What problems for Diamond do we have at our disposal?
- **1(b).** How does DE work and what are its auxiliary optimization parameters?

RQ 2. What approaches for determining values for auxiliary optimization parameters can we find in academic literature?

RQ 3. What is a good approach for determining values for DE's auxiliary optimization parameters in Diamond?

RQ 4. How does the proposed approach perform?

In Chapter 2 we discuss research question 1. We introduce three problems from practice and explain the workings of DE in this chapter. In a literature review in Chapter 3, we discuss research question 2. In Chapter 4 we answer research question 3 by proposing an approach for determining the values of DE's auxiliary optimization parameters in Diamond. We discuss the performance of this approach in Chapter 5 and answer research question 4 this way. In Chapter 6 we conclude this research and give recommendations for future research.

Chapter 2

Current Situation

In this chapter we give an overview of the current situation. We introduce the three problems from practice that we have at our availability in Section 2.1. In Section 2.2 we describe how DE, the metaheuristic that is implemented in Diamond, works and identify the auxiliary parameters that a user of Diamond currently has to set before performing an optimization. We conclude this chapter in Section 2.3.

2.1 Problems

Recall from Chapter 1 that Diamond should give decision-making support for two types of problems that FC often encounters in processes in their factories. In one type of problem, the decision has to be made which raw materials to use in the process and in what volume. In the other type of problem, the decision has to be made what values to set for several variable technology settings. In the remainder of this report we refer to the first type of problems as mixing problems and to the second type of problems as technology problems. Recall furthermore that problems in Diamond are defined with a datastore, a network, and optimization variables.

We have one mixing and two technology problems from practice at our availability. They are modelled in Diamond and the optimization variables are selected. We anonymize the networks of these problems so that we can display their structures. In this report we refer to the problem that is of the mixing type as the Mix problem. We display the structure of the Mix problem in Figure 2.1. The optimization variables in this problem are the input volumes of eight different raws. We call one of the technology problems the Split problem, because the optimization variables in this problem are twenty-five settings that determine how several product flows are divided (splitted) over the network. The other technology problem we call the Tech problem. There are fifteen optimization variables in this problem, namely the input volume of a raw and fourteen

technology settings. We display the network structures of the Split and the Tech problem in Appendix A.

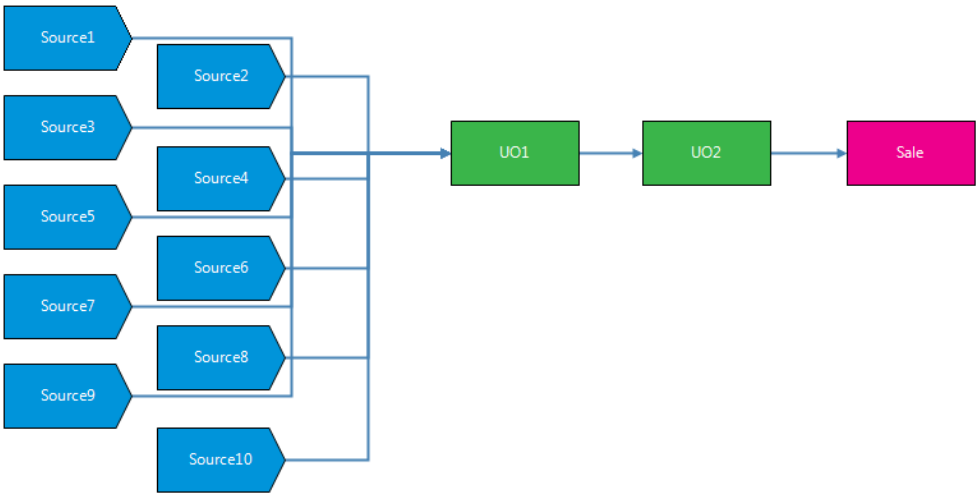


Figure 2.1: Network structure of the Mix problem

We summarize some information about the three problems that we have at our availability in Table 2.1. In this table we introduce n, the problem dimension. It equals the amount of optimization variables in a problem. The evaluation time in the third column of the table denotes the approximate time it takes Diamond to evaluate one point in the solution space of the corresponding problem. This gives an indication of the complexity of the problems and the time it takes to perform one optimization run, which consists of thousands such evaluations.

Table 2.1: Some information about the available problems

Problem type	Problem name	n	Evaluation time
Mixing	Mix problem	8	8 milliseconds
Technology	Tech problem	15	18 milliseconds
	Split problem	25	45 milliseconds

For each of the three problems that we have at our availability, only one instance is defined. Other instances of the problems arise when an alteration is made in a problem. Examples of such alterations are price fluctuations and changes in the product specifications of a certain raw or sale due to governmental decisions on when a product can be labelled low-fat or calcium-rich. At times,

the transfer functions defining the processing step that takes place in a certain unit operation might require alteration, for example when a piece of machinery is replaced by a slightly different one or when research points out that there is a better formula to describe a certain chemical process.

2.2 Differential Evolution

DE is introduced by Storn and Price (1997). It is a stochastic optimization method belonging to the class of evolutionary algorithms (EAs). EAs are metaheuristics that incorporate mechanisms inspired by biological evolution such as reproduction, mutation, and, recombination. The older and more widely known genetic algorithms (GAs) (Goldberg, 1989) belong to this same class. According to some taxonomies, DE is considered a type of GA because both DE and GAs make use of a population of solutions on which selection, mutation, and crossover take place to iteratively create new population members. In both GAs and DE the population size remains constant by discarding old members when new individuals enter the population. We however are of the opinion that there are too many characteristics setting DE apart from GAs for it to be considered one. We present these characteristics in Table 2.2. Studies focusing on the comparison of several metaheuristics indicate that GAs and other EAs are frequently outperformed by DE (Vesterstrøm & Thomsen, 2004; Kannan, Slochanal, & Padhy, 2005; Xu & Li, 2007).

Table 2.2: Some differences that set DE apart from GAs

In a GA	In DE
Selection takes place at the beginning of an iteration	Selection takes place at the end of an iteration
Two parents create two offspring in each iteration	Every parent creates one offspring in each iteration
No other population members are involved in creating the offspring but the two parents	Three randomly selected other population members are involved in creating the offspring of a parent
The two offspring always replace two current population members	An offspring only replaces its parent if it corresponds to a better solution value

DE consists of two stages, an initialization stage and a main loop. In the initialization stage, a population of solutions is created. A solution in the context

of Diamond is a set of feasible values for the optimization variables. With feasible we mean that the set of values yields a solution value smaller than a very large value when it is evaluated by the evaluator. The evaluator returns this very large value when one or more of the hard constraints of the NLP resulting from the network, datastore, and optimization variables in Diamond are not satisfied.

The constraints that define the ranges of the optimization variables are hard constraints so that, for example, a negative amount of product does not become possible anywhere in the process. All other constraints are incorporated in the objective function as soft constraints, meaning that they lead to a penalty in the objective function if, and based on the extend that, they are not satisfied. Because of this it is possible to evaluate solutions that are actually infeasible in the current process, which has as a result that sets of feasible values can be found relatively quickly via random search and that the initialization stage of DE in Diamond will not take long.

Although the goal in Diamond is to maximize profit, the objective function is such that we are dealing with a minimization problem, which is also the reason for assigning infeasible solutions a very large value. In fact, the objective function is such that negative solution values correspond to profit and positive solution values correspond to losses. Large positive solution values generally indicate unsatisfied soft constraints.

The solutions for a problem in Diamond can be represented by n-dimensional vectors in which each element represents the value selected for one of the optimization variables. We denote the solution value corresponding to a vector $\vec{x}$ in the solution space by $f(\vec{x})$. $f(\vec{x})$ is the output of the evaluator from Figure 1.2 when it receives the vector of values for the optimization variables $\vec{x}$ as input. The population size is denoted by NP. NP is one of the auxiliary optimization parameters of DE that the user has to set a value for in Diamond before performing an optimization. After the initialization stage, a population of NP n-dimensional solution vectors have been generated. In Algorithm 1 we display pseudocode for the initialization stage of DE in Diamond. We denote the very large value that is assigned to infeasible solutions by ∞ in this algorithm.

The second and final stage of DE in Diamond is the main loop. After the initialization stage, mutation, crossover, and selection take place iteratively until a termination criterion is met. We depict this schematically in Figure 2.2.

In the mutation step, a mutated vector is obtained for each of the NP population members. This is done by perturbing a randomly selected population member with a scaled difference of two other randomly selected population mem-

Algorithm 1 Initialization stage of DE in Diamond

INPUT: Auxiliary optimization parameter NP,
search space of the problem
OUTPUT: Population of NP vectors $\overrightarrow{x_1}, .., \overrightarrow{x_{NP}}$
and their corresponding solution values

1: **for** $i \leftarrow 0, NP$ **do**
2: **repeat**
3: Pick random $\overrightarrow{x_i}$ within the search space
4: Determine $f(\overrightarrow{x_i})$
5: **until** $f(\overrightarrow{x_i}) < \infty$
6: **end for**

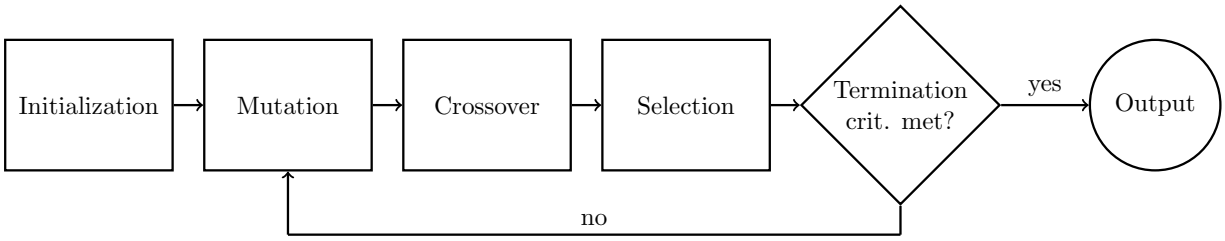


Figure 2.2: Schematic depiction of DE in Diamond

bers. The current population members can be seen as the parents and we hence denote them by $\overrightarrow{p_i}$, $i = 1, .., NP$. For each parent, three random other population members are selected. If we let r_1, r_2, and r_3 be distinct random integers in the set $\{1, 2, .., NP\} \setminus i$, we can denote the three population members that are randomly selected for parent vector $\overrightarrow{p_i}$ by $\overrightarrow{x_{r_1(i)}}$, $\overrightarrow{x_{r_2(i)}}$, and $\overrightarrow{x_{r_3(i)}}$.

The mutant vector $\overrightarrow{m_i}$ belonging to parent vector $\overrightarrow{p_i}$ is constructed via the equation $\overrightarrow{m_i} = \overrightarrow{x_{r_1(i)}} + F \cdot (\overrightarrow{x_{r_2(i)}} - \overrightarrow{x_{r_3(i)}})$. The parent vector is not explicitly taken into account in the construction of its corresponding mutant vector except for that it cannot be one of the randomly selected population members. In the mutation step, another auxiliary optimization parameter of DE is introduced, the mutation factor F. According to Storn and Price (1997), F has to be in the range $[0, 2]$. We visualize the creation of a mutant vector in a two-dimensional problem space in Figure 2.3. In this figure, the black dots represent the current population members and the grey dot represents the mutant vector.

After the mutation step there are NP mutant vectors, one for each current population member. Now the crossover step takes place. In this step NP new solution vectors are generated. These solution vectors can be seen as the children of the current population members and we hence denote them by $\overrightarrow{c_i}$, $i = 1, .., NP$.

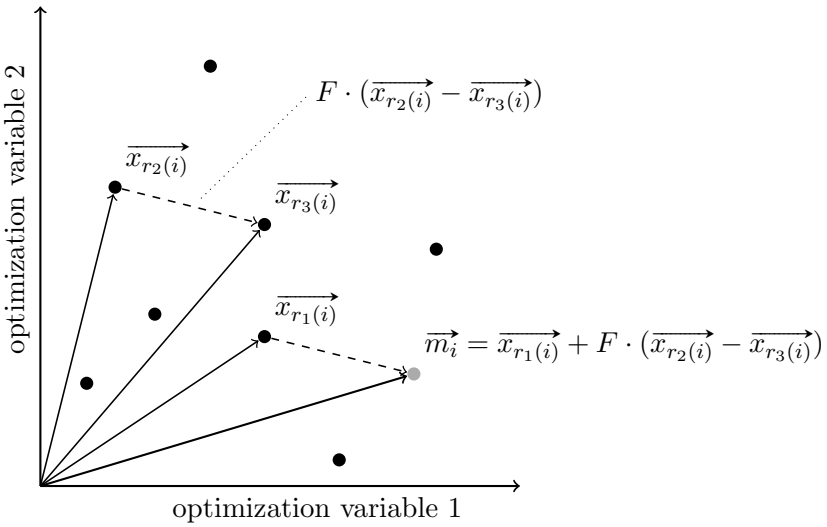


Figure 2.3: The mutation step illustrated for a 2-dimensional problem

For the construction of the child $\vec{c_i}$, its parent $\vec{p_i}$ and the corresponding mutant vector $\overrightarrow{m_i}$ are used. Each vector element of the child $\vec{c_i}$ is either copied from the parent vector $\vec{p_i}$ or from the corresponding mutant vector $\overrightarrow{m_i}$. In the crossover step another auxiliary optimization parameter of DE is introduced, the crossover constant CR. CR influences how many of the vector elements of a child on average originate from its parent and how many on average originate from the corresponding mutant vector. CR has to be in the range $[0, 1]$. We visualize the idea behind the crossover step in Figure 2.4. In this figure it can be seen how a child $\vec{c_i}$ is constructed with the vector elements of its parent $\vec{p_i}$ and the vector elements of the mutant $\overrightarrow{m_i}$.

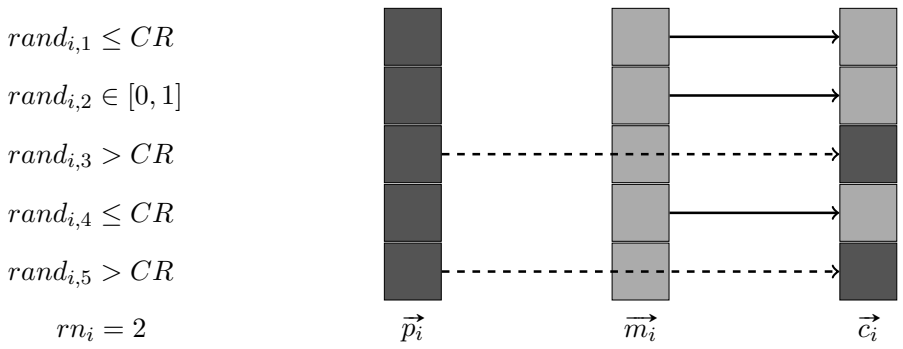


Figure 2.4: The crossover step illustrated for a 5-dimensional problem

In the crossover step, for each population member i, n random numbers are

drawn from the interval $[0, 1]$ and one random integer is drawn from the set $\{1, 2, .., n\}$. We denote these numbers by $rand_{i,j}$, $j = 1, .., n$ and rn_i respectively. If $rand_{i,j}$ is larger than the constant CR, the j^{th} element of the child vector $\vec{c_i}$ is similar to the j^{th} element of the vector of its parent $\vec{p_i}$. If this is not the case, the j^{th} element of the child vector $\vec{c_i}$ is similar to the j^{th} element of the mutant vector $\vec{m_i}$. The ${rn_i}^{\text{th}}$ element of the child vector $\vec{c_i}$ is always similar to the j^{th} element of the mutant vector $\vec{m_i}$. This is to ensure that $\vec{c_i}$ differs from $\vec{p_i}$ in at least one element. In Figure 2.4, the relevant values of the random numbers that are drawn are denoted to the left of the figure.

After the crossover step all mutant vectors are discarded, leaving us with NP parents (the current population) and NP children, each child belonging to one parent. Recall that the population members of DE in Diamond are n-dimensional vectors that each represent a set of values for the optimization variables in the problem that is being optimized by Diamond. Because of this, each population member corresponds to a solution value. The solution values of the children are evaluated and if the solution value corresponding to a child is better than the one corresponding to its parent, the child replaces its parent in the current population. This is the selection step. It marks the end of an iteration.

The idea behind DE is that through mutation, crossover, and selection, the population will hopefully become more and more concentrated around local optima and eventually concentrate itself around the global one or global ones. Depending on the values selected for the auxiliary optimization parameters and the specific problem at hand though, DE might converge too quickly and thus end up in a local optimum or not convergence and thus end up in no optimum at all[1] (Locatelli & Vasile, 2014). Other values can lead to very slow convergence and are hence unable to provide the user with a good solution within a reasonable time limit or number of function evaluations. Finding proper values for the auxiliary optimization parameters for the problem at hand is therefore essential. This process is known as tuning. We deal with tuning in Chapter 3.

In Algorithm 2, we display pseudocode for DE in Diamond. In line 3 of this algorithm, we refer to a termination criterion. In the current situation this termination criterion can either be a manual stop, a maximum number of iterations, or a maximum number of iterations without change in the solution value corresponding to the best-encountered solution $\vec{y}$. This means that users have to decide when to stop the algorithm or select a value for either the maximum num-

[1]There are DE variants for which convergence to the global optimum in probability can be proven but these do not take speed or a maximum number of evaluations into account (Hu, Xiong, Su, & Zhang, 2013). These variants are hence not very useful in practice.

ber of iterations or the maximum number of iterations without change, which is not in line with FC's desire that no decisions concerning auxiliary optimization settings have to be made by the user.

Algorithm 2 DE in Diamond

INPUT: Auxiliary optimization parameters NP, F, and CR,
search space of the problem
OUTPUT: Best-encountered solution $\vec{y}$

```
1: Initialization (see Algorithm 1)
2: y⃗ ← (x⃗_i corresponding to lowest f(x⃗_i) of Algorithm 1)
3: while Termination criterion not met do
4:     for i ← 0, NP do
5:         Perform mutation and crossover to obtain child c⃗_i of x⃗_i
6:         Determine f(c⃗_i)
7:         if f(c⃗_i) < f(x⃗_i) then                          ▷ Selection
8:             x⃗_i ← c⃗_i
9:             if f(x⃗_i) < f(y⃗_i) then                      ▷ Update best
10:                y⃗ ← x⃗_i
11:            end if
12:        end if
13:    end for
14: end while
```

We restrict the software to the termination criterion of a maximum number of iterations such that a limited amount of function evaluations is performed in one optimization run. A termination criterion based on a maximum number of function evaluations makes sense from a practical perspective. This way, users can easily be informed of the approximate runtime that is left until termination and there is a possibility to reproduce results on different PC's[2]. We only count the evaluations of points satisfying all hard constraints towards reaching the maximum number of function evaluations. The hard constraints define the ranges of the optimization variables and if one of them is not satisfied, the evaluator immediately returns a very large value. The time this takes is negligible compared to the time it takes the evaluator to evaluate a point in the search space of the problem that does not violate the ranges of the optimization variables. Different optimization runs on the same problem can therefore only vary little in runtime as long as the computation environment remains unchanged. The maximum number of function evaluations (resulting in a feasible solution) can be seen as another auxiliary parameter of DE in Diamond. In this report we refer to it as E.

[2]Because of DE's stochasticity, results cannot exactly be reproduced unless the same random number stream is used.

2.3 Conclusions on Current Situation

We have three problems at our availability that we briefly discuss in Section 2.1. One of those problems is of the mixing type and the other two are of the technology type. A new problem instance is created when an alteration is made to an existing problem. In Section 2.2 we describe how DE works. DE is a stochastic optimization method that makes use of a population of solutions in the search space of a problem on which mutation, crossover, and selection take place such that the population hopefully converges in the direction of the global optimum. Like all metaheuristics, DE's effectiveness and efficiency depend on the values of several auxiliary optimization parameters. The auxiliary optimization parameters of DE are the population size NP, the mutation factor F, and the crossover constant CR. The maximum number of function evaluations E can also be seen as an auxiliary parameter of DE in Diamond.

Chapter 3

Literature Review

In this chapter we describe the literature relevant to the process of tuning auxiliary optimization parameters. The focus is on tuning the auxiliary optimization parameters of DE. We describe different tuning approaches in Section 3.1. One of the approaches is meta-optimization, which requires the selection of a meta-level optimizer. In Section 3.2 we review different meta-level optimizers. We conclude this chapter in Section 3.3.

3.1 Tuning Auxiliary Optimization Parameters

Because FC would like to see that no decisions concerning auxiliary optimization settings have to be made by the user, the auxiliary optimization parameters need to be tuned for DE in Diamond. Even though tuning is crucial to metaheuristic optimization both in academic research and for practical applications, only limited research has been devoted to it (Birattari, 2009). There are three different ways in which parameter tuning can be done, namely parameter selection, online parameter initialization, and offline parameter initialization.

Parameter selection involves selecting values for the auxiliary optimization parameters relying on conventions and default values. A default set of parameters however might lead to satisfying results on some problems but can fail to yield good results on other problems. Parameter selection is therefore generally not a good tuning strategy. In practice though it is often applied.

Online parameter initialization is also referred to as parameter control. It involves changing the parameter values during the search. The following approaches can be distinguished in the field of online parameter initialization:

- Deterministic parameter control: Random or deterministic changes in parameter values are made at predefined moments, meaning that the progress of the search is not taken into account. A deterministic parameter control strategy that has been used in combination with DE is steadily decreasing

the crossover constant CR as the number of performed iterations increases (Mezura-Montes & Palomeque-Ortiz, 2009).

- Adaptive parameter control: Feedback from the search progress is used to control the values of the auxiliary optimization parameters. An adaptive parameter control strategy that has been used in combination with DE is selecting a value for the mutation factor based on the relative difference between the solution values corresponding to the best and worst population members (Ali & Törn, 2004).

- Self-adaptive parameter control: This can be seen as a subclass of adaptive parameter control. In self-adaptive parameter control, each member of the population has an individual auxiliary optimization parameter for a certain step that evolves during the search. Self-adaptive parameter control strategies for DE would involve replacing F by an NP-dimensional vector of F_is or replacing CR by an NP-dimensional vector of CR_is such that each population member corresponds to their own mutation factor or crossover constant. F_is or CR_is that did not lead to the generation of good trial vectors during a part of the search can then be replaced by other F_is or CR_is that did lead to the generation of good trial vectors or by randomly selected other values within a certain range (Brest, Greiner, Boskovic, Mernik, & Zumer, 2006).

There are quite a few DE adaptations that make use of online parameter initialization (Das & Suganthan, 2011). None of these seem promising for Diamond though: online parameter initialization approaches usually introduce new auxiliary optimization parameters whose values the user must decide upon. Furthermore, experiments have shown that there is no general or consistent advantage to using online parameter initialization in combination with DE as opposed to using classical DE with good parameters (Pedersen, 2010).

In offline parameter initialization, the values of the different auxiliary parameters are fixed before the start of an optimization run instead of updated during the execution of the run. The following approaches can be distinguished within the field of offline parameter initialization:

- Manual tuning: This is also referred to as experimentational tuning. It involves trying a default set of values for the auxiliary parameters and based on the results thereof trying a new set of values (Talbi, 2009). This process is repeated until a satisfying set of values for the auxiliary parameters is

found. Manual tuning is a widely applied tuning strategy for metaheuristics but it is not a feasible approach for Diamond since manual tuning requires a lot of input from the user and is hence a time consuming approach, even if the user is familiar with the optimization method (Adenso-Diaz & Laguna, 2006).

- Design of experiments: Performing a design of experiments (DOE) can overcome the problems involved with manual tuning (Box, Hunter, & Hunter, 2005). In tuning parameters with a DOE, each auxiliary parameter is assigned a number of values. These values can be selected randomly or via a specific procedure such as Latin hypercube sampling (McKay, Beckman, & Conover, 1979). All combinations of the values for the different auxiliary parameters are then evaluated several times to give an indication of how well each combination of values performs[1]. The number of values for each variable cannot be too small because in that case the best-encountered auxiliary optimization setting might not be as close to the optimal settings as one would like them to be and hence not yield satisfactory results on the actual problems. A large number of values leads to a very large number of experiments though, which is a drawback of performing a DOE. DOE is a popular method to determine auxiliary parameters for an algorithm, especially when the actual problems are theoretical functions that do not require much evaluation time. In Diamond though, the computation time of each experiment can be large, making DOE a less suitable approach.

- Meta-optimization: The search for the best auxiliary optimization settings of a metaheuristic can be treated as an optimization problem in its own right. Dealing with this optimization problem defines the concept of meta-optimization. In this concept, a black-box optimization method is used as an overlaying meta-optimizer for finding good auxiliary optimization parameters for another optimization method which in turn is used to optimize the actual problem (Pedersen, 2010). We portray this concept graphically in Figure 3.1. With an effective and efficient meta-optimizer, a near-optimal set of auxiliary optimization parameters for another optimization method for a specific problem can be obtained. As opposed to performing a DOE, meta-optimization requires the evaluation of only a small number of values for the auxiliary parameters, provided that an efficient meta-optimizer has

[1] If the base-level algorithm is deterministic, each combination of values for the different auxiliary parameters only has to be evaluated once. Metaheuristics are generally defined as stochastic algorithms though (Luke, 2013).

been selected. We review different meta-optimizers in Section 3.2.

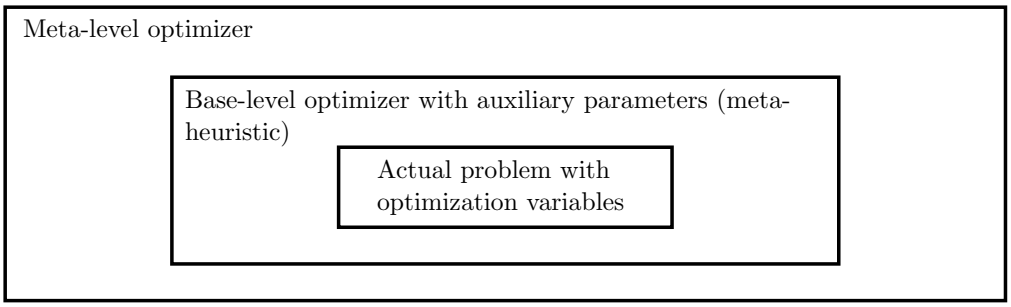


Figure 3.1: The concept of meta-optimization, based on Pedersen (2010)

3.2 Meta-level Optimizers

We have identified three approaches within the field of offline parameter initialization. These approaches are parameter selection, DOE, and meta-optimization, which requires the implementation of a meta-optimizer. In this section we review different meta-optimizers. We distinguish the following three types:

- Metaheuristics: Since it is difficult to make assumptions about the search spaces resulting from varying the auxiliary optimization parameters of a base-level algorithm, a metaheuristic is typically used as overlaying meta-optimizer (Bäck, 1994; Cortez, Rocha, & Neves, 2001; Meissner, Schmuker, & Schneider, 2006). The auxiliary optimization parameters of DE have been meta-optimized with a metaheuristic by Neumüller, Wagner, Kronberger, and Affenzeller (2012), who use a GA as meta-optimizer but restrict their research to only one specific test function. Using a metaheuristic as meta-optimizer has a considerable drawback. As we have explained before, all metaheuristics require the setting of auxiliary optimization parameters for them to be effective and efficient. This is what makes them so generally applicable. Following the outlines of Section 3.1, one would find that the best approach to obtain good auxiliary parameters for the metaheuristic that is implemented as meta-optimizer would be to implement a meta-meta-optimization method. But where does this stop? We would hence like to find a meta-optimization method that does not require the setting of any auxiliary optimization parameters.

- Race algorithms: Racing involves iteratively evaluating several possible auxiliary parameter settings and discarding a setting as soon as sufficient statistical evidence is gathered against it. Because of this, racing works best for base-level algorithms that have low stochasticity. Race algorithms are originally invented as a way to reduce the amount of experiments that need to be done when performing a full factorial experiment or other type of DOE (Maron & Moore, 1994; Birattari, 2002). Because of this, all auxiliary parameter combinations that are evaluated need to be fixed in the initialization phase of a race algorithm. This however has as result that the optimal settings resulting from racing might not be as close to optimal as one would like them to be since only a limited amount of auxiliary parameter combinations can be selected. Race algorithms can be adapted though by generating a new possible auxiliary parameter setting as soon as another setting is discarded (Van Dijk, Mes, Schutten, & Gromicho, 2014). This approach brings performing a DOE and doing meta-optimization together by turning racing into a (guideline for constructing a) population-based meta-optimizer. For deterministic[2] or low-stochastic base-level algorithms, racing is probably the best way to go. The same goes for base-level algorithms depending on categorical parameters because, since racing is based on performing a DOE, it can easily deal with those as well.

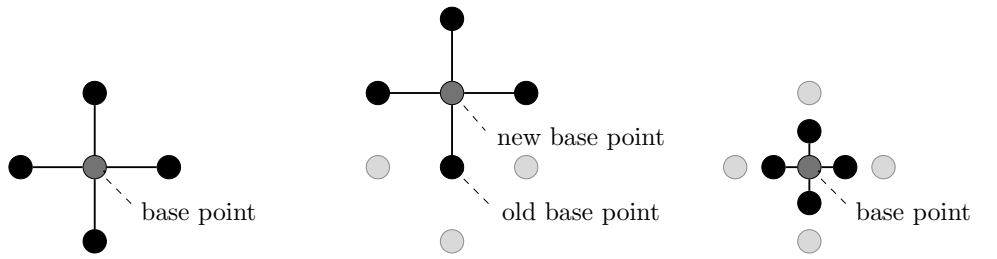
- Classical direct search methods: Recently, some classical direct search methods have been implemented as meta-optimizers. Classical direct search methods are relatively intuitive optimization methods, stemming from the beginning of the digital age, that were invented for optimizing one- or few-dimensional functions without requiring any knowledge about the gradient of the function that is being optimized (Lewis, Torczon, & Trosset, 2000). Such methods have fallen out of favour with the mathematical optimization community by the early 1970s because they lacked coherent mathematical analysis but are still used in some practical applications (Kolda, Lewis, & Torczon, 2003). Although not at all competitive with metaheuristics on most types of problems, classical direct search methods are generally able to quickly locate satisfactory solutions in low-dimensional search spaces and do not require the setting of auxiliary optimization parameters, omitting the need for meta-meta-tuning. This makes them well suitable for meta-optimization in practice, provided that the base-level algorithm does not

[2]In this case multiple problem instances have to be defined and the stochastic component is in which of those instances are tested.

have many auxiliary optimization parameters. Three well-known classical direct search methods that use only function evaluations to search for the optimum are the Hooke-Jeeves (HJ) method, the Nelder-Mead (NM) method, and the Luus-Jaakola (LJ) method (Armaou & Kevrekidis, 2005). We briefly review each of those.

The HJ method

The HJ method (Hooke & Jeeves, 1961), also known as pattern search, has been implemented as meta-level optimizer for several base-level algorithms (Cohen & Meyer, 2011; Gao et al., 2012). The method considers $2n$ points in the search space that lie around one randomly selected base point in a pattern such that each of those $2n$ points is equally far away from the base point and differs from the base point in only one of the variables that make up the search space. We depict a two-dimensional pattern that follows those rules in Figure 3.2a. In meta-optimization, the variables that make up the search space are the auxiliary parameters of the base-level optimizer.



(a) A pattern *(b)* An exploratory move *(c)* A shrink move

Figure 3.2: A pattern and its movements in a two-dimensional search space

This pattern is iteratively moved across the search space or shrunk towards its base point so that hopefully the global optimum is more and more closely approximated as the iterations pass. In each iteration, the points surrounding the base point are evaluated and the one corresponding to the best solution value becomes the new base point, provided that this solution value is better than the one corresponding to the current base point. This is an exploratory move, which we visualize in Figure 3.2b. If none of the solution values corresponding to the surrounding points are better than the one corresponding to the base point, the base point remains unchanged and in the following iteration $2n$ new points surrounding the same base point are evaluated. Those new surrounding points are located half as far away from the base point as those in the previous iteration. This is a shrink move, which we visualize in Figure 3.2c. An iteration is finished

after either an exploratory or a shrink move has been performed. We construct pseudocode for the HJ method and display it in Algorithm 3.

Algorithm 3 The HJ method for a minimization problem

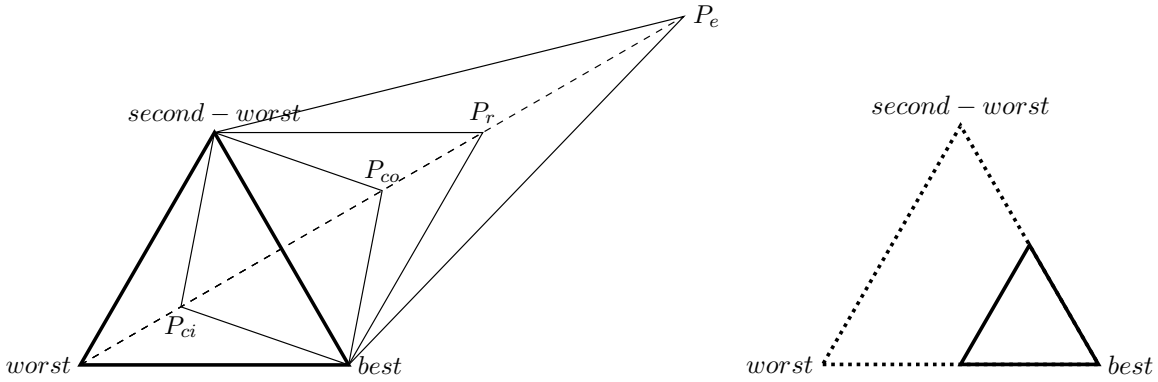
INPUT: Search space of the problem
OUTPUT: Best-found position in the search space P

```
Pick initial pattern size based on search space
Pick random base point P in the search space
y ← f(P)
while Termination criterion not met do
    SHRINK ← TRUE
    for j ← 1, 2n do
        Determine surrounding point P_j
        if f(P_j) < y then
            P ← P_j
            y ← f(P_j)
            SHRINK ← FALSE
        end if
    end for
    if SHRINK then
        Shrink pattern to half its size
    end if
end while
```

The NM method

The NM method (Nelder & Mead, 1965) is one of the most popular classical direct search methods because of its nice analogy with geometry and its ability to quickly locate an optimum by making use of the structure of the search space (Wright, 2012). It appears though that the NM method has not been implemented as meta-optimization method. This might be due to it being more difficult to understand and implement, or to it being more likely to get stuck in local optima compared to other classical direct search methods. The method considers the vertices of an n-dimensional simplex that iteratively moves through the search space, hopefully in the direction of the global optimum. An n-dimensional simplex consists of $n + 1$ vertices, each connected with one another. A one-dimensional simplex is a line segment and a two-dimensional simplex is a triangle. We depict a two-dimensional simplex and the movements it can perform in the NM method in Figure 3.3.

The idea behind the NM method is to update the simplex in each iteration by replacing the worst vertex with a more promising one or to shrink the simplex towards the best point. Each iteration starts with reflecting the simplex away



(a) Reflection, expansion, and contraction

(b) Shrinking

Figure 3.3: A simplex and its movements in a two-dimensional search space

from the worst vertex so that point P_r is created, see Figure 3.3a. P_r replaces the worst vertex if it corresponds to a better solution value than the second-worst vertex. This leads to an updated simplex and ends the iteration unless P_r corresponds to a better solution value than the best vertex. In that case, the simplex is expanded and point P_e is created, see Figure 3.3a. P_r is replaced by P_e if P_e corresponds to an even better solution value. This then marks the end of the iteration.

If P_r did not replace the worst vertex, implying that it did not correspond to a better solution value than the second-worst vertex and hence did not lead to an updated simplex, a contraction is performed. This can either be an outside contraction yielding point P_{co} or an inside contraction yielding point P_{ci}, see Figure 3.3a. An outside contraction is performed if P_r corresponds to a better solution value than the worst vertex and an inside contraction is performed if this is not the case. The contracted point replaces the worst vertex if it corresponds to a better solution value, leading to an updated simplex and ending the iteration. If the contracted point does not correspond to a better solution value, the simplex is shrunk towards the best vertex such that n new vertices are created, see Figure 3.3b. We construct pseudocode for the NM method and display it in Algorithm 4. We also create a flowchart for the NM method which is more detailed than the pseudocode. We refer the interested reader to Appendix B.

The LJ method

The LJ method (Luus & Jaakola, 1973) has been implemented as meta-level optimizer for several base-level algorithms (Rychlicki-Kicior & Stasiak, 2014; Rathore, Chauhan, & Singh, 2015). The auxiliary optimization parameters of

Algorithm 4 The NM method for a minimization problem

INPUT: Search space of the problem
OUTPUT: Best-found position in the search space P_n

Pick $n+1$ random points in the search space: $P_0, P_1, .., P_n$
Order points such that $f(P_0) > f(P_1) > \cdot\cdot > f(P_n)$
while Termination criterion not met **do**
    Reflection to obtain point P_r
    if $f(P_r) < f(P_1)$ **then**
        $P_0 \leftarrow P_r$
        if $f(P_r) < f(P_n)$ **then**
            Expansion to obtain point P_e
            if $f(P_e) < f(P_r)$ **then**
                $P_r \leftarrow P_e$
            end if
        end if
    else
        if $f(P_r) < f(P_0)$ **then**
            Outside contraction to obtain point P_c
        else
            Inside contraction to obtain point P_c
        end if
        if $f(P_c) < f(P_0)$ **then**
            $P_0 \leftarrow P_c$
        else
            Shrink towards point P_n
        end if
    end if
    Order points such that $f(P_0) > f(P_1) > \cdot\cdot > f(P_n)$
end while

DE have been meta-optimized with Local Unimodal Sampling (LUS) (Pedersen & Chipperfield, 2008), a minor adaptation of the LJ method, by Pedersen (2010). The LJ method starts with the selection of a random point in the search space. We refer to this point as the current point. In each iteration, a random point is selected from a range and added to the current point. This range is initially equal to the range of the search space. If the point resulting from the addition corresponds to a better solution value than the current point, it replaces the current point. The search range is then re-centred around the current point and the iteration is finished. If the point resulting from the addition does not correspond to a better solution value than the current point, the search range from which the random points are drawn is decreased in size and the current point remains

unchanged. Because the search range is always re-centred around the current point, it can extend over the boundaries of the search space. If a point is drawn inside the search range that does not lie inside the search space, it is discarded and a new point is drawn without decreasing the search range. We construct pseudocode for the LJ method and display it in Algorithm 5.

Algorithm 5 The LJ method for a minimization problem

INPUT: Search space of the problem
OUTPUT: Best-found position in the search space P

1: Pick random point P in the search space
2: Set search range equal to search space
3: **while** Termination criterion not met **do**
4: Pick random point P_r in search range
5: $P_n \leftarrow P + P_r$
6: **if** $f(P_n) < f(P)$ **then**
7: $P \leftarrow P_n$
8: Re-centre search range around P
9: **else**
10: Decrease search range by a factor 0.95 in each direction
11: **end if**
12: **end while**

LUS differs from the LJ method in the sense that the factor 0.95 used to decrease the search range in line 9 of the algorithm is replaced by $(1/2)^{1/3n}$. This adaptation of the original LJ method has as a result that the method is able to more quickly locate near-optimal solutions in low-dimensional search spaces and is less likely to converge to non-optimal solutions in high-dimensional search spaces.

3.3 Conclusions on Literature Review

In Section 3.1 we discuss several approaches for auxiliary parameter tuning. These approaches are parameter selection, online parameter initialization, and offline parameter initialization. One form of offline parameter initialization is meta-optimization, which requires the implementation of a meta-optimizer. We review several types of meta-optimizers in Section 3.2, namely metaheuristics, race algorithms, and classical direct search methods.

Chapter 4

Solution Approach

In Chapter 3 we have discussed several approaches for tuning auxiliary optimization parameters. We select a tuning strategy for DE in Diamond in Section 4.1 and determine our solution approach this way. In Section 4.2 we construct a flowchart of the new situation in Diamond. We conclude this chapter in Section 4.3.

4.1 Tuning in Diamond

The approaches for tuning auxiliary optimization parameters that we have discussed in Chapter 3 are parameter selection, online parameter initialization, and offline parameter initialization. We discuss the applicability of parameter selection to Diamond in Section 4.1.1 and the applicability of offline parameter initialization to Diamond in Section 4.1.2. Online parameter initialization approaches usually introduce new auxiliary optimization parameters whose values the user must decide upon, worsening the problem that decisions concerning auxiliary optimization settings have to be made by the user. Online parameter initialization does therefore not seem promising in the context of Diamond.

4.1.1 Parameter Selection in Diamond

Parameter selection is the simplest tuning approach. It involves selecting values for the auxiliary optimization parameters relying on conventions and default values. Parameter selection is generally not a good tuning strategy since a default set of parameters might lead to satisfying results on some problems but can fail to yield good results on other problems. However, due to its simplicity, parameter selection is a preferred approach when it can yield good results.

In Chapter 2 we have identified four auxiliary parameters in Diamond that require tuning: the population size NP, the mutation factor F, the crossover constant CR, and the maximum number of function evaluations E. Some conventions and default values regarding DE's auxiliary optimization parameters

NP, F, and CR can be found in literature. We summarize these in Table 4.1. We base the required ranges in this table on Storn and Price (1997) and the common ranges and values on Storn and Price (1997), Lampinen et al. (2005), Rönkkönen, Kukkonen, and Price (2005), and Talbi (2009).

Table 4.1: The three auxiliary optimization parameters of DE

Parameter	Required range	Common range	Common value
NP	$\{4, 5, ...\}$	$\{5n, 5n+1, .., 10n\}$	$\lceil 7.5n \rceil$ †
F	$[0, 2]$	$[0.4, 1]$	0.7 †
CR	$[0, 1]$	$\{[0, 0.2], [0.8, 1]\}$	0.9

† Due to lack of consensus on a proper default value for this parameter, we select the midpoint of its common range.

Although good values for NP and F depend on the problem at hand, on the runtime, and on each other's value, a value of 0.9 appears to be near-optimal for CR in a wide variety of problems, independent of the values selected for NP and F and the runtime, and is therefore generally advised in literature (Rönkkönen et al., 2005; Montgomery, 2009; Talbi, 2009). $CR \in [0, 0.2]$ has been shown to be effective for quite a lot of problems as well and on those problems more efficient than $CR = 0.9$, but research has pointed out that the problems on which $CR \in [0, 0.2]$ is effective and efficient are all separable functions[1] (Lampinen et al., 2005; Rönkkönen et al., 2005). Lots of theoretical test and benchmark functions are separable but the problems in Diamond are definitely not. We can thus apply parameter selection to the auxiliary optimization parameter CR by setting $CR = 0.9$ as default value, and tune the auxiliary optimization parameters NP and F using a different approach.

We also apply parameter selection to the fourth auxiliary parameter of DE in Diamond, the maximum number of function evaluations E. This is because a larger maximum number of function evaluations generally leads to a better solution value but also to more computation time. We have to cut it off somewhere. We decide to terminate the algorithm as soon as $1500 \cdot n$ function evaluations resulting in a feasible solution value have been performed, n being the amount of optimization variables of the problem. Because a number of function evaluations much larger than $1500n$ is generally required to obtain good solution values with DE, we make sure that our solution approach is such that the maximum number

[1] Separable functions are functions depending on multiple variables that can be represented as a combination of functions depending on one variable, such as $f(x, y) = g(x)h(y)$.

of function evaluations can be altered if desired in practice or in further research.

4.1.2 Offline Parameter Initialization in Diamond

Recall from Chapter 3 that the field of offline parameter initialization can be subdivided in the fields of manual tuning, DOE, and meta-optimization. Manual tuning is not a promising approach for Diamond because it requires a lot of input from the user. Performing a DOE and applying meta-optimization make it possible to search for good auxiliary DE parameters for a certain problem without requiring any input from the user during the search. No input from the user is required before the search either (except of course for a datastore, a network, and optimization variables), provided that the method for determining the auxiliary parameter values on which the DOE is performed or the optimizer implemented at the meta-level is parameter-free.

Both performing a DOE with a parameter-free method for determining the auxiliary parameter values on which the DOE is performed and applying meta-optimization with a parameter-free meta-level optimizer are promising approaches for Diamond. They can tackle the problem that we have identified in Chapter 1. Meta-optimization has the advantage over DOE that less combinations of values for the auxiliary optimization parameters have to be evaluated, provided that a good meta-optimizer has been selected, which can result in better solutions and large computational time savings. We therefore select meta-optimization in our solution approach as the tuning strategy for DE's auxiliary optimization parameters NP and F.

Meta-optimization requires the implementation of a meta-level optimizer, which we want to be parameter-free. To decide upon the optimization method that we implement as meta-level optimizer, we introduce the concept of meta-level search spaces. Just like the problems in Diamond induce search spaces resulting from varying the optimization variables, the meta-optimization problems in Diamond induce search spaces resulting from varying DE's auxiliary optimization parameters. We call the search spaces that result from varying NP and F the meta-level search spaces of Diamond.

We can examine the meta-level search spaces in Diamond with the creation of 3-D plots. This can help us in selecting an effective and efficient meta-level optimizer. To examine the meta-level search space of a problem in Diamond, we perform a DOE with grid search. In a grid search each auxiliary optimization parameter is assigned a range. These ranges are split up in equally sized intervals of which the endpoints define the values of the auxiliary parameters that are used

in performing the DOE. In two dimensions the combinations of values can easily be visualized as a grid, hence the name. As we have explained in Chapter 3, performing a DOE is a computationally time expensive task. We hence only examine the meta-level search space of the Mix problem. Optimization runs on this problem take considerably less computation time than optimization runs on the other two problems due to its smaller dimension and smaller evaluation time per point in the base-level search space (see Table 2.1).

To perform a DOE with grid search, we need to select ranges and interval sizes for the auxiliary parameters, and decide how many times each combination of auxiliary parameter values is evaluated. The ranges for the auxiliary parameters we choose such that the common ranges of Table 4.1 are widely spanned, namely $[n, 15n]$ for NP and $[0.1, 1.5]$ for F. We select n as interval size for NP and 0.1 for F so that we obtain 15 values for each auxiliary parameter. We perform a DOE by evaluating each combination of values 25 times to give an indication of how well each combination of values performs.

To approximate the meta-level search space of the Mix problem, we depict the average solution value resulting from each combination of auxiliary optimization parameters in a 3-D plot. We do a log transformation on the solution values to depict the structure of the entire search space more clearly. This is necessary because there exist large differences between the average solution values resulting from different auxiliary parameter settings due to a large amount of penalized soft constraints that can be far from satisfied in some optimization runs. Because solution values indicating profit are negative and log transformation are impossible on negative values, we subtract the lowest solution value of all solution values prior to the log transformation. The resulting plot is an approximation of the meta-level search space of the Mix problem. We depict it in Figure 4.1.

We learn from Figure 4.1 that the meta-level search space of the Mix problem is likely nearly bimodal. With bimodal we mean that there are only two optima in the search space. Of these optima one is located on either side of the discontinuity at $F = 1$. The discontinuation around $F = 1$ can be explained by inspecting the mutation step. Recall from Chapter 2 that mutation is performed using the equation $\overrightarrow{m_i} = \overrightarrow{x_{r_1(i)}} + F \cdot (\overrightarrow{x_{r_2(i)}} - \overrightarrow{x_{r_3(i)}})$, $i = 1, .., NP$. For $F = 1$, $\overrightarrow{x_{r_1(i)}} + F \cdot (\overrightarrow{x_{r_2(i)}} - \overrightarrow{x_{r_3(i)}}) = \overrightarrow{x_{r_2(i)}} + F \cdot (\overrightarrow{x_{r_1(i)}} - \overrightarrow{x_{r_3(i)}})$. This has as a result that the offspring is slightly less diversified after each iteration when DE is performed with $F = 1$ compared to when DE is performed with, say, $F = 0.95$ or $F = 1.05$. Apparently, after a large amount of iterations, this effect becomes quite a problem, especially for small population sizes.

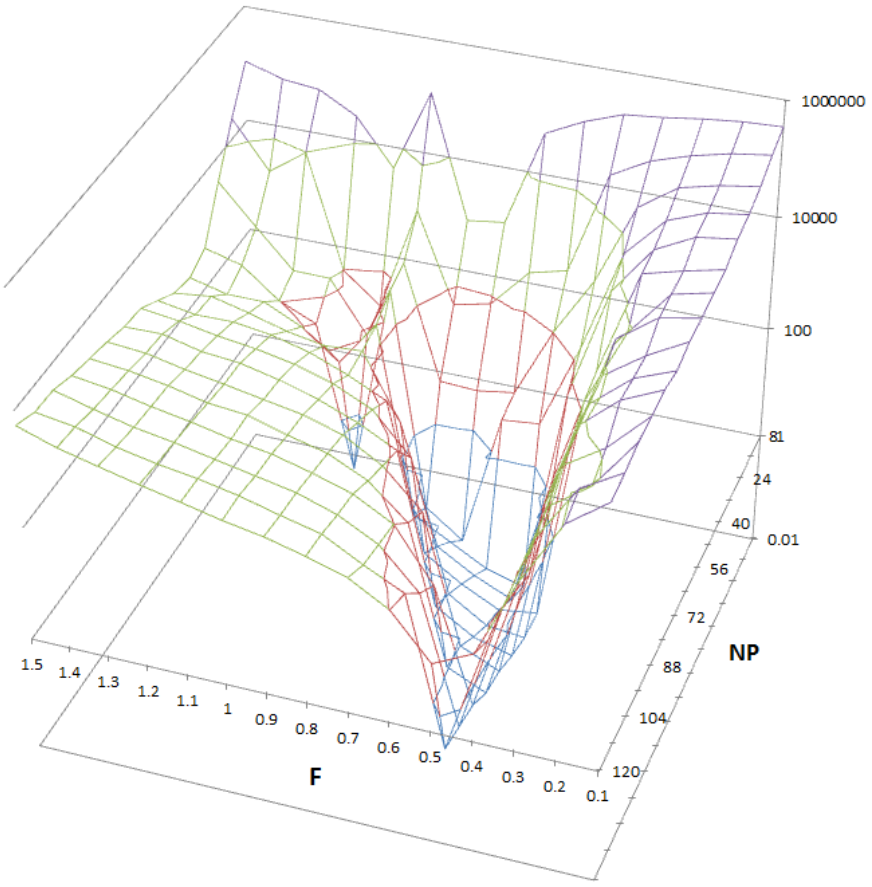


Figure 4.1: Approximation of the meta-level search space of the Mix problem

If we assume that the meta-level search space of the Mix problem is somewhat representative for all Diamond's meta-level search spaces resulting from varying DE's auxiliary optimization parameters NP and F, the NM method seems to be the most promising meta-level optimizer for Diamond. This is because the meta-level search space of the Mix problem appears to be near-unimodal[2], with statistical noise, as long as either $F \in [0, 1]$ or $F \in [1, 2]$. We state in Chapter 3 that it appears that the NM method has not been implemented as meta-optimizer thus far and that this might be due to the fact that it is more likely to get stuck in local optima compared to other classical direct search methods. This is not a problem though if our assumption of near-unimodality of the meta-level search spaces resulting from varying DE's auxiliary optimization parameters NP and F is correct. It is in fact advantageous if the assumption is correct, because the

[2]Unimodal means that there is only one optimum in the search space. With near-unimodal we mean that there can be multiple optima but that these optima do not differ much in solution value from one another.

NM method can make use of the unimodality of a search space to quickly locate an optimum. Another advantage of the NM method is that it can cope well with some (statistical) noise in search spaces. If at times a wrong point is accepted as new vertex in the simplex, it does not mean that the simplex cannot converge in the direction of the optimum anymore.

That there exists an optimum on either side of $F = 1$, we resolve by restricting the domain of the meta-level search space to $F \in [0, 1]$. $F > 1$ is rarely efficient and has not been required for any problem that has successfully been optimized with DE (Rönkkönen et al., 2005; Das & Suganthan, 2011). If the simplex from the NM method is reflected or expanded to a point outside the domain of the meta-level search space, we immediately assign it a very large value so that no computation time is wasted on such points. We further restrict the domain of the meta-level search space to $NP \in [4, 15n]$. This way the values for NP that the meta-level optimizer can encounter satisfy the required range and widely span the common range from Table 4.1.

Just like we evaluated each setting 25 times in performing the DOE, we use the average solution value over 25 base-level optimization runs as the meta-level performance measure. Although performing less base-level optimization runs per setting can speed up meta-optimization runs considerably, we decide not to do so due to DE's stochastic nature, which we back up by depicting the standard deviations of the 25 solution values per point in the approximate meta-level search of the Mix problem in Figure 4.2. Even though neighbouring points in the approximate meta-level search space of Figure 4.1 are not that close to one another, the standard deviations in the points are quite large relative to the differences in average solution values between neighbouring points. We therefore expect the meta-optimizer to be prone to misconvergence if we take the average solution value over a number of base-level optimization runs that is too small and settle with 25. If the meta-level standard deviation space of the Mix problem (Figure 4.2) is representative for all Diamond's meta-level standard deviation spaces, we might be able to speed up our solution approach by making use of the fact that more promising points in the meta-level search space generally correspond to lower standard deviations. We let this remain a topic for further research.

We round the values obtained for NP in a meta-optimization run to the nearest integer for performing the base-level optimization runs. We do this in such a way that it does not influence the locations of the vertices in the meta-level search space. Furthermore, we do not pick the initial simplex vertices randomly

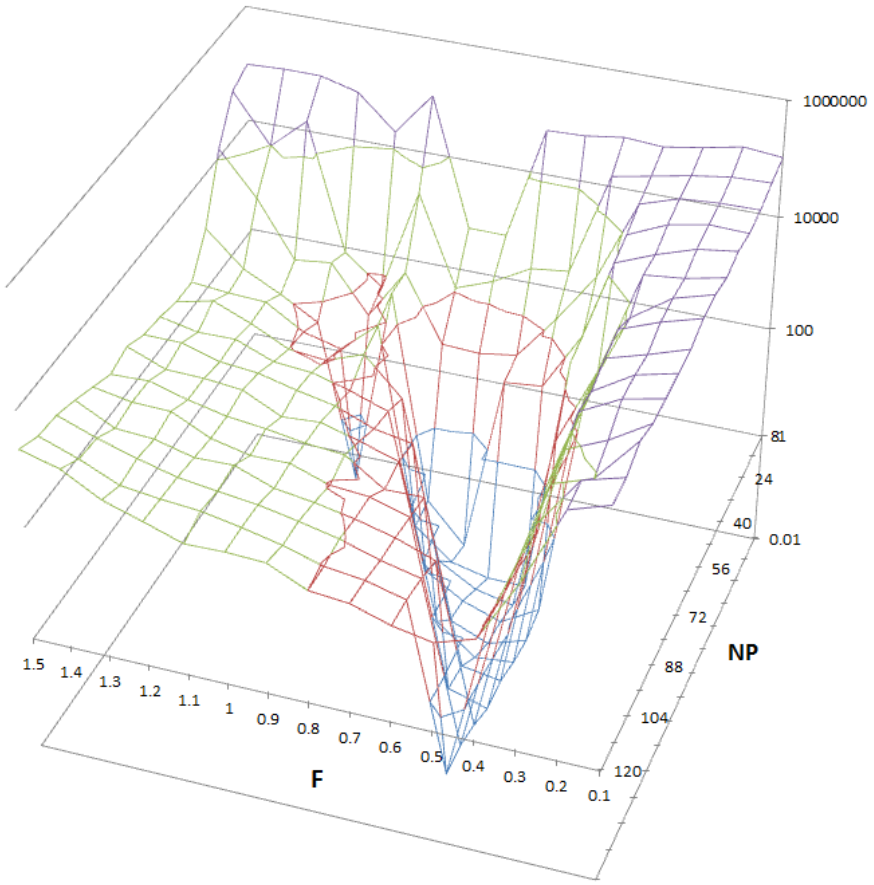


Figure 4.2: Standard deviations of the 25 solution values per point in the approximate meta-level search of the Mix problem

in the entire meta-level search space, but only in that part of the meta-level search space in which the points satisfy the common ranges that we have identified in Table 4.1. This way we hopefully encounter near-optimal settings more quickly.

4.2 The New Situation in Diamond

Our solution approach imposes a new situation in Diamond. We construct a flowchart of this new situation and depict it in Figure 4.3. We base this flowchart on Figures 1.2 and 3.1. There are a couple of boxes in this flowchart that do not have uninterrupted but dashed contour lines. With a dashed contour line we denote that the surrounding box contains an iterative procedure. The colour of the dashed contour line specifies the amount of times the procedure in that box is evaluated when the procedure in the surrounding box is evaluated once.

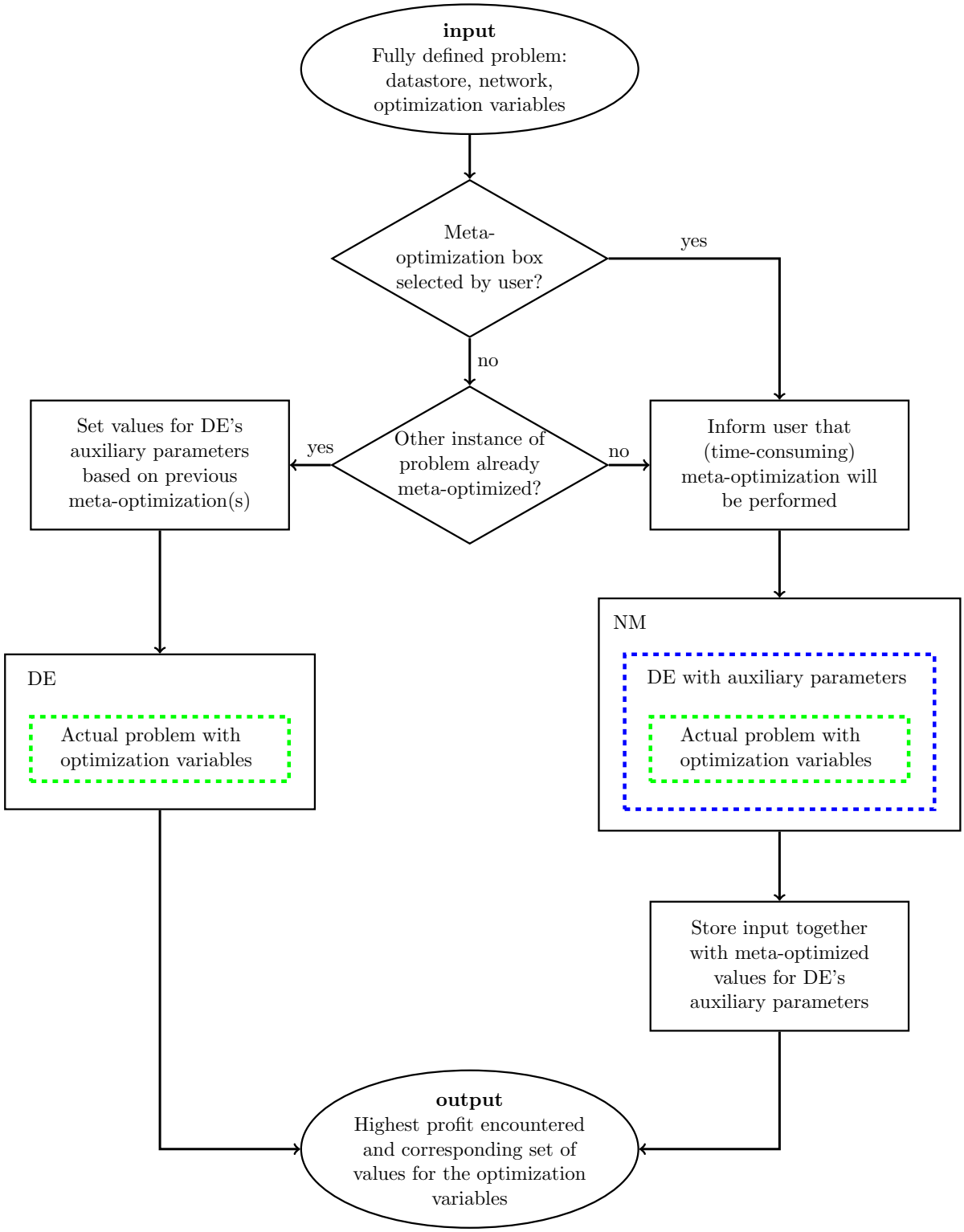


Figure 4.3: A flowchart of the new situation in Diamond

With a green dashed contour line we denote $1500 \cdot n$ evaluations, in line with Section 4.1.1. With a blue dashed contour line we denote $p \cdot q$ evaluations, p being the number of times we evaluate each point in the meta-level search space and q the number of points in the meta-level search space that we evaluate. In line with Section 4.1.2, we let p equal 25. Pedersen (2010) states that a good value for q to aim for in the meta-optimization of metaheuristics is 20 times the number of auxiliary optimization parameters of the base-level optimizer that are being meta-optimized. In our solution approach, this number is 2. We however believe we can use a significantly smaller value for q than 40 because the meta-optimizer we select makes specific use of the structure of the meta-level search space. We decide upon the value $q = 25$. In Chapter 5 we perform a sensitivity analysis on q.

From the perspective of the user, Diamond only changes marginally. The dialog that appears when a user clicks the optimize button is going to be slightly different. After all, in the new situation, the user does not have to set any auxiliary optimization parameters anymore. The part in which the auxiliary parameters currently have to be selected can be replaced by one tick box and a bit of explanation regarding that box. The box should be ticked if the user desires to perform a meta-optimization. In the explanation it should be made clear that ticking this box will drastically (by a factor pq) increase the computation time and that it should be done only when the PC on which Diamond is running can be occupied for a while. It can also state that meta-optimization might be a good idea when standard optimization did not yield satisfactory results. A button in this dialog that the user can press to obtain a rough approximation of the required runtime for a meta-optimization is optional and so is a subfield where the user can adjust the number of base-level function evaluations and thus the runtime.

We indicate in the flowchart that once a problem has been meta-optimized, new instances of that problems are not meta-optimized by default. This implies that good auxiliary optimization parameters are generalizable across problem instances. We test in Chapter 5 whether this is really the case.

4.3 Conclusions on Solution Approach

In Section 4.1 we describe our solution approach for tuning the auxiliary parameters of DE in Diamond. Our solution approach involves parameter selection and meta-optimization. We apply parameter selection to the crossover constant (CR) and the maximum number of function evaluations (E), and apply meta-

optimization to the population size (NP) and the mutation factor (F). We select the NM method as meta-optimizer. We construct a flowchart of the new situation in Diamond in Section 4.2. From the perspective of the user, Diamond only changes marginally.

Chapter 5

Solution Tests

In this chapter we discuss the performance of the solution approach that we have proposed in Chapter 4. To do so, we determine suitable tests in Section 5.1. We present and discuss the results of those tests in Section 5.2. We conclude this chapter in Section 5.3.

5.1 Test Design

We evaluate several performance aspects for our solution approach, namely reliability, robustness, and, efficiency. We formulate questions for the evaluation of these aspects in Table 5.1 and describe how we assess each of them in the context of Diamond.

Table 5.1: Performance aspects we evaluate for our solution approach

Aspect	We evaluate this aspect by asking the question
Reliability	Does the solution approach yield approximately similar results when the input remains unchanged?
Robustness	Does the solution approach yield approximately similar results when the input slightly changes?†
Efficiency	Does the solution approach yield good results in comparison to the current situation or alternative approaches?

† Both the actual problems in Diamond and the meta-level parameter values can be seen as input. We discuss robustness against changes in both types of input.

Reliability

To evaluate the reliability of our solution approach, we apply our solution approach 5 times to each of the problems that we have at our availability. With each of the 3 times 5 resulting auxiliary parameter settings, we perform 20 base-level optimizations to assess whether the different auxiliary parameters we obtain for a problem perform similarly.

Robustness against changes in the actual problems

We can evaluate changes in the actual problems by performing base-level optimization runs on one instance of a problem using the auxiliary parameters that result from multiple meta-level optimization runs on another instance of that problem. Recall from Chapter 2 that a new problem instance arises when an alteration is made to an existing problem. We construct several new instances of the Mix problem and perform 20 base-level optimization runs on those new instances with each of the 5 auxiliary parameter settings we obtain with testing the reliability of the approach. We assume that if the different settings all perform similarly on a new problem instance, they perform well and thus induce robustness. There are more precise approaches to assess the robustness of our solution approach against changes in the actual problems, but we prefer this one for our research because it omits the need for performing meta-optimization runs on the new problem instances.

We construct 7 new instances of the Mix problem. We do this by creating 3 separate adaptations that can be made to the original Mix problem and denoting a new instance with every possible combination of adaptations. We assign a letter to each adaptation so that we can refer to the new instances using letter codes. We depict the changes and their corresponding letters in Table 5.2. An example of a letter code is BA, with which we denote the problem instance in which the buy limit of a raw is significantly decreased and an optimization variable is added. We do not test for robustness against changes in the two technology problems that we have at our availability.

Table 5.2: The adaptations we make to the Mix problem

Brief description of the adaptation	Letter
Decreasing the price of the sale product by 10%	P
Significantly decreasing the buy limit of a raw[†]	B
Adding the input volume of another raw as optimization variable	A

[†] With a significant decrease in buy limit we denote that the supply of the raw in question can be no more than approx. 25% of its advised value in successful optimization runs on the default problem instance.

Robustness against changes in the meta-level parameters

Although we ensured that our solution approach does not require any meta-level input from the user, we did set two parameters that can be seen as input of the meta-level optimizer. For clarity, we summarize all parameters and variables

related to our solution approach that we have introduced in this research in Table 5.3. The two meta-level parameters are p and q. In our research, we perform a sensitivity analysis on q. A sensitivity analysis on p can be performed as well but we do not do so in our research since it would require a lot more meta-optimization runs, which is very computationally time expensive. The same goes for the auxiliary optimization parameters to which we applied the strategy of parameter selection, CR and E.

The most appropriate way to perform a sensitivity analysis on q would be to set several values for q, perform multiple meta-optimization runs with each of those values, and perform base-level optimization runs with the resulting auxiliary optimization parameters to give an indication of how well each different value performs. We however perform the sensitivity analysis on q by keeping track of the average target value corresponding to the best vertex of the simplex in each meta-optimization run we perform to evaluate our approach's reliability. We use the average target value (of all meta-optimization runs we perform on one problem) after q vertices have been evaluated as indication for how well that value of q performs.

Table 5.3: Parameters and variables related to our solution approach

Level (Fig.3.1)	Name	Tuning approach	Robustness test
Meta-level	p †	N/A	None
”	q ‡	N/A	Sensitivity analysis
Base-level	NP	Meta-optimization	None
”	F	”	None
”	CR	Parameter selection	None
”	E §	”	None
Actual problem	-	N/A	Adding a variable, adapting a parameter, and combinations

† p denotes the amount of base-level optimization runs that are performed in the evaluation of one point in the meta-level search space (Chapter 4).

‡ q denotes the amount of points in the meta-level search space that are evaluated in one meta-level optimization run (Chapter 4).

§ E denotes the maximum number of function evaluations resulting in a feasible solution per base-level optimization run (Chapter 2).

Efficiency

To assess the efficiency of our solution approach, we can compare it to the current

situation, as far as we can speak of one at least. The best approach a user unfamiliar with the field of optimization could adopt in the current situation is to select the values for DE's auxiliary parameters that are most commonly advised in literature before performing an optimization. In Chapter 4 we have identified those values to be $NP = \lceil 7.5n \rceil$, $F = 0.7$, and $CR = 0.9$. We can compare our solution approach with the strategy of selecting those values for DE's auxiliary parameters. To this end, we perform 20 base-level optimization runs with the auxiliary parameter values from Table 4.1 for each of the problems that we have at our availability. We compare the 15 times 20 base-level optimization runs we obtain in evaluating the reliability of our approach with the optimization runs representing the current situation to assess whether our solution approach yields better results.

The comparison between our solution approach and the current situation we propose does not take meta-optimization runtime into account. To validate our choice for the NM method as meta-optimizer, we compare it to two straightforward strategies that require a similar amount of computation time. We apply both strategies 5 times to the Mix problem and perform 20 base-level optimization runs with the resulting auxiliary optimization parameters to compare our solution approach with these strategies. The two straightforward strategies are:

- DOE: Arguably the best strategy a user unfamiliar with the field of optimization could adopt in the current situation that is comparable to our solution approach in terms of computation time is to perform a DOE with grid search (as we have done in Chapter 4 to approximate a meta-level search space), in which each auxiliary parameter combination is evaluated 25 times. We restrict the grid to the common ranges of the auxiliary optimization parameters and equally divide the ranges such that 25 auxiliary parameter combinations are evaluated.

- Randomization: The randomization strategy involves randomly selecting 25 distinct auxiliary parameter combinations within the common ranges and evaluating each combination 25 times.

5.2 Test Results

In this section we present and discuss the results of the tests that we describe in Section 5.1. We evaluate the reliability, robustness against changes in the Mix problem, robustness against changes in the meta-level parameter q, and efficiency

of our solution approach.

Reliability

We apply our solution approach 5 times to each of the problems that we have at our availability and depict the resulting values for NP and F in Tables 5.4 – 5.6. It can be seen in these tables that, with a small exception, the meta-optimizations that yield larger values for NP, yield smaller values for F and vice versa. This is an indication that the meta-level search spaces of the Tech and Split problems contain some sort of valley like the one in Figure 4.1. We perform 20 base-level optimization runs using each parameter combination from Tables 5.4 – 5.6 and depict the results hereof in Figure 5.1.

Table 5.4: Results of the meta-optimization runs on the Mix problem

Name of meta-optimization run	M1	M2	M3	M4	M5
Value found for NP	49	35	43	38	45
Value found for F	0.566	0.692	0.618	0.649	0.595

Table 5.5: Results of the meta-optimization runs on the Tech problem

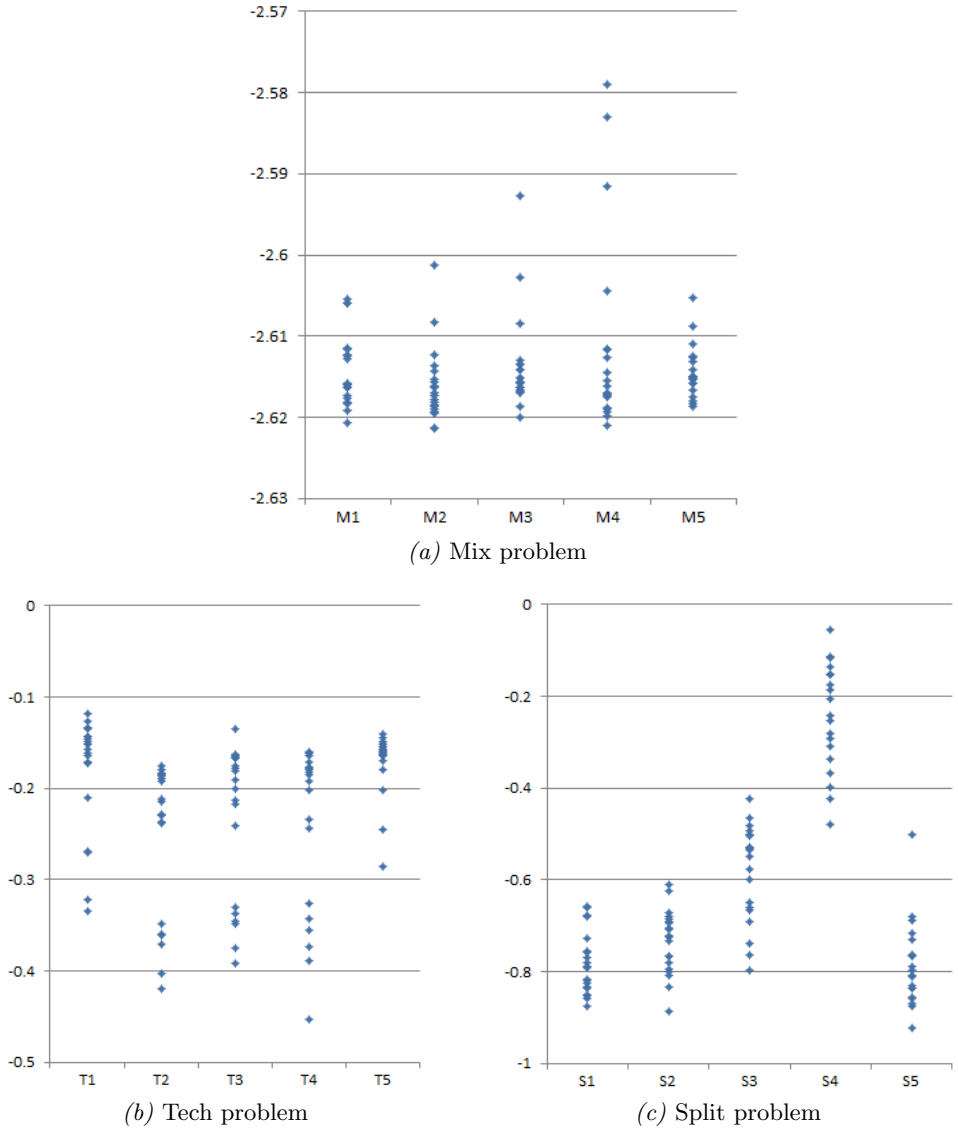
Name of meta-optimization run	T1	T2	T3	T4	T5
Value found for NP	117	58	70	67	113
Value found for F	0.283	0.469	0.415	0.408	0.334

Table 5.6: Results of the meta-optimization runs on the Split problem

Name of meta-optimization run	S1	S2	S3	S4	S5
Value found for NP	39	40	55	156	40
Value found for F	0.483	0.469	0.391	0.310	0.478

Some meta-optimized auxiliary values for the Tech and Split problems show a distinct worse performance than others. These are the values for NP and F that stand out from the rest in Tables 5.5 and 5.6. Apparently the meta-level search spaces of the different problems in Diamond do not look that much alike as we assumed in Chapter 4. In particular, the assumption of unimodality in the meta-level search spaces does not appear to be correct.

That the meta-level search spaces corresponding to the Tech and Split prob-



(a) Mix problem

(b) Tech problem

(c) Split problem

Figure 5.1: Results of base-level optimization runs using meta-optimized values

lems are not unimodal can be due to statistical noise, which is there because of DE's stochasticity and the limited number of base-level optimization runs (p) we perform to evaluate a point in the meta-level search space. In this case, increasing p would increase unimodality of the meta-level search spaces. It would increase meta-optimization runtime too though, which grows linearly with p. It is also possible that the meta-level search spaces corresponding to the Tech and Split problems are different from the one corresponding to the Mix problem in such a way that they contain local optima no matter how large we set p. This induces that our choice of meta-optimizer has not been such a good one, because the NM

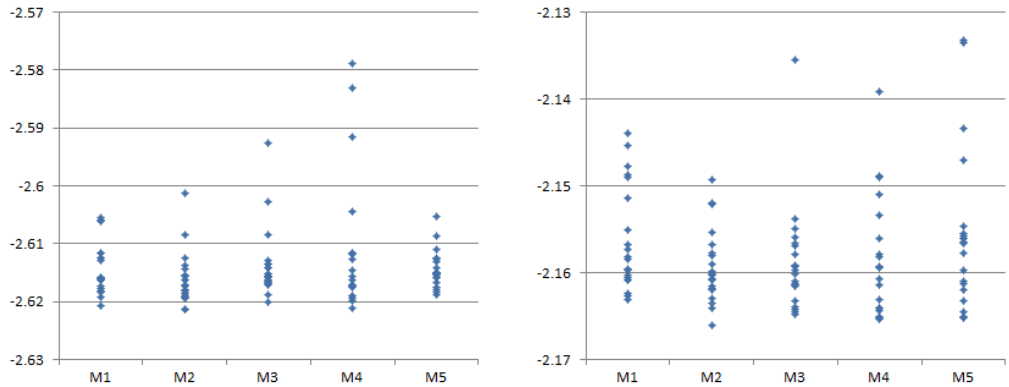
method requires near-unimodality in a search space to perform well.

Furthermore, there is no proof of convergence for the NM method[1]. In fact, there are counterexamples demonstrating that, even in smooth two-dimensional search spaces, the NM method can misconverge (McKinnon, 1998; Han, 2013). A common way to deal with this in practice is to perform a (manual or automated) restart when the simplex becomes ill-conditioned or stagnates (Baudin, 2010; Wright, 2012).

Robustness against changes in the Mix problem

To evaluate the robustness of our solution approach against changes in the Mix problem, we perform 20 base-level optimization runs on the new instances using each of the 5 different auxiliary parameter combinations from Table 5.4 and see how they compare to one another. We depict the results hereof in Figures 5.2 and 5.3.

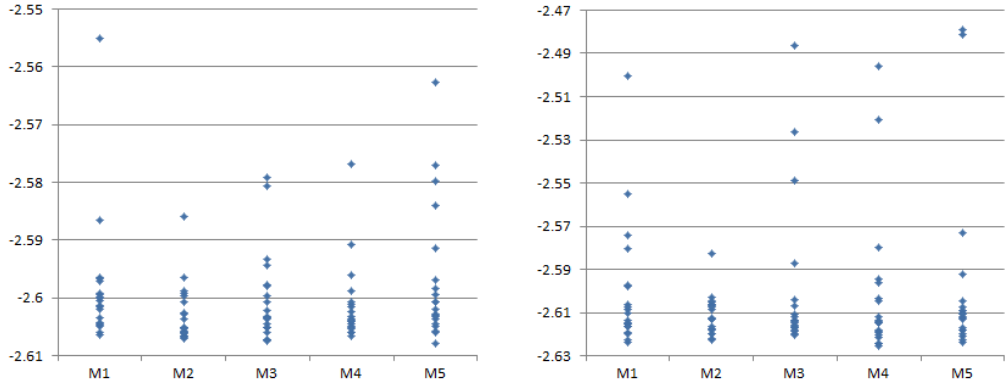
The different auxiliary parameter settings show quite similar performance in the new problem instances. An exception to this is M1, which appears to perform somewhat worse than the other settings in instances where the sales price has been decreased. In Table 5.4 it can be seen that M1 corresponds to the largest value for NP and the smallest value for F. Perhaps the valley in the meta-level search space of the Mix problem becomes smaller or steeper as a result of decreasing the sales price. This could explain why M1 shows a lesser performance than the other auxiliary parameter settings.



(a) Default problem instance *(b)* Problem instance P

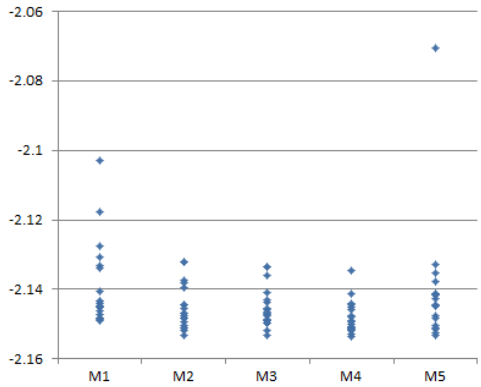
Figure 5.2: Applying the auxiliary parameters obtained in meta-optimization runs M1–M5 to the different instances of the Mix problem, part 1.

[1]Except for one-dimensional, unimodal objective functions (Lagarias, Reeds, Wright, & Wright, 1998).

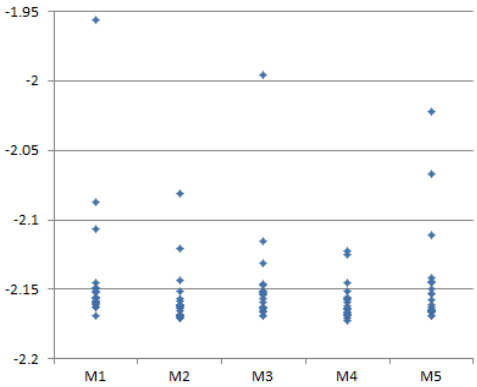


(a) Problem instance B

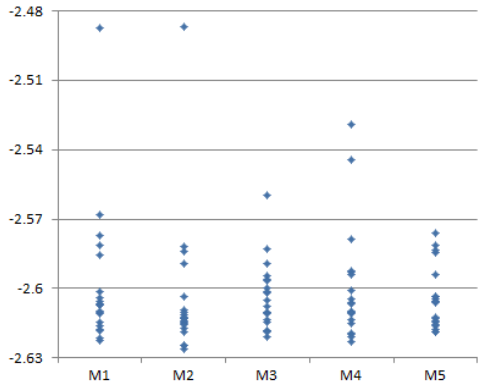
(b) Problem instance A



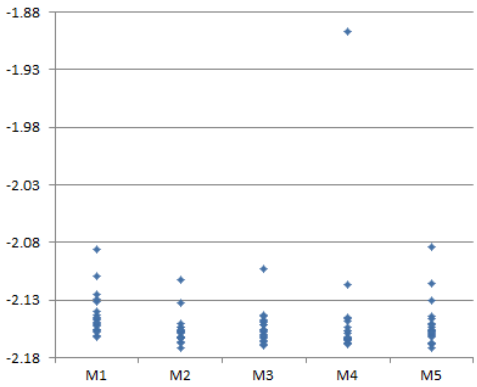
(c) Problem instance PB



(d) Problem instance PA



(e) Problem instance BA



(f) Problem instance PBA

Figure 5.3: Applying the auxiliary parameters obtained in meta-optimization runs M1–M5 to the different instances of the Mix problem, part 2.

In the new problem instances, the variance amongst runs performed with the same settings and the variance between different settings is larger than in the default problem. This is especially true for instances in which an optimization

variable is added. A possible explanation for this is that adding an optimization variable results in adding an extra dimension to the base-level search space. This means that there are a lot more possible solutions, which makes the problem harder to optimize. Since the different settings have resulted in quite similar performance nonetheless, we conclude that our solution approach is quite robust. For the Mix problem at least, and specified to the types of adaptations we have made. More tests have to be performed to acquire knowledge about how robust our solution approach is against changes in the actual problems and thus to what extend meta-optimized auxiliary parameter values are generalizable across problem instances.

Sensitivity analysis on meta-level parameter q

To decide upon a good value for the meta-level parameter q, the number of points in the meta-level search space to evaluate in one meta-optimization run, we perform a sensitivity analysis on this parameter. We depict the results hereof in Figures 5.4 – 5.6. It appears that a value between 25 and 30 is a good choice for q. Increasing the value further is unlikely to result in an improved target value but does increase meta-optimization runtime, which grows linearly with q.

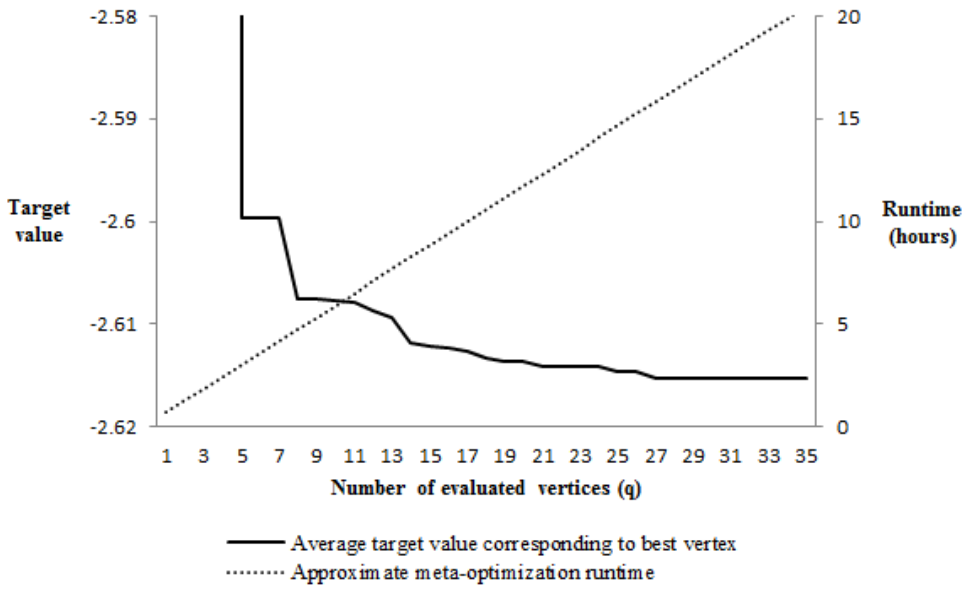


Figure 5.4: Sensitivity analysis on q using runs M1, M2, M3, M4, and M5

Efficiency

To evaluate the efficiency of our solution approach, we compare it with the current

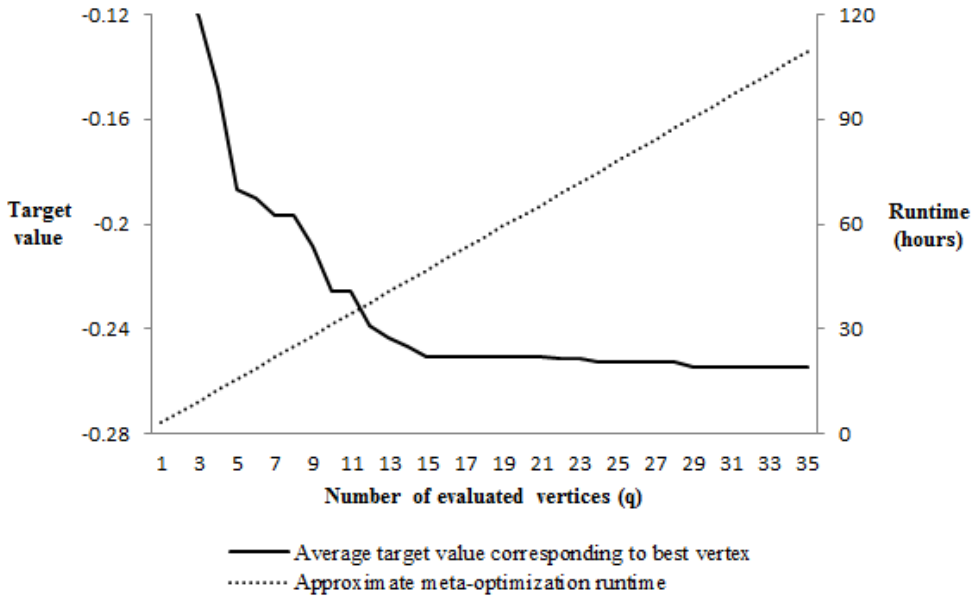


Figure 5.5: Sensitivity analysis on q using runs T1, T2, T3, T4, and T5

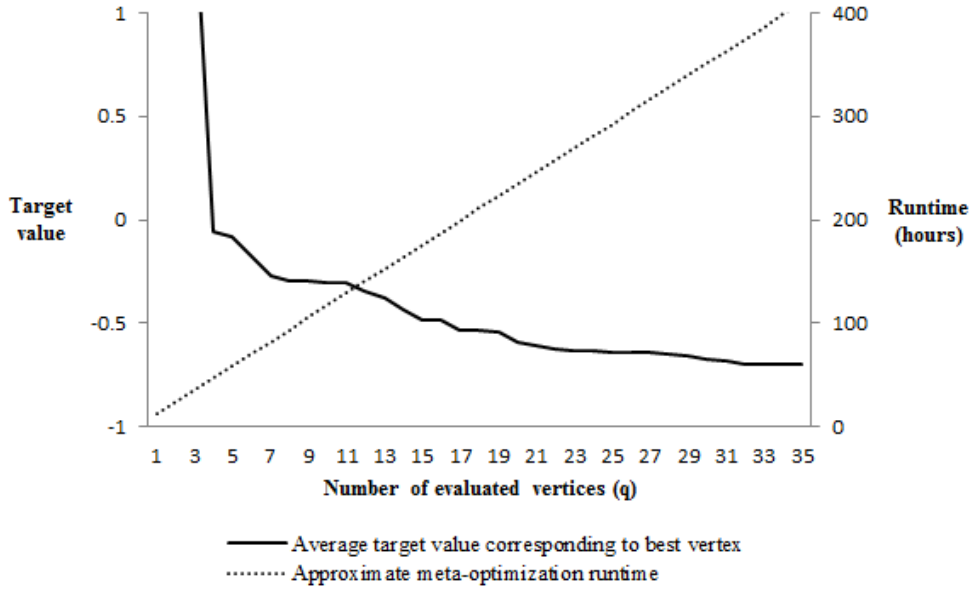
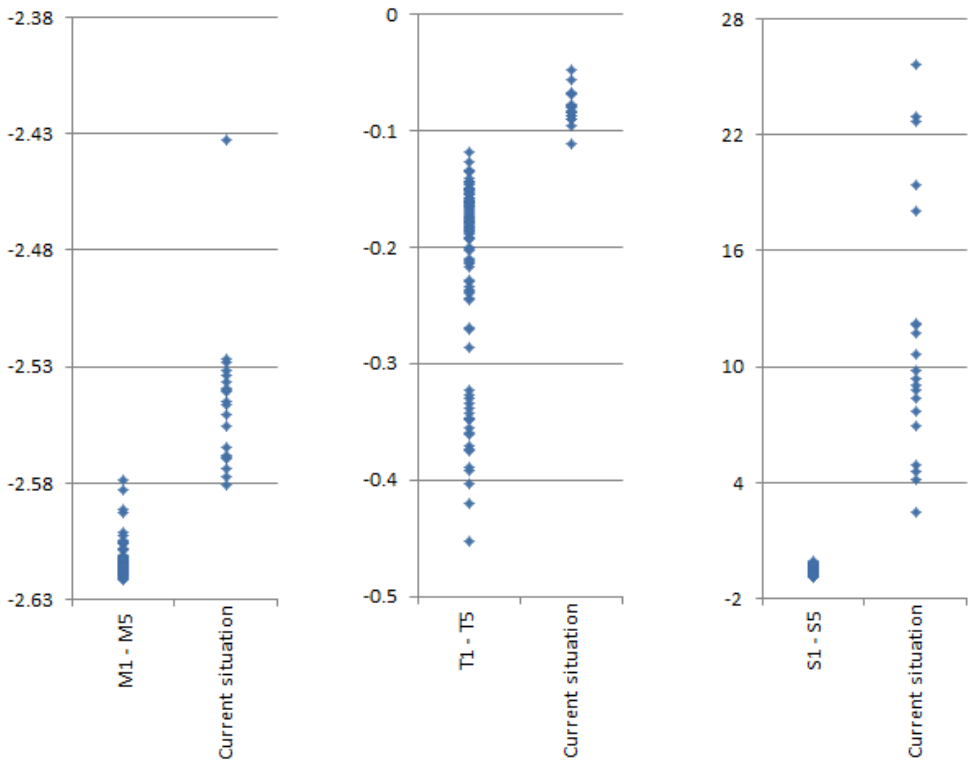


Figure 5.6: Sensitivity analysis on q using runs S1, S2, S3, S4, and S5

situation. In Section 5.1 we have identified this as the approach of selecting the in literature most commonly advised values for DE's auxiliary optimization parameters. We perform 20 optimization runs with the auxiliary parameter values from Table 4.1 on each of the problems that we have at our availability. We depict

the results hereof, together with the results of the base-level optimization runs using the meta-optimized values we have obtained in testing the reliability of our approach, in Figure 5.7.



(a) Mix problem *(b)* Tech problem *(c)* Split problem

Figure 5.7: The results of our solution approach versus the current situation

Because our solution approach has yielded values for NP and F that lead to better performance than the current situation in all three problems, we consider it efficient. However, the comparison on which this conclusion relies ignores meta-optimization runtime. Selecting the default values from Table 4.1 requires no computation time at all, whereas it costs our solution approach $p \cdot q \cdot E$ times the evaluation time of one point in the base-level search space of the problem at hand to obtain good auxiliary optimization parameters. For our research this already comes down to computation times of roughly 17, 70, and, 293 hours for respectively the Mix, Tech, and Split problems. In practice, E likely has to be much larger than $1500n$, leading to even longer computation times. A comparison to the current situation is therefore not a fair approach for drawing conclusions about the efficiency of our solution approach.

As we have explained in Section 5.1, we also compare our solution approach to

two straightforward strategies that require a similar amount of computation time. These two strategies are DOE and randomization. We apply both strategies 5 times to the Mix problem and depict the resulting values for NP and F of these runs in Table 5.7. In this table, R1 – R5 denote the five times we apply randomization. The reason that there is only one DOE entry in the table, is that the resulting best auxiliary optimization parameter combination was the same in each of the 5 times we performed a DOE. We perform 20 base-level optimization runs using each auxiliary parameter combination from Table 5.7 and depict the results hereof in Figure 5.8. In Table 5.8 we summarize the results of the base-level optimization runs on the Mix problem using the auxiliary optimization parameter values obtained with the 4 different strategies.

Table 5.7: Resulting auxiliary parameter values of DOE and randomization

Name of strategy	DOE	R1	R2	R3	R4	R5
Value found for NP	40	60	46	53	43	40
Value found for F	0.7	0.543	0.561	0.536	0.657	0.581

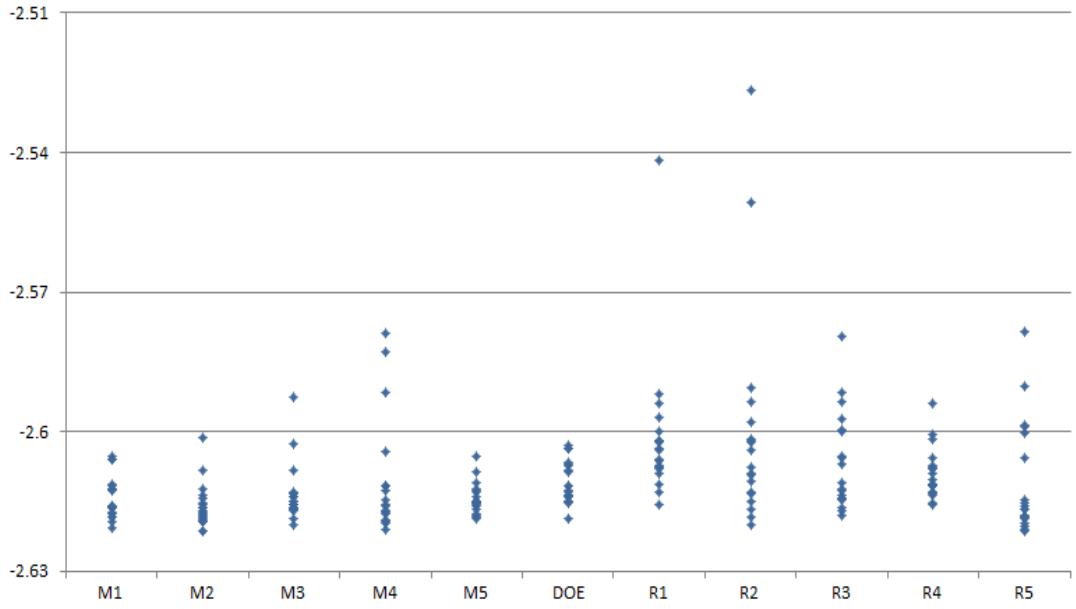


Figure 5.8: Results of base-level optimization runs using the auxiliary parameter values from Tables 5.4 and 5.7

We note that our solution approach does not appear as efficient anymore when we compare it with strategies that are allowed a similar amount of com-

Table 5.8: Base-level optimization results on the Mix problem using the auxiliary optimization parameter values obtained with different strategies

Strategy	#runs	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
Solution approach	100	-2.614	0.0070	-2.621	-2.618	-2.616	-2.613	-2.579
DOE	20	-2.611	0.0045	-2.619	-2.614	-2.612	-2.607	-2.603
Randomization	100	-2.605	0.0149	-2.621	-2.614	-2.608	-2.602	-2.527
Current situation	20	-2.545	0.0316	-2.581	-2.569	-2.546	-2.534	-2.433

putation time. From Table 5.8, we note that the base-level optimization runs using the auxiliary optimization parameter values obtained with our solution approach yield an average solution value that is approximately 0.115%[2] better than the base-level optimization runs using the auxiliary optimization parameter values obtained with DOE and approximately 0.315%[3] better than the base-level optimization runs using the auxiliary optimization parameter values obtained with randomization. Although our solution approach does perform better on the Mix problem than the DOE and randomization strategies (significantly better according to Welch's t-tests at the 0.05 level), the differences in performance are small. How this generalizes to the Tech, Split, and future problems is difficult to predict. Because the range of the auxiliary parameter NP needs to be larger to find satisfactory solutions on the Tech and Split problems, we expect the performance of the DOE and randomization strategies to be somewhat less on these problems than on the Mix problem. On the other hand, DOE and randomization strategies are not dependent on unimodality for their reliability and cannot misconverge like the NM method. They are also easily combinable with approaches that can significantly decrease computation time and increase preciseness, like racing, sharpening, and preemptive fitness evaluations (Smit & Eiben, 2009; Pedersen, 2010).

5.3 Conclusions on Solution Tests

In Section 5.1 we describe several aspects we use to evaluate the performance of our solution approach. These aspects are effectiveness, robustness, and efficiency. In Section 5.2 we depict and discuss the results of the solution tests we have

[2] 95% confidence interval is [0.021%, 0.210%], resulting from a 95% confidence interval regarding the difference in means from Table 5.8 of $[5.38 \times 10^{-4}, 54.9 \times 10^{-4}]$.

[3] 95% confidence interval is [0.191%,0.440%], resulting from a 95% confidence interval regarding the difference in means from Table 5.8 of $[4.97 \times 10^{-3}, 11.5 \times 10^{-3}]$.

performed.

The assumption that we made in Chapter 4 regarding the similarity between meta-level search spaces, and in particular the assumption of unimodality, does not appear to be correct. Because of this, our solution approach is not very reliable. Our solution approach does appear to be quite robust against changes in the actual problem, but more tests in this direction have to be performed. A sensitivity analysis on q, the number of points in the meta-level search space to evaluate in one meta-optimization run, reveals that a value between 25 and 30 is probably a good choice for this parameter.

Our solution approach is efficient compared to the current situation in the sense that the meta-optimized auxiliary parameters yield considerably better results than the most commonly advised auxiliary parameters from literature . In this comparison though, the computation time it takes to obtain the meta-optimized auxiliary parameters has not been taken into account. Compared with two straightforward strategies that require a similar computation time, our solution approach does not appear as efficient anymore. Although it shows a better performance on the Mix problem, the differences are small. Furthermore, as opposed to our solution approach, these strategies are not dependent on unimodality in a meta-level search space for their reliability and cannot misconverge like the NM method.

Chapter 6

Conclusions and Recommendations

In this chapter we conclude our research and give recommendations for further research. In Section 6.1 we present the conclusions. In this section we discuss to what extent the research goal has been achieved and the practical implications of our research for Diamond. We give recommendations for further research in Section 6.2.

6.1 Conclusions

DE is a stochastic optimization method that makes use of a population of solutions in the search space of a problem on which mutation, crossover, and selection take place such that the population hopefully converges in the direction of the global optimum. DE's effectiveness and efficiency depend on several auxiliary optimization parameters. In this research we have developed an approach for determining values for those parameters.

Our solution approach for determining values for the auxiliary optimization parameters of DE in Diamond comprises parameter selection and meta-optimization. Parameter selection involves choosing fixed values for auxiliary parameters relying on conventions and default values. Meta-optimization comes down to treating the search for good auxiliary optimization parameters of a meta-heuristic as an optimization problem in its own right. It hence requires the implementation of an overlaying optimization method. For DE in Diamond we have selected the NM method as overlaying optimization method. The reasons for this were that it is efficient in unimodal search spaces and robust to some (statistical) noise.

By applying parameter selection and meta-optimization, we have constructed an approach for determining values for DE's auxiliary optimization parameters so that they do not have to be set by the user anymore before performing an optimization, which was the goal of our research. But even though it is successful in tackling the research problem, our solution approach is not very promising for

Diamond. In particular, the NM method is not such a good meta-optimizer for DE in Diamond. In a more general sense, the practical suitability to Diamond of the combined parameter selection and meta-optimization approach that we have applied can be questioned.

By selecting the NM method as meta-optimizer, the reliability of our solution approach has become dependent on near-unimodality of the meta-level search spaces. We positively tested this for one of the problems that we have at our availability (the Mix problem), but it does not seem to hold for other problems in Diamond. Our solution approach showed a slightly better performance on the Mix problem than two straightforward strategies that use a similar amount of computation time, DOE and randomization. Our solution approach yields solution values that are 0.021–0.210% better compared to DOE and 0.191–0.440% better compared to randomization. The differences in performance between the two straightforward strategies and our solution approach are hence small and, as opposed to those straightforward strategies, the NM mehtod can misconverge and is difficult to combine with approaches that decrease computation time, such as racing, sharpening, and preemptive fitness evaluations.

A downside of the combined parameter selection and meta-optimization approach in general is that the auxiliary parameter values obtained in a meta-optimization run cannot easily be generalized to different values for the auxiliary parameters to which parameter selection has been applied. This means that the values for NP and F obtained in a successful meta-optimization run are good provided that $CR = 0.9$ and $E = 1500n$ but can be bad if a different value for CR or E is selected, even though the problem remains unchanged. Regarding CR, this is not much of a problem. As we have explained in Chapter 4, literature points out that $CR = 0.9$ is a good choice for most inseparable problems and, if a sensitivity analysis would indicate otherwise, CR can easily be meta-optimized together with NP and F[1]. Regarding E on the other hand, the lack of generalizability of meta-optimized values for NP and F can be quite problematic. If the optimization runtime for future problem instances is known and similar for all future instances, a meta-optimization run can be performed with the value for E set to match this runtime. In Diamond however, it is difficult to predict the available runtime for a future problem instance. The amount of computation time can be spend on the optimization of a new instance depends on the importance of the adaptation that was made and the haste with which a possible alteration

[1]Replacing the NM method by DOE, randomization, or another meta-optimizer is necessary in this case because adding an auxiliary optimization parameter likely results in more complex and multimodal meta-level search spaces.

in product mix or technology settings needs to be proposed.

Furthermore, to offset the disadvantage of meta-optimizations being computationally expensive, meta-optimized auxiliary parameter values have to be generalizable across problem instances. We have performed a limited amount of tests, on only one of the problems that we have at our availability, regarding such generalizability. Although the results hereof were promising, more tests need to be performed to acquire knowledge about the extend to which meta-optimized auxiliary parameter values are generalizable across problem instances in Diamond. Only if these tests yield positive results, and the optimization runtime for future instances of a problem is known and similar, can the combined parameter selection and meta-optimization approach be considered valid and useful.

6.2 Recommendations for Further Research

Although our solution approach comprises the outermost level of Figure 3.1, our main recommendations for further research are on the deeper levels of this figure. This is because we have concluded that the combined parameter selection and meta-optimization approach in general, and our solution approach in particular, is not very promising for Diamond.

The actual problems

Currently, the actual problems in Diamond are treated as black boxes, which has been the reason for using a metaheuristic as optimization strategy. We have already indicated in Chapter 1 that the actual problems are not really black boxes. Research regarding the actual problems and similarities between future problems can potentially be used to develop an optimization strategy tailored for Diamond, which can be a lot more efficient and robust than any general metaheuristic. If the metaheuristic is kept in place, performance gains are also possible from researching the actual problems in Diamond. Analyzing the optimization variables in a network could for example yield that some only have minor influence and do not need to be taken into account in any optimizations.

The base-level

Although we have dismissed online parameter initialization in Chapter 4 because it usually introduces new auxiliary optimization parameters whose values the user must decide upon, we recommend further research in this direction. The new auxiliary parameters in (self-)adaptive DE frequently comprise ranges in which

auxiliary optimization parameters need to lie, distributions according to which auxiliary optimization parameters need to be selected, and probabilities with which an auxiliary optimization parameter needs to be updated. Such auxiliary parameters are generally a lot more robust than those of standard DE. It might therefore be possible to determine values for these auxiliary parameters that can be applied to Diamond in general instead of being suitable for only one problem and perhaps other instances of that problem.

Since DE with a standard set of parameters is generally efficient in the exploration of search spaces but less suitable for exploitation, it can be good to combine DE with a direct search or an approximate gradient method. Another option dealing with the exploration-exploitation trade-off is a form of adaptive DE in which the population size is decreased as the search progresses or re-diversified when the search stagnates (Brest & Zamuda, 2012; Yang, Li, Cai, & Guan, 2015).

The meta-level

We sum up and discuss several recommendations for further research, aimed at improving the quality of our solution approach:

- In Chapter 4 we gave the recommendation of speeding up our solution approach by making use of the fact that more promising points in the meta-level search space generally correspond to lower standard deviations. This recommendation, however, was based on the assumption that the meta-level standard deviation space of the Mix problem we depict in Figure 4.1 is representative for all Diamond's meta-level standard deviation spaces. We can see from Figures 5.1b and 5.7b that this assumption turned out to be incorrect. In the Tech problem, promising points in the meta-level search space can correspond to a much higher variance between base-level optimization runs than less promising points. It is still possible though to alter our solution approach such that the information we have regarding the standard deviations of the points in the meta-level search space that we encounter is incorporated. We can make use of confidence bounds to obtain a more precise comparison between points in the meta-level search space and hopefully dismiss non-promising points more quickly, or we can adapt the meta-level performance measure so that not only the average solution value over p optimization runs but also the standard deviation of these solution values is taken into account (Neumüller et al., 2012).

- Adapting or replacing the meta-optimizer. In Chapter 5 we have suggested that restarts could overcome the problem of misconvergence with the NM

method, which we have encountered in several of our meta-optimization runs. This has as advantage that it only adds computation time to misconverging meta-optimization runs but as disadvantage that it is impossible to predict when and how much extra runtime is required. Replacing the meta-optimizer by DOE, randomization, or an alternative meta-optimizer might be more promising. Alternative meta-optimizers can be classical direct search methods such as LUS and pattern search, or metaheuristcs such as DE itself. As we have explained in Chapter 3, implementing a metaheuristic as meta-optimizer induces the need for meta-meta-tuning. Because all meta-level search spaces are of the same dimension though (2, in our solution approach), meta-meta-tuning is a lot less complex than meta-tuning and can result in well-generalizable meta-level auxiliary optimization parameters (Pedersen, 2010).

- Making use of information that has been obtained in previous meta-optimization runs. For the NM method and other classical direct search methods, starting points can be based on previously meta-optimized auxiliary parameter values instead of selected randomly. For DOE, randomization, and metaheuristics, the search ranges and the initial population can be adjusted based on the results of other meta-optimizations.

The following recommendations for further research can be applied to our solution approach but might also be useful when the combined strategy of parameter selection and meta-optimization is discarded:

- Investigating the use of other termination criteria for DE. In our research, we have terminated base-level optimization runs after a set amount of function evaluations, but there are lots of other possible base-level termination criteria combinable with DE. These can for example be related to the improvement of solution values and to the movements of the population members in the search space (Zielinski & Laur, 2008).

- Ignoring unpromising auxiliary parameter combinations. Auxiliary parameter combinations for which either $NP \notin [4, 15n]$ or $F \notin [0, 1]$ are not evaluated in our solution approach. Perhaps these ranges can be smaller. Another possibility is to not evaluate certain combinations of values for NP and F. The research of Zaharie (2002), who derives an equation that DE's auxiliary optimization parameters should satisfy in order for the population diversity to increase after the crossover and mutation steps, can be used here.

- Parallelizing the algorithms such that (meta-)optimizations can be performed in a shorter amount of time by simultaneously operating on multiple computational resources. For practical purposes, a significant increase of E, the number of function evaluations resulting in a feasible solution per base-level optimization run, is likely required to find satisfactory solution values. This makes our solution approach even more computationally expensive, which can be offset to some extend by parallelization, provided that sufficient computational resources are available.

References

Adenso-Diaz, B., & Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, *54*(1), 99-114. doi: 10.1287/opre.1050.0243

Ali, M. M., & Törn, A. (2004). Population set-based global optimization algorithms: Some modifications and numerical studies. *Computational Operations Research*, *31*(10), 1703-1725. doi: 10.1016/S0305-0548(03)00116-3

Armaou, A., & Kevrekidis, I. (2005). Equation-free optimal switching policies for bistable reacting systems. *International Journal of Robust and Nonlinear Control*, *15*(15), 713-726. doi: 10.1002/rnc.1019

Bäck, T. (1994). Parallel optimization of evolutionary algorithms. In *Proceedings of the international conference on evolutionary computation* (p. 418-427).

Baudin, M. (2010). *Nelder-Mead user's manual*. Orsay, France: Scilab Enterprises.

Birattari, M. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the genetic and evolutionary computation conference* (p. 11-18). San Francisco, CA: Morgan Kaufmann.

Birattari, M. (2009). *Tuning metaheuristics: A machine learning perspective*. Berlin, Heidelberg: Springer-Verlag.

Box, G. E. P., Hunter, J. S., & Hunter, W. G. (2005). *Statistics for experimenters: design, innovation, and discovery*. Hoboken, NJ: Wiley-Interscience.

Brest, J., Greiner, S., Boskovic, B., Mernik, M., & Zumer, V. (2006). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Transactions on Evolutionary Computation*, *10*(6), 646-657. doi: 10.1109/TEVC.2006.872133

Brest, J., & Zamuda, A. (2012). Population reduction differential evolution with multiple mutation strategies in real world industry challenges. *Lecture Notes in Computer Science*, *7269*, 154-161. doi: 10.1007/978-3-642-29353-5_18

Civicioglu, P., & Besdok, E. (2013). A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. *Artificial Intelligence Review*, *39*(4), 315-346. doi: 10.1007/s10462-011-9276-0

Cohen, G., & Meyer, R. (2011). Optimal asymmetrical SVM using pattern search: A health care application. *Studies in Health Technology and Informatics*, *169*, 554-558. doi: 10.3233/978-1-60750-806-9-554

Cortez, P., Rocha, M., & Neves, J. (2001). A meta-genetic algorithm for time series forecasting. In *Proceedings of the workshop on articial intelligence techniques for financial time series analysis, 10th portuguese conference on articial intelligence* (p. 21-31).

Das, S., & Suganthan, P. N. (2011). Differential evolution: A survey of the state-of-the-art. *Transactions on Evolutionary Computation*, *15*(1), 4-31. doi: 10.1109/TEVC.2010.2059031

Gao, S., Yang, H., Sun, Z., Jiang, Y., Zhai, G., & Li, M. (2012). High accuracy temperature control research on charge stable colloidal crystals. In *Recent advances in computer science and information engineering* (Vol. 2, p. 71-77). Berlin, Heidelberg: Springer-Verlag. doi: 10.1007/978-3-642-25789-6_11

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning* (1st ed.). Boston, MA: Addison-Wesley Longman Publishing Corporation.

Han, L. (2013). Nelder-Mead simplex slgorithm: Effect of dimensionality and new implementation [Powerpoint slides]. Retrieved from `https://www.youtube.com/watch?v=r6HZMJGzlDc`

Hooke, R., & Jeeves, T. A. (1961). Direct search solution of numerical and statistical problems. *Journal of the ACM (JACM)*, *8*(2), 212-229. doi: 10.1145/321062.321069

Hu, Z., Xiong, S., Su, Q., & Zhang, X. (2013). Sufficient conditions for global convergence of differential evolution algorithm. *Journal of Applied Mathematics*, *2013*. doi: 10.1155/2013/193196

Jansen, T. (2013). *Analyzing evolutionary algorithms: The computer science perspective*. Berlin, Heidelberg: Springer-Verlag.

Kannan, S., Slochanal, S. M. R., & Padhy, N. P. (2005). Application and comparison of metaheuristic techniques to generation expansion planning problem. *IEEE Transactions on Power Systems*, *20*(1), 466-475. doi: 10.1109/TPWRS.2004.840451

Kolda, T. G., Lewis, R. M., & Torczon, V. (2003). Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, *45*(3), 385-482.

Lagarias, J. C., Reeds, J. A., Wright, M. H., & Wright, P. E. (1998). Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM Journal on Optimization*, *9*(1), 112-147.

Lampinen, J. A., Storn, R. M., & Price, K. (2005). *Differential evolution: A practical approach to global optimization.* Berlin, Heidelberg: Springer-Verlag.

Lewis, R. M., Torczon, V., & Trosset, M. W. (2000). Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*, *124*(12), 191-207. doi: 10.1016/S0377-0427(00)00423-4

Locatelli, M., & Vasile, M. (2014). (Non) convergence results for the differential evolution method. *Optimization Letters*, *9*(3), 413-425. doi: 10.1007/s11590-014-0816-9

Luke, S. (2013). *Essentials of metaheuristics: A set of undergraduate lecture notes.* Department of Computer Science, George Mason University.

Luus, R., & Jaakola, T. (1973). Optimization by direct search and systematic reduction of the size of search region. *AIChE Journal*, *19*(4), 760-766.

Maron, O., & Moore, A. (1994). Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in neural information processing systems, 6* (p. 59-66). San Francisco, CA: Morgan Kaufmann.

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *21*(2), 239-245.

McKinnon, K. I. M. (1998). Convergence of the Nelder-Mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, *9*(1), 148-158.

Meissner, M., Schmuker, M., & Schneider, G. (2006). Optimized particle swarm optimization and its application to artificial neural network training. *BMC Bioinformatics*, *7*. doi: 10.1186/1471-2105-7-125

Mezura-Montes, E., & Palomeque-Ortiz, A. G. (2009). Self-adaptive and deterministic parameter control in differential evolution for constrained optimization. *Studies in Computational Intelligence*, *198*, 95-120. doi: 10.1007/978-3-642-00619-7_5

Montgomery, J. (2009). Differential evolution: Difference vectors and movement in solution space. *IEEE Congress on Evolutionary Computation*, 2833-2840. doi: 10.1109/CEC.2009.4983298

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, *7*, 308-313.

Neumüller, C., Wagner, S., Kronberger, G., & Affenzeller, M. (2012). Parameter meta-optimization of metaheuristic optimization algorithms. *Lecture Notes in Computer Science*, *6927*(1), 367-374. doi: 10.1007/978-3-642-27549-4_47

Pedersen, M. E. H. (2010). *Tuning & simplifying heuristical optimization.* Ph.D. thesis, University of Southampton, School of Engineering Science, Computational Engineering and Design Group.

Pedersen, M. E. H., & Chipperfield, A. J. (2008). *Local unimodal sampling* (Tech. Rep. No. HL0801). University of Southampton, School of Engineering Science, Computational Engineering and Design Group.

Rathore, N. S., Chauhan, D. P. S., & Singh, V. P. (2015). Luus-Jaakola optimization procedure for PID controller tuning in reverse osmosis system. *International Journal of Electrical, Electronics and Data Communication*, *3*(6), 77-80.

Rönkkönen, J., Kukkonen, S., & Price, K. (2005). Real-parameter optimization with differential evolution. In *Proceedings of the IEEE congress on evolutionary computation* (p. 506-513). doi: 10.1109/CEC.2005.1554725

Rychlicki-Kicior, K., & Stasiak, B. (2014). Metaheuristic optimization of multiple fundamental frequency estimation. *Advances in Intelligent Systems and Computing*, *242*, 307-314. doi: 10.1007/978-3-319-02309-0_33

Smit, S. K., & Eiben, A. E. (2009). Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the IEEE congress on evolutionary computation* (p. 399-406). doi: 10.1109/CEC.2009.4982974

Storn, R., & Price, K. (1997). Differential evolution; a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341-359. doi: 10.1023/A:1008202821328

Talbi, E. G. (2009). *Metaheuristics: From design to implementation.* Wiley Publishing.

Van Dijk, T., Mes, M., Schutten, M., & Gromicho, J. (2014). *A unified race algorithm for offline parameter tuning.* Retrieved from `http://beta.ieis.tue.nl/node/2164`

Vesterstrøm, J., & Thomsen, R. (2004). A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 congress on evolutionary computation* (p. 1980-1987).

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *Transactions on Evolutionary Computation*, *1*(1), 67-82. doi: 10.1109/4235.585893

Wright, M. (2012). Nelder, Mead, and the other simplex method. *Documenta Mathematica – Extra Volume ISMP*, 271-276.

Xu, X., & Li, Y. (2007). Comparison between particle swarm optimization, differential evolution and multi-parents crossover. In *Proceedings of the 2007 international conference on computational intelligence and security* (p. 124-127). doi: 10.1109/CIS.2007.113

Yang, M., Li, C., Cai, Z., & Guan, J. (2015). Differential evolution with auto-enhanced population diversity. *IEEE Transactions on Cybernetics*, *45*(2), 302-315. doi: 10.1109/TCYB.2014.2339495

Zaharie, D. (2002). Critical values for the control parameters of differential evolution algorithms. In *Proceedings of the 8th international mendel conference on soft computing* (p. 62-67).

Zielinski, K., & Laur, R. (2008). Stopping criteria for differential evolution in constrained single-objective optimization. *Studies in Computational Intelligence*, *143*, 111-138. doi: 10.1007/978-3-540-68830-3_4

Appendix A

Network Structures

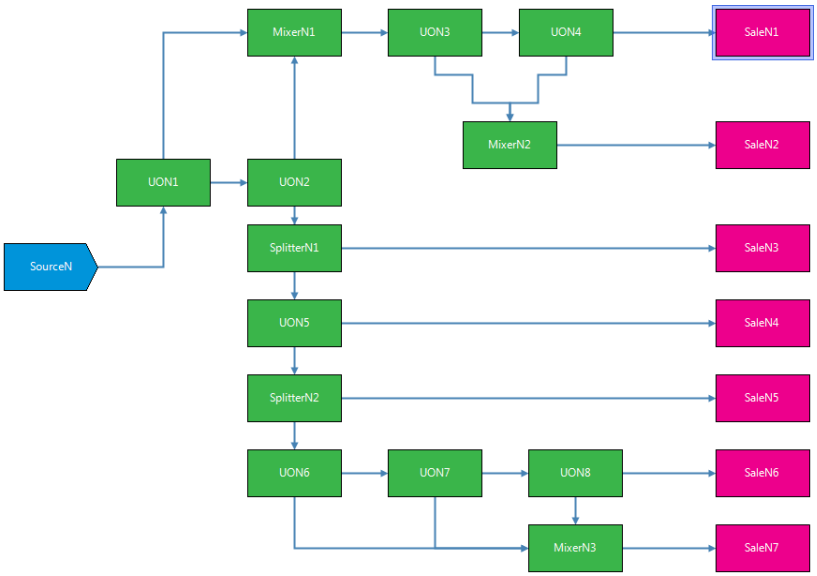


Figure A.1: Network structure of the Split problem – part N of Figure A.2

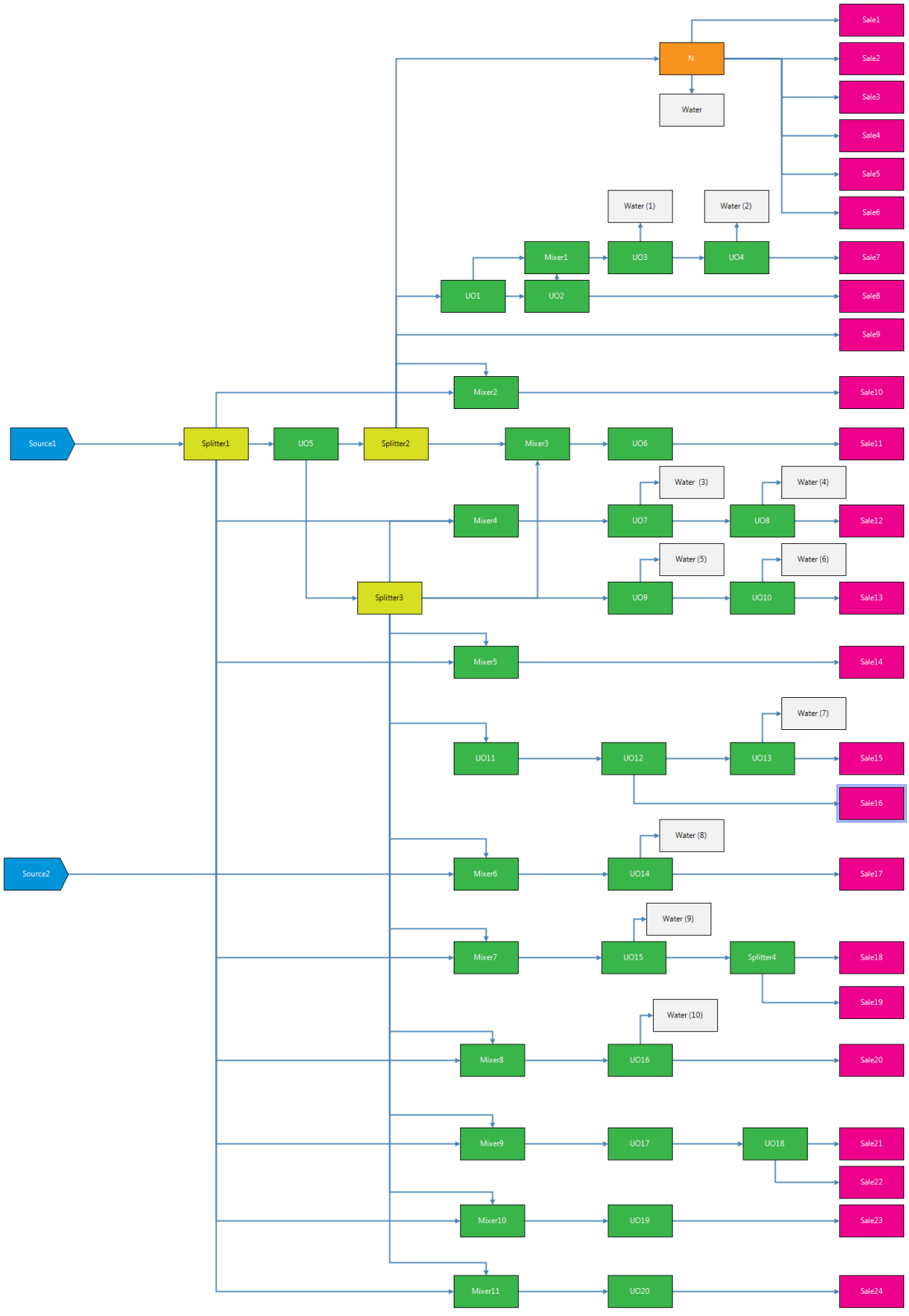


Figure A.2: Network structure of the Split problem

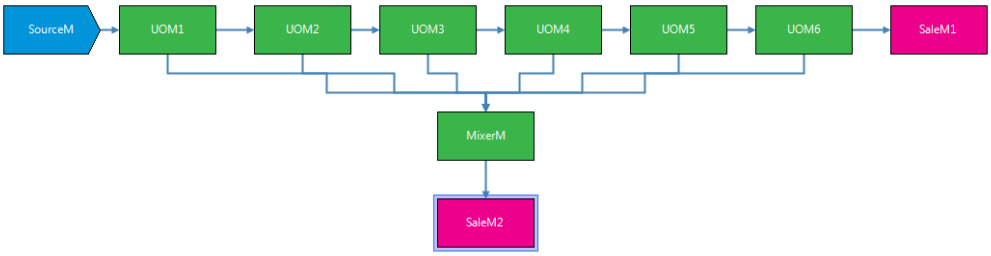


Figure A.3: Network structure of the Tech problem – Part M of Figure A.4

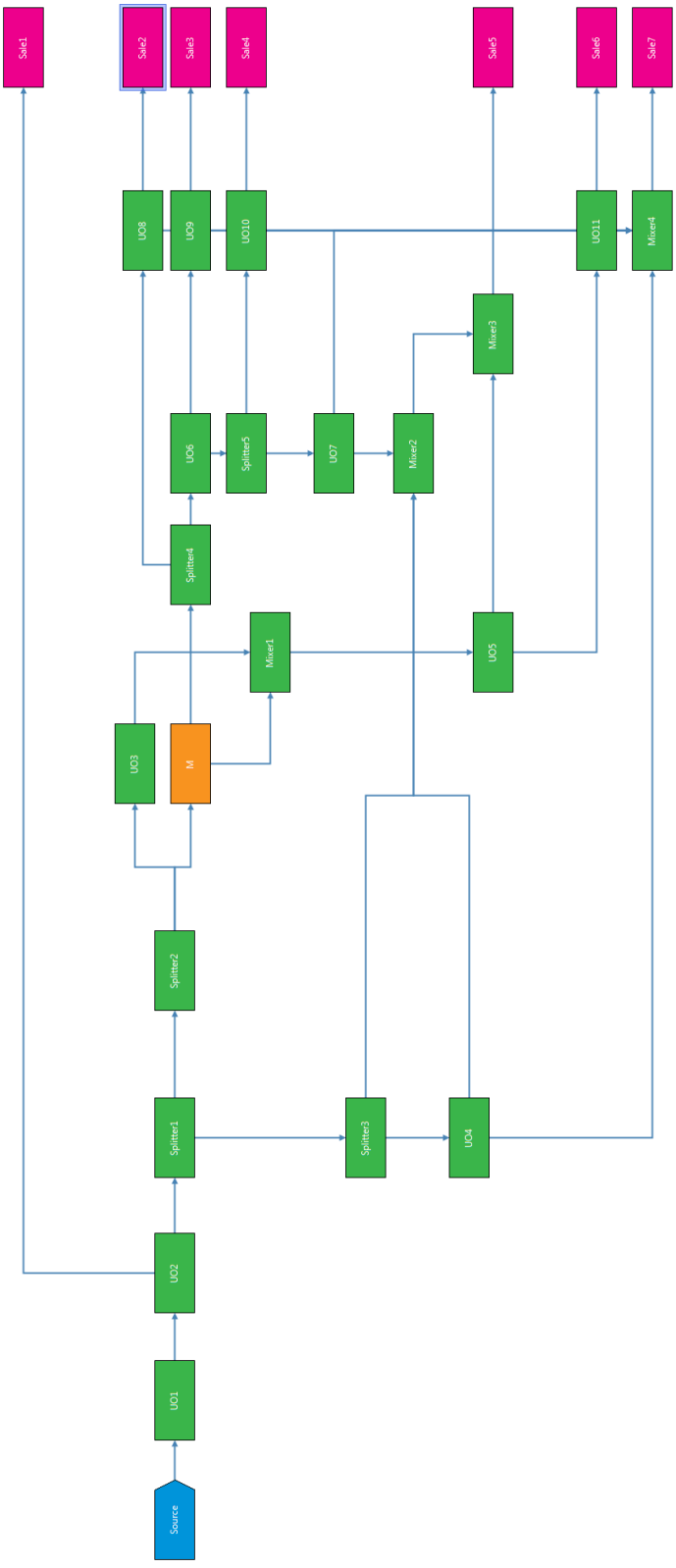


Figure A.4: Network structure of the Tech problem

Appendix B

Flowchart of the NM Method

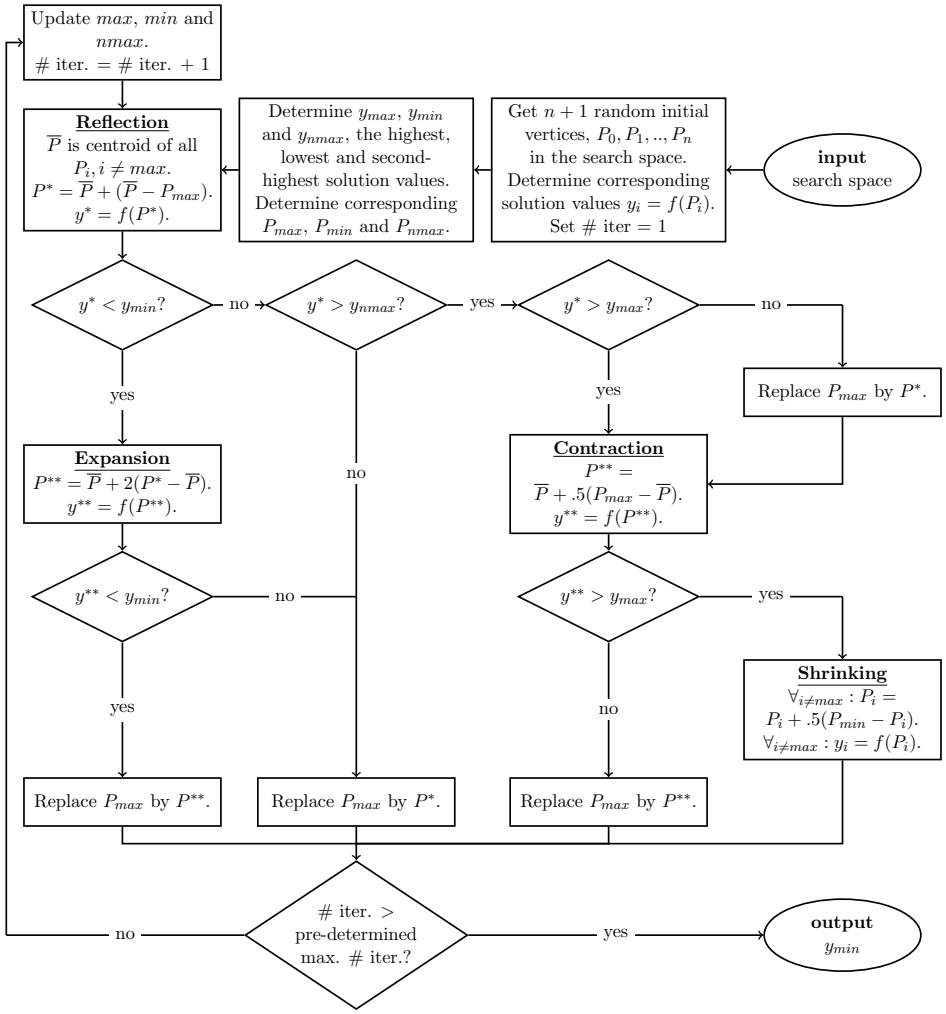


Figure B.1: NM flowchart for a minimization problem of dimension n