Surrogate Models for Discrete Optimization Problems

Dissertation

zur Erlangung des Grades eines

Doktors der Ingenieurwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Martin Zaefferer

Dortmund

2018

Tag der mündlichen Prüfung:
19. Dezember 2018

Dekan:
Prof. Dr.-Ing. Gernot A. Fink

Gutachter:
Prof. Dr. Günter Rudolph, Technische Universität Dortmund
Prof. Dr. Thomas Bartz-Beielstein, Technische Hochschule Köln

Acknowledgments

This thesis was some years in the making. During these years, it profited from the indispensable support (moral and professional) of many people. Hence, I express my gratitude to my

- supervisor Prof. Dr. Thomas Bartz-Beielstein, who triggered my interest into this research field. He has since been a strong support and a source of motivation for my scientific work.
- supervisor Prof. Dr. Günter Rudolph. Our discussions always provided new impulses, and opened up questions that I might not have touched on otherwise.
- mentor Prof. Dr. Gabriele Kern-Isberner, for helpful discussions on work-life-balance, ideation, and systematic approaches of research.
- colleagues, who often discussed potential applications and connected research (especially Oliver, Jörg, Martina, Andreas, Frederik), gave support for the computational experiments and related technical aspects (Sebastian, Oliver, Andreas), proofread drafts (Frederik), gave feedback on my articles and presentations, helped me to see my research from a different view, and/or helped to reduce my workload in the final phase of my thesis.
  Thanks to all of you: Beate Breiderhoff, Sowmya Chandrasekaran, Andreas Fischbach, Oliver Flasch, Martina Friese, Lorenzo Gentile, Dimitri Gusew, Dani Irawan, Christian Jung, Patrick Koch, Sebastian Krey, Olaf Mersmann, Steffen Moritz, Boris Naujoks, Quoc Cuong Pham, Stefanie Raupach, Margarita Rebolledo, Frederik Rehbach, Viktoria Schaale, Jörg Stork, Katya Vladislavleva, and Svetlana Zakrevski.
- parents Monika and Rolf Zaefferer, for the patient support and encouragement during these years.

Contents

1 Introduction **9**
- 1.1 Incentive . . . 9
- 1.2 Objective . . . 10
- 1.3 Synopsis . . . 11
- 1.4 Publications . . . 12

I Background and State of the Art 15

2 Continuous Methods for Expensive Black-box Optimization **16**
- 2.1 Evolutionary Algorithms . . . 16
- 2.2 Surrogate Model-based Optimization . . . 20
- 2.3 Introduction to Kriging . . . 22
  - 2.3.1 Kriging Model . . . 22
  - 2.3.2 Kriging Prediction . . . 24
  - 2.3.3 Uncertainty Estimate . . . 24
  - 2.3.4 Nugget Effect . . . 25
- 2.4 Efficient Global Optimization . . . 27

3 A Survey of Surrogate Models for Discrete Problems **29**
- 3.1 Strategies for Dealing with Discrete Structures . . . 29
- 3.2 Naive Approach . . . 30
- 3.3 Customized Models . . . 31
- 3.4 Inherently Discrete Models . . . 32
- 3.5 Mapping . . . 32
- 3.6 Feature Extraction . . . 34
- 3.7 Kernel-based Models . . . 34
- 3.8 Alternative Models . . . 36
- 3.9 Conclusions . . . 37

4 Kernels: Terminology and Definitions **38**
- 4.1 Motivation: The Kernel Trick . . . 38
- 4.2 Similarity Measures . . . 41
- 4.3 Dissimilarity Measures . . . 41

4.4 Definiteness 43
4.4.1 Definiteness of Matrices 43
4.4.2 Definiteness of Kernels 44
4.5 Conditioning of Matrices 45

II Main Contributions 46

5 Kriging-based Combinatorial Optimization 47
5.1 Kriging for Combinatorial Data 47
5.2 Combinatorial Efficient Global Optimization 49
5.3 Proof of Concept 51
5.3.1 Case Study: Kriging with Hamming Distance 51
5.3.2 Results and Analysis 55
5.3.3 Case Study: Changing the Distance Function 58
5.4 Conclusions 59

6 How to Find the Right Kernel 61
6.1 Selecting Kernels 63
6.1.1 Random Selection 63
6.1.2 Selection with Correlation Measures 64
6.1.3 Selection with Maximum Likelihood Estimation 65
6.1.4 Selection with Cross-Validation 66
6.2 Combining Kernels 66
6.2.1 Weighted Sum via Linear Regression 67
6.2.2 Weighted Sum via MLE 68
6.2.3 Weighted Sum via Stacking 69
6.2.4 Superposition of Kriging Models 71
6.3 Experimental Validation 72
6.3.1 Experimental Setup 72
6.3.2 Results and Discussion 73
6.4 Conclusions 80

7 Definiteness of Distance-based Kernels: An Empirical Approach 82
7.1 Dealing with Definiteness 83
7.2 Probing Definiteness 83
7.2.1 Random Sampling for Probing Definiteness 84
7.2.2 Optimization for Probing Definiteness 85
7.3 Experimental Validation 86
7.3.1 Random Sampling 86
7.3.2 Optimization 86
7.3.3 Other Distance Measures 87
7.4 Observations and Discussion 87
7.4.1 Sampling Results 87

7.4.2 Optimization Results 90
7.4.3 Verification: Impact on Model Quality 90
7.4.4 Additional Simple Examples 92
7.5 Conclusions 95

8 Kriging-based Optimization with Indefinite Kernels 97
8.1 Correcting Indefinite Kernels 98
8.1.1 Spectrum Transformation: Kernel 98
8.1.2 Spectrum Transformation: Distance 101
8.1.3 Spectrum Transformations: Condition Repair 102
8.1.4 Nearest Matrix Approach 104
8.1.5 Feature Embedding 104
8.2 Indefiniteness and Non-Stationarity 105
8.3 A Simple Example 106
8.4 Experimental Validation 108
8.5 Observations and Discussion 110
8.5.1 Preliminary Observations: Modeling Failures 110
8.5.2 Results: Real-valued Problems 111
8.5.3 Results: Permutation Problems 115
8.6 Conclusions 119

9 Applications and Extensions 120
9.1 A Surrogate for Symbolic Regression 120
9.1.1 Related Work 121
9.1.2 Bi-level Problem Definition 122
9.1.3 Kernels for Bi-level Symbolic Regression 124
9.1.4 Case Study 127
9.1.5 Summary 135
9.2 Modeling Hierarchical Variables 136
9.2.1 Kernels for Hierarchical Search Spaces 137
9.2.2 Previous Experiments 143
9.2.3 Future Experiments 144
9.3 Simulation for Test Function Generation 144
9.3.1 Motivation 145
9.3.2 Related Approaches 147
9.3.3 Simulation-based Test Function Generator 149
9.3.4 Illustrative Examples 154
9.3.5 Protein Landscape Application 156
9.3.6 Summary 161
9.4 Conclusions 162

III Closure **163**

10 Final Evaluation and Outlook **164**
10.1 Summary . . . 164
10.2 Advice for Practitioners . . . 166
10.3 Future Work . . . 168

Appendices **170**

Appendix A Analysis Tools **170**
A.1 Visual Analysis of Results . . . 171
A.2 Statistical Analysis of Results . . . 171
A.2.1 Single Factor: Comparing Distinct Algorithms . . . 171
A.2.2 Multiple Factor: Algorithms and Parameters . . . 176

Appendix B Employed Kernels and Distances **178**
B.1 Real Vectors . . . 178
B.2 Discrete Vectors and Sequences . . . 179
B.3 Permutations . . . 180
B.4 Signed Permutations . . . 184
B.5 Trees . . . 184
B.6 Hierarchical Search Spaces . . . 185

Appendix C Variation Operators **186**
C.1 Permutation Operators . . . 186
C.1.1 Mutation . . . 186
C.1.2 Recombination . . . 187
C.2 Binary String Operators . . . 187
C.2.1 Mutation . . . 188
C.2.2 Recombination . . . 188
C.3 Tree Operators . . . 188

Appendix D Additional Figures **189**
D.1 Box Plots: Multi-Kernel Experiments . . . 189
D.2 Kriging with Indefinite Kernels: Examples . . . 195

Bibliography **199**

Notation **225**

List of Symbols **226**

Index **234**

Intelligence takes chances with limited data
in an arena where mistakes are not only possible
but also necessary.
Frank Herbert; Chapterhouse: Dune (1985)

Chapter 1

Introduction

In real-world optimization, it is often expensive to evaluate the quality of a candidate solution. The costs may be due to run-time of a complex computer simulation, time required for a laboratory experiment, expensive material for building a prototype, or cost of operator work hours. In these cases, it is desirable to evaluate as few candidate solutions as possible.
Surrogate Model-Based Optimization (SMBO) techniques try to achieve this by shifting the evaluation load from the expensive experiment to a surrogate model. This model is expected to be a cheap-to-compute abstraction of the real-world problem.

1.1 Incentive

Surrogate modeling has been firmly established in the field of continuous optimization with real-valued parameters [136, 257, 232, 137, 26]. Nevertheless, many optimization problems are not continuous. Rather, search spaces are often spanned by discrete variables or are based on complex structures such as graphs. In fact, one of the first experiments with evolution strategies dealt with an expensive and discrete optimization problem: Schwefel's optimization of a two-phase jet nozzle [225, 149]. Here, the experimental restrictions imposed a discrete search space: 239 available nozzle segments could be combined to form different nozzle shapes [225].
More recent application examples can be found in the fields of mechanical engineering [254, 11, 231, 42, 244, 245], aerospace engineering [131], chemical engineering [104, 118, 108], bioinformatics [71, 212], operations research [85, 121, 191, 192, 190, 174] or data science [235] and algorithm tuning [126, 128, 34, 33, 241, 246, 80, 125, 38, 198, 72, 161]. In these problem domains, combinatorial data structures (e.g., categorical variables, sequences, strings, or graphs) have to be optimized, and the evaluation procedures are often very expensive.
Many of these examples are quite recent developments; others are not relying on surrogate models at all. Several works in recent years suggested that this field has not been investigated in depth and poses a challenge to SMBO algorithms [257, 232, 179, 137, 9, 110]. So what is the reason for this, and what is really missing? Clearly, a large

variety of model-free optimization techniques for black-box problems is readily available, e.g., evolutionary algorithms or other metaheuristic optimization algorithms [43]. It seems that we lack suitable surrogate models to apply techniques like evolutionary algorithms to expensive, discrete problems. Moreover, the interdisciplinary nature of this research field (engineering applications, combinatorial optimization, machine learning, evolutionary computing) causes additional complications in practice.
It is the aim of this work to fill this void. Based on the state of the art in surrogate modeling, we attempt to establish the missing links in combinatorial, discrete, and mixed variable SMBO. We focus on developing or improving suitable models.

1.2 Objective

The main objective of this thesis is to develop methods for solving the optimization problem

$$\min_{x \in \mathcal{X}} f(x),$$

where $\mathcal{X}$ is a nonempty set (the search space). An element x of that set represents a candidate solution of the optimization problem. The objective function $f(x)$ is usually assumed to be expensive (with respect to money, time or material resources) and a black-box. Here, the term "black-box" implies that knowledge about the objective function $f(x)$ can only be obtained by evaluating it, no further prior knowledge is available. At the same time, the cost of the overall optimization procedure is dominated by the number of evaluations of $f(x)$. This causes the main complexity of this problem type: the black-box nature of the problem necessitates a certain amount of function evaluations, yet the cost of these evaluations imposes a strict limit on their number. This motivates the use of surrogate models.
An important premise for this thesis is the nature of the solutions x. Surrogate modeling is well-established for $x \in \mathbb{R}^m$, where m indicates the dimensionality of the search space, that is, the number of real-valued parameters. We aim for a more general case, where x is allowed to have a more complex structure. Most importantly, it may be of a combinatorial or discrete nature, e.g., permutations, strings, graphs, or mixtures of representations. In this case, relatively few approaches have been investigated in the literature. There is only one restriction required for most of the methods described in this thesis. A kernel function $\mathrm{k}(x, x')$ has to be available, which comprehends some measure of similarity or distance. This implies that $\mathrm{k}(x, x')$ can be computed for any $x, x' \in \mathcal{X}$.
If such a kernel is available, the methods developed in this thesis are applicable to any kind of continuous, mixed, discrete, or combinatorial solution representation. In particular, this thesis deals with the following solution representations:

- Continuous variables are used in various illustrative examples, and are considered in Chapter 2.
- Permutations are used as a frequent test case in Chapters 5 to 8.

- Binary variables are used in experiments in Chapter 5.
- Strings are briefly considered in Section 7.4.4.
- Tree structures are considered in Section 9.1, in the context of genetic programming and symbolic regression.
- Graphs that encode hierarchical dependencies of variables are considered in Section 9.2.
- Categorical variables are considered in an application in Section 9.3.

1.3 Synopsis

A graphical overview of this thesis is given in Fig. 1.1. In the following, we extend this overview with more details.
Firstly, we provide the required background on black-box optimization methods and surrogate modeling in Chapter 2. This initial overview mostly deals with methods from the continuous domain. Chapter 3 extends this with an overview of surrogate models in the discrete domain. This provides a review of the state of the art, to give a clear indication of what other approaches exist. Our further work heavily relies on kernels, which are discussed in Chapter 4.
Based on the introduced background, Chapter 5 deals with the core methods of this thesis: a Kriging model for discrete search spaces based on appropriate distances or kernels. Kriging is a frequently used surrogate model for continuous problems that regresses or interpolates data based on measures of correlation, or kernel functions [93].

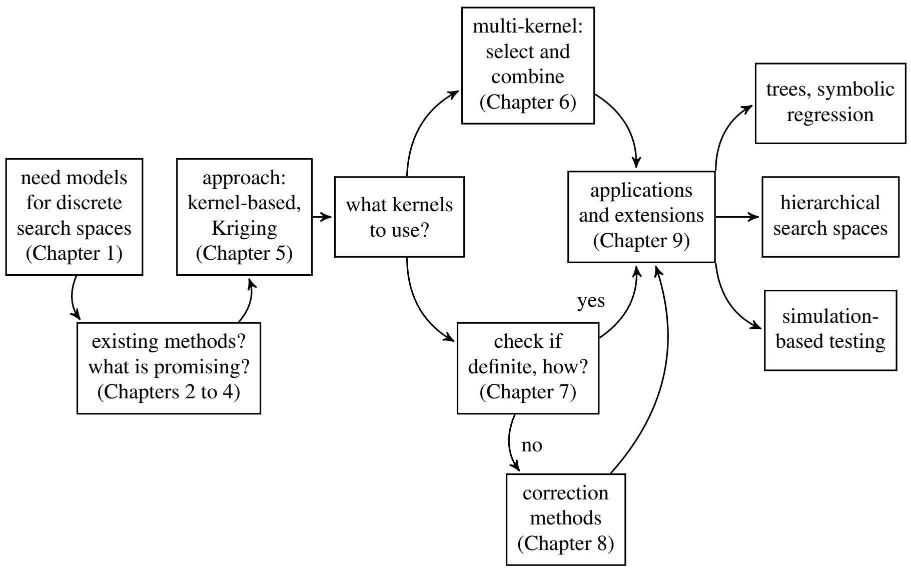

multi-kernel: select and combine (Chapter 6)
trees, symbolic regression
need models for discrete search spaces (Chapter 1)
approach: kernel-based, Kriging (Chapter 5)
what kernels to use?
applications and extensions (Chapter 9)
hierarchical search spaces
yes
simulation-based testing
existing methods? what is promising? (Chapters 2 to 4)
check if definite, how? (Chapter 7)
no
correction methods (Chapter 8)

Figure 1.1: *A brief overview of this thesis, following the main lines of the investigation.*

The intuitive idea is to change the underlying kernel, thus adapting Kriging to the discrete data types investigated in this thesis. The rationale behind this is explained, and a first set of experiments is investigated as a proof of concept.

As the experiments show, it is not always clear which kernel to use for a specific application. Hence, we investigate methods that select or combine kernels in Chapter 6. These multi-kernel methods mostly employ the following concepts: linear relationships between kernels and observations, likelihoods, and cross-validation. Our results indicate that it is crucial to employ multi-kernel methods, to avoid poor algorithm performance.

However, not every kernel fulfills the requirements of the modeling techniques. Importantly, kernels are expected to be definite. An indefinite kernel may deteriorate the accuracy of the model. Even worse, an indefinite kernel may sometimes produce no feasible model at all. Hence, it is important to know whether a kernel is definite. In Chapter 7, we propose a novel, empirical approach to that end. A search procedure either tries to find indefinite cases by random sampling or by optimizing a critical eigenvalue.

Once a kernel is determined to be indefinite, appropriate counter-measures have to be taken to avoid performance losses. Chapter 8 demonstrates how Kriging models can be enabled to deal with indefinite kernels. Most importantly, transformations of the eigenspectrum are employed to correct definiteness. This idea is borrowed from the field of support vector learning. Additionally, we suggest further correction approaches and condition repair methods that allow transplanting these approaches to Kriging models.

Finally, in Chapter 9, we explore whether our methods are able to handle problems that are more complex. A tree-based symbolic regression task and hierarchical search spaces are investigated. Importantly, this is extended by a discussion of algorithm benchmarking. We propose a test function generator based on a data-driven Kriging simulation. This generator produces diverse test functions, which are able to reflect the behavior of real-world problems.

This work is finalized in Chapter 10, by giving an overall summary, providing some advice for practitioners and outlining future research directions.

To supplement these considerations, the appendix collects further odds-and-ends, including tools for the statistical analysis of results (Appendix A), a description of specific kernels employed in the thesis (Appendix B), a description of the employed variation operators (Appendix C), and additional figures (Appendix D).

1.4 Publications

Significant parts of this work are based on material that has previously been published or is being prepared for publication during the writing of this thesis. Of significance are the following documents: [273, 271, 264, 26, 266, 268, 270, 272].

All these works have been extended, rewritten, and restructured in some form before their inclusion into this thesis. The rewriting and restructuring were mostly aimed to increase the clarity, to unify notation and terminology, and put the described contributions in the overall context of this thesis. In most cases, this includes a revision or extension of the experiments. In the following, the changes are briefly outlined.

If not otherwise specified, these publications are in large parts a contribution of the author of this thesis. This is not intended to underrate the contributions of the co-authors, who provided inspiration to some of the presented ideas, helped to improve the writing quality, and gave support for the experimental procedures and analysis of these studies. In the following, we discuss each publication. For the sake of clarity, these remarks are repeated at the start of the respective sections or chapters.

Chapter 3 is partly based on the article *"Model-based Methods for Continuous and Discrete Global Optimization"* by Bartz-Beielstein and Zaefferer [26]. This includes many text elements that are taken verbatim from that publication. In the article, the discussion of continuous optimization methods was mostly contributed by Thomas Bartz-Beielstein. The part that is relevant to this chapter was mostly contributed by the author of this thesis. The survey was rewritten, restructured, and expanded. To a lesser extent, a few considerations have been taken from the article *"Efficient Global Optimization for Combinatorial Problems"* by Zaefferer et al. [273].

In less detail, some of the terms and definitions from Chapter 4 have already been discussed in Section 2 of *"An Empirical Approach for Probing the Definiteness of Kernels"* by Zaefferer et al. [266]. Hence, there is some overlap with that section. We added more details, including an illustrative motivation of the kernel trick.

Chapter 5 is partially based on the article *"Efficient Global Optimization for Combinatorial Problems"* by Zaefferer et al. [273]. Occasionally, text elements have been adopted verbatim from that publication. Overall, the text was significantly rewritten and extended before its inclusion into this thesis. We also added an illustrative example. The described experiments were repeated in a more thorough way. This included a more flexible, self-adaptive evolutionary algorithm, a broader set of test functions, and a parameter sensitivity study.

Chapter 6 is based on the article *"Distance Measures for Permutations in Combinatorial Efficient Global Optimization"* by Zaefferer et al. [271], with similar extensions as in Chapter 5. Some text elements have been adopted verbatim from the original contribution. The content of the publication been extended significantly. Firstly, several additional methods for dealing with multiple kernels are discussed, including a more comprehensive explanation of the original ideas. Secondly, a more thorough experimental study now includes these methods as well as additional distance measures and test functions. Furthermore, the results are subject to a more in-depth analysis.

Chapter 7 is based on the article *"An Empirical Approach for Probing the Definiteness of Kernels"* by Zaefferer et al. [266]. The material was revised to embed it into the context, notation, and structure of this thesis. Otherwise, a majority of the text has been adopted verbatim from the original document. The experiments and analysis were not changed.

Chapter 8 is based on the article *"Efficient Global Optimization with Indefinite Kernels"* by Zaefferer et al. [264]. It has been extended and rewritten. Some parts are

taken verbatim from the original publication. An illustration with a one-dimensional example was added. Further, the experimental setup was extended to deal with test problems that are more varied and of a higher dimension. Correspondingly, the experimental analysis had to be adapted. The remarks on non-stationarity are also not part of the original article. Finally, some additional repair methods based on a linear combination and a nearest-neighbor approach were added.
Chapter 9 is largely based on three publications [272, 270, 268].
Section 9.1 is based on the article *"Linear Combination of Distance Measures for Surrogate Models in Genetic Programming"* by Zaefferer et al. [272]. Especially in the problem description, the description of the distances, and the experimental setup, major parts were taken verbatim from the original article. Otherwise, the text was revised and the description of the distances and the analysis were supplemented with a discussion of definiteness. This includes additional experimental results with definiteness correction methods. The analysis was extended, especially with visualizations.
Section 9.2 is based on the article *"A First Analysis of Kernels for Kriging-based Optimization in Hierarchical Search Spaces"* by Zaefferer and Horn [270]. It was prepared in a coequal cooperation with Daniel Horn, who especially contributed to the statistical and visual analysis of the experimental investigation. Since this article was written in equal parts by both of its authors, it is not discussed entirely. We discuss the kernels contributed by the author of this thesis in more detail (with few verbatim adoptions) and briefly summarize the experimental results. An additional kernel (Wedge-kernel) is proposed.
Section 9.3 is mostly based on *"Simulation Based Test Functions for Optimization Algorithms"* by Zaefferer et al. [268]. The text is in parts taken verbatim from that publication. However, it was largely rewritten and extended to 1) give a clearer motivation supported by visualizations, 2) add some additional remarks on related approaches, 3) discuss the advantages and disadvantages in a more structured way, 4) provide more illustrative examples to explain how the test function generator works, and 5) show results from an additional experiment with a random forest model.
In addition to these main chapters, a few excerpts regarding the description of kernels, distances, and variation operators described in the appendix are taken from already mentioned publications [273, 271, 264, 266].
Finally, the author of this thesis was involved in several research projects and studies that are not directly related to the core issues that are discussed here. Hence, they are not included in these deliberations. In particular, some of these publications discussed interesting applications of SMBO algorithms in fields like process optimization [269], algorithm tuning [265, 267], and mechanical engineering [143, 27]. While none of these applications considered discrete or combinatorial problems, they provided an initial motivation and foundation for this work.

Part I

Background and State of the Art

Chapter 2

Continuous Methods for Expensive Black-box Optimization

The modeling and optimization methods developed in this thesis are based on established techniques in the field of vector-valued, continuous optimization. The foundations of methods for expensive, black-box optimization problems are introduced in the following. In this chapter, a candidate solution x is considered to be an m-dimensional real-valued vector in the search space $\mathcal{X} \subseteq \mathbb{R}^m$, and $f(x)$ is a continuous, expensive, black-box function.

2.1 Evolutionary Algorithms

One prominent class of solvers for black-box optimization problems are Evolutionary Algorithms (EAs) [226, 208, 13, 82, 23]. These algorithms transfer the concept of natural evolution to numerical optimization. EAs combine selection, mutation, and recombination of candidate solutions iteratively, as outlined in Algorithm 1 and Fig. 2.1.

The following list summarizes some of the most important aspects of an EA.

- Candidate solutions $x \in \mathcal{X}$ are called *individuals*.
- A *population* $X \subset \mathcal{X}$ is a set of individuals.
- The objective function $f(x)$ is called the *fitness function*. The output of that function is the *fitness* of an individual x.
- A *recombination operator* is a function that creates one or more *offspring* (new individuals, denoted X' in Algorithm 1), using information of two or more *parents* (old individuals). For instance, two parents $x^{(1)}$, $x^{(2)}$ can be recombined by taking the average in each dimension: $x' = (x^{(1)} + x^{(2)})/2$. This particular example is called intermediate crossover.

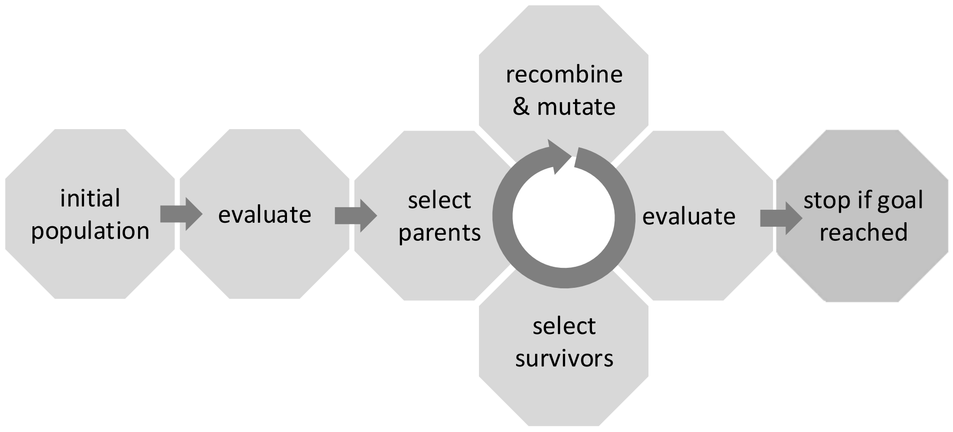


Figure 2.1: *A simplified schema of how an evolutionary algorithm evolves better solutions.*

- Mutation operators induce slight changes in offspring. For example, a normally distributed error is added to an individual $x' = x' + \mathbf{z}$, where $\mathbf{z}$ is a vector of independent samples from the normal distribution, i.e., $z_i \sim N(\mu, \sigma^2)$, with $\mu = 0$. Here, $N(\mu, \sigma^2)$ denotes the normal distribution with mean μ, and variance σ^2. In this context, σ^2 is also called the *step-size*, step-length, or mutation strength.

- A selection mechanism is required to select which of the individuals in a population are chosen as parents (parent selection). Furthermore, a subset of the parent population and the offspring needs to be selected for the next iteration (survivor selection). The selection process usually depends on the fitness (or fitness ranks).

One important aspect of EAs is the configuration of their parameters, e.g.,

- the population size n_{pop} (the size of the set X),
- the number of generated offspring n_{off} (the size of the set X'),
- the choice of mutation operator (Line 11 of Algorithm 1),
- the choice of recombination operator (Line 10 of Algorithm 1),
- the choice of selection operators (Lines 9 and 19 of Algorithm 1), and
- parameters of the operators (e.g., step-sizes or selection probabilities).

All these parameters can influence the performance of an EA. A good configuration may be critical to the success of the algorithm. This implies a meta-optimization problem: the optimization of the algorithm's performance. Optimal values depend not only on the algorithm but also on the problem. For example, highly multimodal problems may require a larger population size, whereas a small population size is more

Algorithm 1 Evolutionary algorithm.

```
1: function EA($f(x)$, init(), select() mutate(), recombine(), terminate())
2:     $X$ =init();                              ▷ create initial population with $X \subset \mathcal{X}$
3:     $n = |X|$;                                ▷ size of the population
4:     for $j = 1$ to $n$ do
5:         $y_j = f(x^{(j)})$;                   ▷ evaluate fitness of each individual
6:     end for
7:     $\mathbf{y} = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^\mathrm{T}$;
8:     while not terminate() do
9:         $X_\text{parent}$ = select $(X, \mathbf{y})$;      ▷ parent selection
10:        $X'$ = recombine $(X_\text{parent})$;             ▷ recombination
11:        $X'$ = mutate $(X')$;                             ▷ mutation
12:        $n = |X'|$;
13:        for $j = 1$ to $n$ do
14:            $y'_j = f(x'_j)$;                 ▷ evaluate fitness of offspring
15:        end for
16:        $\mathbf{y}' = \begin{bmatrix} y'_1 & \dots & y'_n \end{bmatrix}^\mathrm{T}$;
17:        $X = X \cup X'$;                      ▷ add offspring to population
18:        $\mathbf{y} = \begin{bmatrix} \mathbf{y} \\ \mathbf{y}' \end{bmatrix}$;
19:        $X, \mathbf{y}$ = select $(X, \mathbf{y})$;        ▷ survival selection
20:    end while
21: end function
```

efficient for a unimodal problem. Therefore, to understand and improve an EA's performance, its parameters have to be considered. In this context, we will call them meta-parameters. Two approaches are important in that respect.

Firstly, parameter tuning can solve this meta-optimization problem via some optimization algorithm. This algorithm can itself be an EA, or some more sophisticated method, such as sequential parameter optimization [24, 20]. Tuning not only improves results, but also enables a fair comparison between the tuned algorithms.

The second approach is parameter control [227, 81]. This includes a process into the algorithm that adapts the algorithm parameters during its runtime. One classical example is the so-called one-fifth rule suggested by Rechenberg [208]. This rule was devised for the step-size adaptation of the (1+1)-Evolution Strategy (ES) on two unimodal fitness functions. The (1+1)-ES operates with a single parent and a single offspring. The rule states that the step-size should be increased if more than one trial in five iterations yield a better fitness, and it should be decreased if fewer improvements are observed. Another parameter control approach is self-adaptation. The term self-adaptation implies that the algorithm parameters are attached to the parameters x of the objective function [81]. That means, $x^* = \begin{bmatrix} x & x_\text{meta} \end{bmatrix}$, where x_meta are meta-parameters of the EA (or

more specifically, all parameters controlled by self-adaptation), x are the parameters of the underlying fitness function, and x^* is their concatenation. Consequently, some of the EA's parameters may vary between different individuals. By processing x^* instead of just x, the EA can search for the optimum of the fitness function and the solution of the meta-optimization problem simultaneously.
Parameter control is not without problems and may still require tuning. Firstly, it has been shown that self-adaptation may lead to premature convergence [217]. Secondly, most adaption schemes introduce new parameters, such as learning rates. We refer to these parameters as learning parameters. Still, it is usually argued (but not necessarily guaranteed) that algorithms should be less sensitive to learning parameters [81].
For example, Algorithm 2 represents a self-adaptation scheme that is used in this thesis. This self-adaptation process is applied to the parent individuals, before an offspring is created by recombination and mutation. Therefore, the process is placed between Lines 9 and 10 of Algorithm 1. For the adaptation of real-valued meta-parameters, we use an approach similar to that in the mixed-integer ES (MIES) [84, 163]. In addition to the MIES, we also consider categorical meta-parameters, e.g., the choice of variation operators. To that end, we use a rather simple concept: If offspring are generated by recombination, the categorical meta-parameter is chosen randomly from each parent. This is also called dominant crossover. Afterwards, mutation may change a categorical meta-parameter with probability p_s. If a mutation is triggered, the adaptation is performed via a uniform random sample from all values (except for the current value).

Algorithm 2 Self-adaptation of an EA's meta-parameters, placed between Lines 9 and 10 of Algorithm 1.

```
1: function SELF-ADAPT(parent population $X^*$ and learning parameters $\tau$, $p_s$)
2:     for all $x^* \in X^*$ do
3:         for all real-valued $x_{\text{meta}} \in x^*$ do                    ▷ e.g., mutation rate
4:             recombine: intermediate crossover;
5:             mutate: $x_{\text{meta}} = x_{\text{meta}} \exp(\tau z)$ with $z \sim N(0, 1)$;
6:         end for
7:         for all categorical $x_{\text{meta}} \in x^*$ do                ▷ e.g., variation operators
8:             recombine: dominant crossover;         ▷ choose randomly from parents
9:             mutate: uniform random sample, with probability $p_s$;
10:        end for
11:    end for
12:    return $X^*$
13: end function
```

In Algorithm 2, the learning rate τ and the probability p_s are new learning parameters introduced by the self-adaptation scheme. Here, these parameters are defined as scalars, but they can as well be specified as vectors to encode different rates or probabilities for each of the controlled meta-parameters.

Overall, EAs have several advantages that make them a promising choice for the black-box problems considered in this thesis:

- EAs require no additional information about the search space, such as gradients.
- Due to their population-based, stochastic nature, EAs have the potential to escape local optima.
- EAs can be easily adapted to different data structures, and are in fact widely used in continuous as well as discrete optimization.
- Although a well-performing EA may require significant parameter tuning effort, little domain knowledge is required for the general setup of an EA.

However, EAs have a crucial feature that puts them at odds with one main constraint of the problems considered in this thesis. EAs require many fitness evaluations, which are often expensive to evaluate in case of real world optimization problems. Hence, we need tools for the efficient optimization of expensive, discrete problems with EAs. In the remainder of this thesis, we focus on surrogate model-based optimization techniques that provide sophisticated tools for solving these problems.

2.2 Surrogate Model-based Optimization

Algorithm 3 defines a typical SMBO algorithm. A simplified schema of this algorithm is depicted in Fig. 2.2. Similar algorithms are used throughout this thesis. In the SMBO algorithm, $f(x)$ is the expensive objective function, `init()` generates an experimental design (a set of initial candidate solutions), `model()` is a function that trains data-driven surrogate models, `optimizer()` is an optimization algorithm that optimizes the model based on some `infill` criterion, and `terminate()` checks whether the algorithm should stop (e.g., based on a budget of evaluations of f).

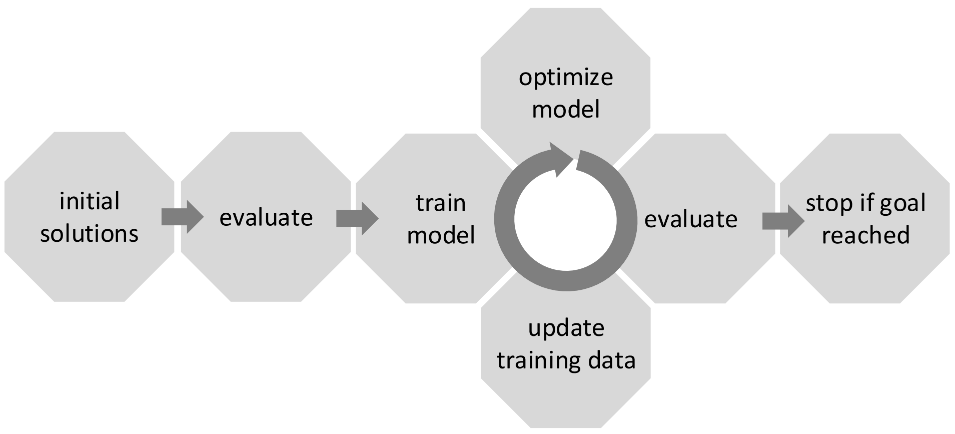


Figure 2.2: *A simplified schema of how an SMBO algorithm searches for better solutions.*

Algorithm 3 A typical SMBO algorithm

```
1: function SMBO($f(x)$, init(), model(), optimizer(), infill, terminate())
2:     $X$ =init();                                   ▷ create initial design with $X \subset \mathcal{X}$
3:     $n = |X|$;
4:     for $j = 1$ to $n$ do
5:         $y_j = f(x^{(j)})$;                         ▷ evaluate design
6:     end for
7:     $\mathbf{y} = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^{\mathrm{T}}$;
8:     while not terminate() do
9:         $M$ = model ($X, \mathbf{y}$);               ▷ create / update the model
10:        $x'$ = optimizer($M$,infill);           ▷ determine promising candidate
11:        $y' = f(x')$;                              ▷ evaluate candidate
12:        $X = X \cup \{x'\}$;                       ▷ add candidate to data set
13:        $\mathbf{y} = \begin{bmatrix} \mathbf{y} \\ y' \end{bmatrix}$;     ▷ add objective function value of candidate
14:    end while
15: end function
```

The `init()` algorithm may create, e.g., Latin hypercube designs [172] or other (preferably space-filling) designs.

The `model()` algorithm may employ, e.g., linear regression models, Random Forests (RF), Radial Basis Function Networks (RBFN), Artificial Neural Networks (ANN), Support Vector Machines (SVM), or other data-driven regression models [126, 137]. These models are assumed to be cheaper to evaluate than the expensive objective function $f(x)$. One of the more frequently used surrogate models is Kriging [220, 93]. It is in the focus of this thesis and is introduced in more detail in Section 2.3.

The `optimizer()` algorithm may be, e.g., the EA described in Algorithm 1, or any other global optimization algorithm. The optimizer can either directly optimize the prediction of the model, or else it could optimize a so-called `infill` criterion that is derived from the model [93] (cf. Section 2.4). In essence, the optimizer can thus avoid evaluating the expensive fitness function directly. Instead, the EA repeatedly optimizes a surrogate model that becomes increasingly accurate during the optimization run.

For a more detailed review of SMBO techniques, we refer to the literature [136, 257, 232, 137, 26]. One important link between EAs and SMBO is the pre-selection approach of the metamodel-assisted evolution strategies discussed by Emmerich et al. [83]. SMBO algorithms find application in many domains. Besides engineering design [93], algorithm tuning is a frequent area of application. The sequential parameter optimization approach [24, 20] mentioned in Section 2.1 is one important framework in this context.

Sometimes, SMBO problems can become multi-objective. That implies, potentially conflicting goals may have to be satisfied by the optimization procedure. We focus on single-objective problems, but the models discussed in this thesis may as well be applied to multi-objective problems. For more details on such considerations, we refer to a recent survey by Chugh et al. [63].

2.3 Introduction to Kriging

The popularity of Kriging (also known as Gaussian process regression) in the field of SMBO is based on its flexibility and predictive accuracy. In addition, it provides an estimate of its own uncertainty. The latter feature is especially useful in SMBO algorithms, as it is crucial for the computation of many infill criteria. The foundations of these ideas are the seminal works by Mockus et al. [175], Sacks et al. [220], and Jones et al. [140]. A very comprehensive description of Kriging-based optimization is given by Forrester et al. [93], whose work is also an important basis for the following remarks and equations.

2.3.1 Kriging Model

We assume that we have some data set $X = \{x^{(1)}, \ldots, x^{(n)}\}$ of $n \in \mathbb{N}$ samples, each sample being an m-dimensional real-valued $x \in \mathbb{R}^m$. The corresponding observations for each sample are represented by the vector $\mathbf{y} \in \mathbb{R}^n$. Kriging can make powerful predictions with the (on the first glimpse) simple model:

$$y_i = \mu + \epsilon_i$$

Importantly, Kriging assumes that the observations $\mathbf{y}$ are realizations of a stochastic process where the errors ϵ_i are spatially correlated. Two nearby observations should have a similar error. Thus, ϵ_i is clearly dependent on the sample location x, and we can write $\epsilon(x^{(i)})$. Errors are considered to be positively correlated if the distance between two sample locations is small. Correlation goes to zero with increasing distance. This relationship between the correlation of the errors and the distance of the samples can be modeled with, e.g.,

$$\mathrm{k}(x, x') = \exp\left(- \sum_{i=1}^{m} \theta_i |x_i - x'_i|^{p_i} \right), \tag{2.1}$$

where $x_i \in \mathbb{R}$ is the i-the component of the vector x. The function given in Eq. (2.1) is one important example of a correlation function, or kernel. Other correlation functions can be defined, but they would usually have to satisfy some important criteria. For instance, they should approach zero for large distances, they should yield one if $x = x'$, and they should usually be positive semi-definite (definiteness is discussed in Chapters 4, 7 and 8). A matrix that collects pairwise correlations between errors of all observations $\mathbf{y}$ at locations X is the correlation matrix (or kernel matrix) $\mathbf{K}$.

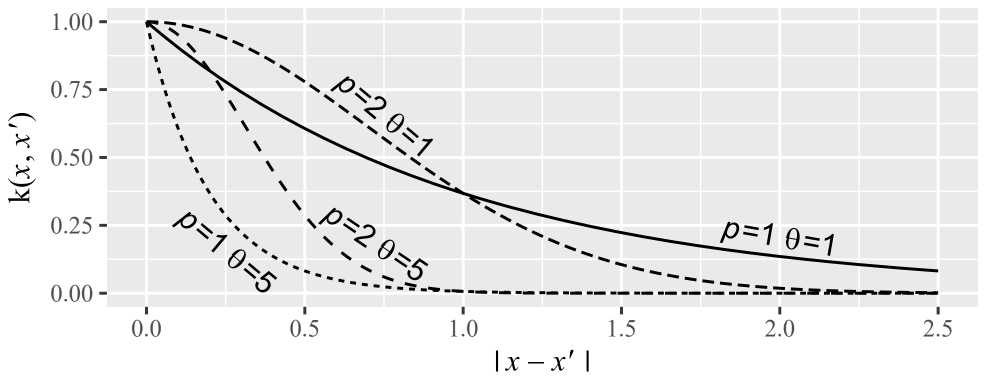


Figure 2.3: *An illustrative example of how the correlation function in Eq.* (2.1) *behaves with different sets of parameters. The plot shows the correlation in relation to the distance (absolute difference) of two real-valued, scalar samples.*

The correlation function in Eq. (2.1) has two parameters for each dimension of the search space, θ_i and p_i. The interpretation of the parameters θ_i and p_i can be best understood when looking at the example in Fig. 2.3. The parameter θ_i influences how fast the function decays to zero. The parameter p_i controls the smoothness, i.e., how acute the curve is when the distance approaches zero.

The values of these parameters can be determined by Maximum Likelihood Estimation (MLE) [93]. With MLE, parameters are chosen so that the observed data has the largest likelihood, given the specified model. The likelihood function of the Kriging model is based on the probability density function of a multivariate normal distribution,

$$\mathrm{pdf}_{\mathrm{MVN}}(\mathbf{y}) = \frac{1}{(2\pi)^{n/2}|\mathbf{C}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{1}\mu)^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{y}-\mathbf{1}\mu)\right),$$

where $\mathbf{1}$ is a vector of ones. The stationary covariance matrix $\mathbf{C}$ is related to the correlation matrix $\mathbf{K}$ with

$$\mathbf{C} = \sigma^2\mathbf{K}. \tag{2.2}$$

This yields the likelihood function

$$L(\epsilon(X)|\mu,\sigma,\theta,p) = \frac{1}{(2\pi\sigma^2)^{n/2}|\mathbf{K}|^{1/2}} \exp\left(-\frac{(\mathbf{y}-\mathbf{1}\mu)^{\mathrm{T}}\mathbf{K}^{-1}(\mathbf{y}-\mathbf{1}\mu)}{2\sigma^2}\right). \tag{2.3}$$

Here, μ and σ are also parameters to be determined by MLE. To determine all parameters (θ, p, σ, μ) by MLE, the partial derivatives of the likelihood function have to be set to zero. The partial derivatives yield closed-form solutions for μ and σ^2,

$$\hat{\mu} = \frac{\mathbf{1}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{y}}{\mathbf{1}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{1}},$$

and

$$\hat{\sigma}^2 = \frac{(\mathbf{y} - \mathbf{1}\hat{\mu})^{\mathrm{T}}\mathbf{K}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu})}{n}.$$

By taking the logarithm, substitution of $\hat{\mu}$ and $\hat{\sigma}$, and subsequent removal of constant terms, Eq. (2.3) becomes the so-called concentrated log-likelihood function:

$$\mathrm{con}(\ln(L)) = -\frac{n}{2}\ln(\hat{\sigma}^2) - \frac{1}{2}\ln(|\mathbf{K}|).$$

This function is used to find the remaining parameters of the correlation function. However, an algebraic, closed-form solution is not available. Instead, numerical optimization is required to maximize the concentrated log-likelihood with respect to the remaining parameters. This can be done with classical optimization methods, or with population-based methods, like EAs, to allow for a global search.

2.3.2 Kriging Prediction

With the equations discussed above, we can derive the necessary parameters of a model that describes the observed training data. But how can this model be used to generate predictions with respect to a new sample x^*? In the following, the basic idea of prediction with Kriging is explained. Forrester et al. give a more detailed derivation of the formulas [93] that we use as a basis for this description.
The unknown function value of a new solution is denoted as $\hat{y}$. Determining this value is denoted as prediction. The core idea is to add the new solution to the set of existing solutions. This yields the augmented vector of observations $\mathbf{y}_{\text{aug}} = \begin{bmatrix}\mathbf{y}^{\mathrm{T}} & \hat{y}\end{bmatrix}^{\mathrm{T}}$. Then, we treat $\hat{y}$ like a model parameter. That is, we maximize the likelihood function with respect to $\hat{y}$. To that end, the vector of correlations between a new sample x^* and the set of training samples X is, $\mathbf{k} = \begin{bmatrix}(\mathrm{k}(x^{(1)}, x^*), ..., \mathrm{k}(x^{(n)}, x^*)\end{bmatrix}^{\mathrm{T}}$. This is used in the augmented correlation matrix $\mathbf{K}_{\text{aug}} = \begin{bmatrix}\mathbf{K} & \mathbf{k} \\ \mathbf{k}^{\mathrm{T}} & 1\end{bmatrix}$. $\mathbf{K}_{\text{aug}}$, $\mathbf{y}_{\text{aug}}$, and all earlier determined model parameters are substituted into the likelihood function in Eq. (2.3). Maximizing the terms of Eq. (2.3) depending on $\hat{y}$ yields the predictor

$$\hat{y}(x^*) = \hat{\mu} + \mathbf{k}^{\mathrm{T}}\mathbf{K}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu}). \tag{2.4}$$

2.3.3 Uncertainty Estimate

An important feature of Kriging is that it also provides an estimate of its own local uncertainty, or its prediction error. Roughly speaking, if the new sample x^* has a large distance to the training data, the uncertainty of the predicted value is also large. Typically, the uncertainty also increases if the model gets more rugged or active in the respective region (i.e., large rates of change). The uncertainty of the prediction can be estimated with [220, 93]

$$\hat{s}^2(x^*) = \hat{\sigma}^2(1 - \mathbf{k}^\mathrm{T}\mathbf{K}^{-1}\mathbf{k}). \tag{2.5}$$

Kriging is not the only modeling tool that can produce uncertainty estimates. For example, uncertainties could always be estimated by training models with subsets of the data (e.g., cross-validation). However, stochastic models, like Kriging, produce these estimates quite naturally, with little additional effort, and impose important properties on the uncertainty estimates. For instance, if no noise is modeled, the uncertainty estimate goes to zero with decreasing distance to a training data sample. Uncertainties estimated by cross-validation would be non-zero at the training samples. The importance of the uncertainty estimate and its properties in model-based optimization becomes more clear in Section 2.4.

Example 2.3.1. To demonstrate the predictor and uncertainty estimate of a Kriging model, we consider the simple function

$$y = f(x) = x^4 - 2x^2 + x,$$

with $x \in \mathbb{R}$. The function has been sampled at $X = \{-1.3, -0.6, 0.1, 0.8, 1.5\}$. The corresponding observations are $\mathbf{y} \approx \begin{bmatrix}-1.82 & -1.19 & 0.08 & -0.07 & 2.06\end{bmatrix}^\mathrm{T}$. We fix $p = 2$ for the sake of simplicity and the maximum likelihood estimate of the remaining kernel parameter is $\theta \approx 1.97$. We provide a plot of the resulting predictor in Fig. 2.4.

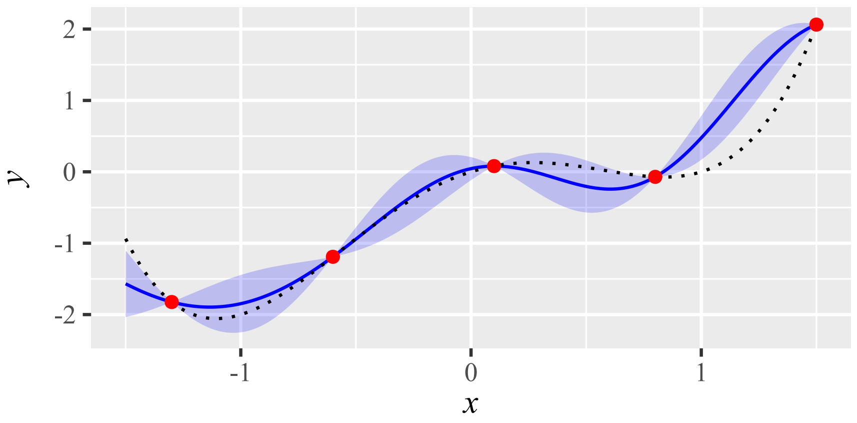


Figure 2.4: *The prediction $\hat{y}(x)$ of a Kriging model (blue solid line) based on five data samples (red dots) from the true function $f(x) = x^4 - 2x^2 + x$ (black dotted line). The shaded region indicates the uncertainty estimate of the model, i.e., $\hat{y}(x) \pm \hat{s}(x)$.*

2.3.4 Nugget Effect

Kriging also allows dealing with noisy data, using the so-called nugget effect. This adds a small constant (the nugget, or the regularization constant η) to the diagonal of

the correlation matrix. If the nugget is employed, the otherwise interpolating Kriging model is able to regress the data, introducing additional smoothness into the predicted variable.
This thesis mostly deals with deterministic problems. Still, the nugget can be useful in this context. It is often used for regularization, that is, to increase numerical stability: If the correlation matrix is close to singular, increasing the diagonal may be necessary to guarantee that the inverse of the matrix can still be reliably computed. Mohammadi et al. [176] cover regularization of ill-conditioned matrices for Kriging in more detail.
One drawback of the nugget effect is that it may deteriorate the uncertainty estimate. That is, the uncertainty estimate becomes non-zero at observed locations. To avoid such an issue, a so-called re-interpolation approach can be employed [93]. This approach re-computes the uncertainty estimates based on the predicted values instead of the true observations.

Example 2.3.2. As an extension of Example 2.3.1, Fig. 2.5 shows a model with a nugget of $\eta = 0.1$ but without re-interpolation. Figure 2.6 shows the same model with re-interpolation. In both cases, the nugget leads to a model that does not exactly reproduce all training samples.

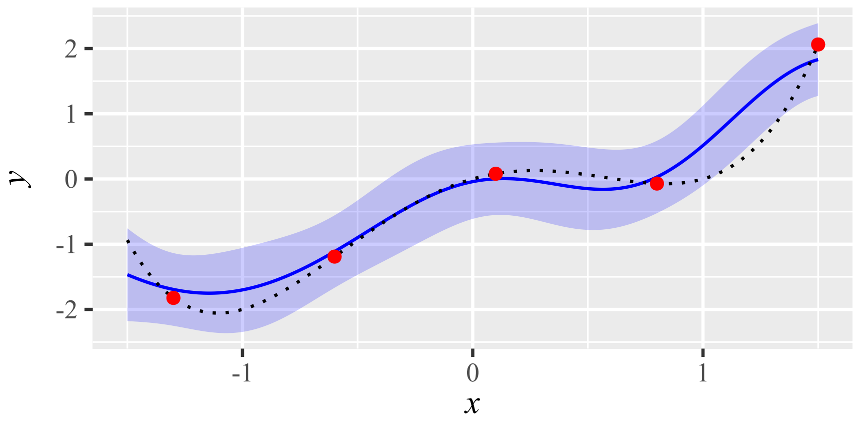


Figure 2.5: *The prediction $\hat{y}(x)$ of a Kriging model with nugget effect but without re-interpolation (blue solid line) based on five data samples (red dots) from the true function $f(x) = x^4 - 2x^2 + x$ (black dotted line). The shaded region indicates the uncertainty estimate of the model, i.e., $\hat{y}(x) \pm \hat{s}(x)$.*

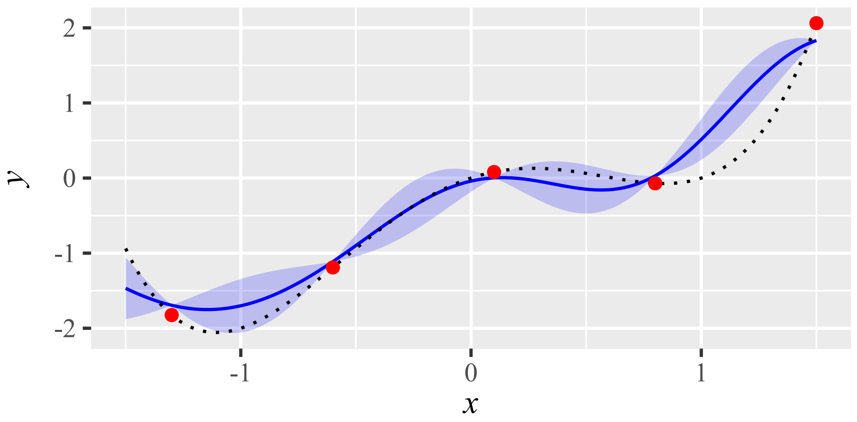


Figure 2.6: *The prediction $\hat{y}(x)$ of a Kriging model with nugget effect and re-interpolation (blue solid line) based on five data samples (red dots) from the true function $f(x) = x^4 - 2x^2 + x$ (black dotted line). The shaded region indicates the uncertainty estimate of the model, i.e., $\hat{y}(x) \pm \hat{s}(x)$.*

2.4 Efficient Global Optimization

The Efficient Global Optimization (EGO) algorithm's [140] most distinctive feature is its use of the uncertainty estimate $\hat{s}^2(x)$. EGO essentially follows the typical SMBO concept presented in Algorithm 3. The most important distinction occurs in Line 10. Here, EGO does not simply minimize the prediction, but rather optimizes a so-called infill criterion. In EGO, this infill criterion is the Expected Improvement (EI) [175, 140]. Other infill criteria include, e.g., the Probability of Improvement (PI) or the Lower Confidence Bound (LCB) [93].

For each sample x, the Kriging model specifies a normal distribution with mean $\hat{y}(x)$ and variance $\hat{s}^2(x)$. Based on that distribution, we can estimate a probability of reaching a certain objective function value or better. For the model from Example 2.3.1, Fig. 2.7 shows the corresponding probabilities. The PI of each sample x is the depicted probability along the dashed line that marks the best-observed function value.

Unfortunately, the PI does not consider how large a potential improvement is. Here, the improvement implies the difference between a potential function value and the best-observed value. Clearly, improvements cannot be negative. Hence, the improvement is defined as

$$I(\hat{y}) = \max\Big(\big(\min(\mathbf{y}) - \hat{y}\big), 0\Big).$$

If the PI is large, but the improvement itself is extremely small, the corresponding sample may not be a very promising candidate solution. The EI intends to alleviate this. It is the expectation of $I(x)$. In contrast to the PI, it takes the likelihood as well as the magnitude of the improvement into account.

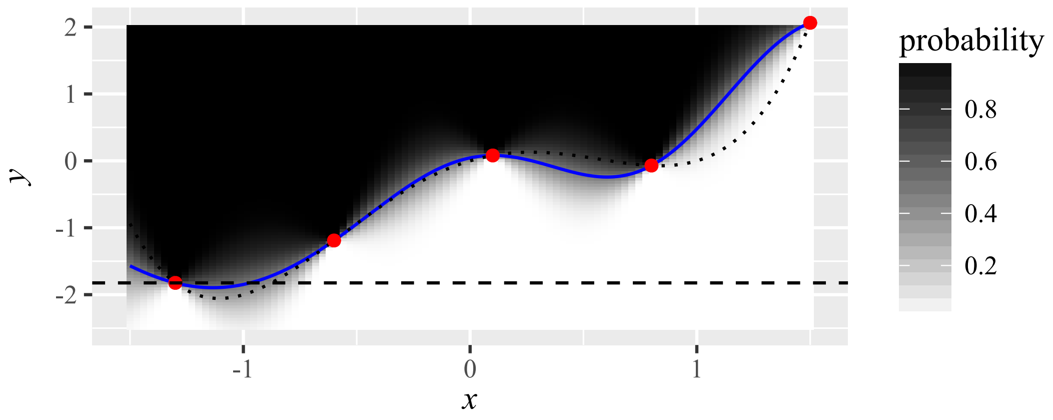


Figure 2.7: *The prediction of a Kriging model without nugget (blue solid line) based on five data samples (red dots) from the true function $f(x) = x^4 - 2x^2 + x$ (black dotted line). The color scale of the background presents the probability of attaining the corresponding y value or better. The dashed black line marks the best-observed function value. Improvements only occur below that line.*

The EI of a sample x is [140]

$$\text{EI}(x) = \begin{cases} y^*\Phi\left(\frac{y^*}{\hat{s}(x)}\right) + \hat{s}(x)\phi\left(\frac{y^*}{\hat{s}(x)}\right) & \text{if } \hat{s}(x) > 0 \\ 0 & \text{else} \end{cases}, \tag{2.6}$$

The EI is based on $y^* = \min(\mathbf{y}) - \hat{y}(x)$, the cumulative distribution function $\Phi(.)$ of the normal distribution, and the probability density function $\phi(.)$ of the normal distribution. Furthermore, $\min(\mathbf{y})$ returns the minimum of all observed function values $\mathbf{y}$. In each iteration of the EGO algorithm, the EI(x) is maximized. The resulting candidate solution x is evaluated with the expensive objective function $f(x)$.
EI is very useful to balance the trade-off between exploitation and exploration in SMBO. By respecting both the prediction $\hat{y}(x)$ *and* the uncertainty estimate $\hat{s}(x)$, EI drives the search into poorly explored regions with potentially good function values. It may prevent cases where a Kriging-based optimization algorithm would get stuck in local optima. Once a local optimum has been found, the EI of nearby solutions is close to zero (and exactly zero at the local optimum itself). Here, we understand a local optimum as a candidate solution with neighbors of equal or worse fitness.
Note that algorithms like EGO are also known as Bayesian optimization, following the terminology of Mockus et al. [175].

Chapter 3

A Survey of Surrogate Models for Discrete Problems

Chapter 2 described basic SMBO approaches established for real, vector-valued search spaces, with $x \in \mathcal{X} \subseteq \mathbb{R}^m$. This chapter comes back to the focus of this thesis: the broad class of search spaces, $x \in \mathcal{X}$ where $\mathcal{X}$ is some nonempty set, e.g., representing discrete, combinatorial, or mixed search spaces.
As already mentioned in Section 1.1, SMBO approaches were rarely discussed in the context of combinatorial, discrete, or mixed optimization. This scarcity is unlikely due to a lack of problems in this field. Rather, suitable surrogate modeling methods are not available or known to experts in potential application domains. To close this gap and to provide a background for our contributions, we provide a survey of discrete surrogate modeling techniques.

3.1 Strategies for Dealing with Discrete Structures

For surrogate modeling in discrete search spaces, six strategies (STR) can be identified in the literature.

STR-1 The naive approach: As long as the data can still be represented as a vector (binary variables, integers, categorical data, permutations) the discrete structure can be ignored. SMBO can be applied without any modifications.

All following sections of this chapter are based on parts of the article *"Model-based Methods for Continuous and Discrete Global Optimization"* by Bartz-Beielstein and Zaefferer [26]. This includes many text elements that are taken verbatim from that publication. In the article, the discussion of continuous optimization methods was mostly contributed by Thomas Bartz-Beielstein. The part that is relevant to this chapter was mostly contributed by the author of this thesis. The survey was rewritten, restructured, and expanded. To a lesser extent, a few considerations have been taken from the article *"Efficient Global Optimization for Combinatorial Problems"* by Zaefferer et al. [273].

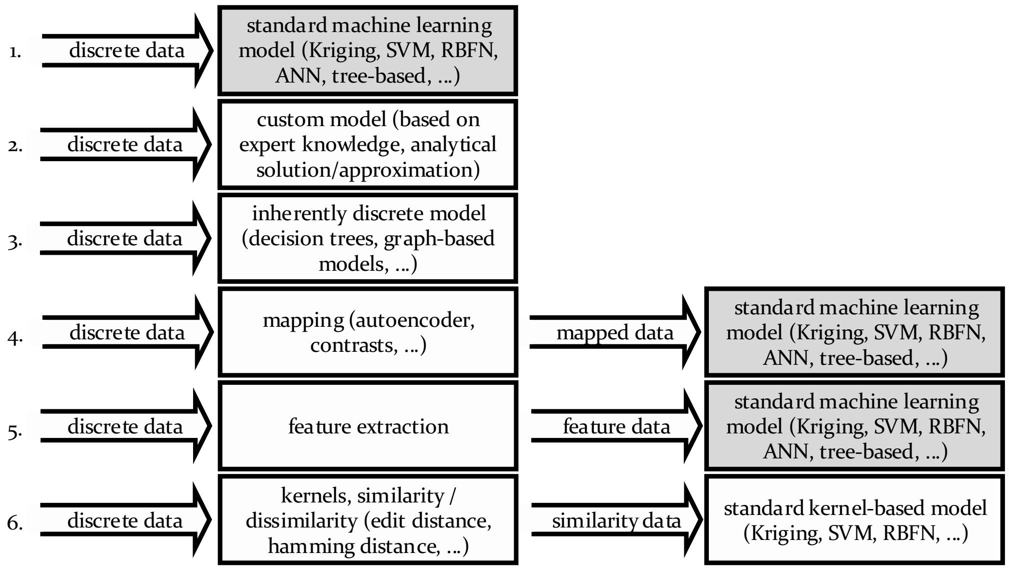

1.
discrete data
standard machine learning model (Kriging, SVM, RBFN, ANN, tree-based, ...)
2.
discrete data
custom model (based on expert knowledge, analytical solution/approximation)
3.
discrete data
inherently discrete model (decision trees, graph-based models, ...)
4.
discrete data
mapping (autoencoder, contrasts, ...)
mapped data
standard machine learning model (Kriging, SVM, RBFN, ANN, tree-based, ...)
5.
discrete data
feature extraction
feature data
standard machine learning model (Kriging, SVM, RBFN, ANN, tree-based, ...)
6.
discrete data
kernels, similarity / dissimilarity (edit distance, hamming distance, ...)
similarity data
standard kernel-based model (Kriging, SVM, RBFN, ...)

Figure 3.1: *Six different strategies for surrogate modeling in discrete search spaces.*

STR-2 Customized models: A specific modeling solution is tailored to fit the needs of a certain application.

STR-3 Inherently discrete models: Some models already are discrete by their own design.

STR-4 Mapping: Discrete variables or structures are mapped to a more easily treatable representation.

STR-5 Feature extraction: Instead of directly modeling the relation between an object (or its representation) and its quality, it is possible to calculate numeric features of the object. The resulting feature values can be modeled with standard techniques.

STR-6 Kernel-based modeling: Where available, discrete kernels (measures of (dis)-similarity) may be used to replace continuous kernels that, e.g., are employed in models like k-Nearest Neighbors (k-NN) [39], SVM [224], RBFN [39], or Kriging.

The subsequent sections present key characteristics of the strategies, and summarize the relevant literature.

3.2 Naive Approach

The most basic strategy to deal with a discrete search space $\mathcal{X}$ is to simply ignore the nature of the search space and apply the usual methods from continuous optimization.

Clearly, this is only feasible if the sample $x \in \mathcal{X}$ can still be represented as a vector. Especially if the discrete variables are of an ordinal nature, the naive approach may still be successful. A potential drawback of this approach is that the model's input space may have large areas of redundancy. Or else, this approach may create large areas of infeasible solutions. Depending on the employed optimization algorithm, this may deteriorate performance compared to strategies that are more advanced.

3.3 Customized Models

A more sophisticated approach is to employ customized, application specific solutions. On the one hand, this procedure can be very efficient because it integrates significant prior knowledge into the model. On the other hand, a customization approach may introduce bias. The result may be hard to transfer to other applications or data structures. If applicable at all, customized models often require significant effort or knowledge of the application domain expert. Clearly, this approach is not applicable to true black-box problems.

One example for customized models is presented by Voutchkov et al. [254]. They optimize a weld sequence, which is represented as a signed permutation. Here, the weld path is divided into segments. The permutation specifies the order in which those segments are welded. Each sign indicates the direction in which a segment is welded. They attempt to find a weld sequence that yields a minimal displacement due to thermal stresses. Simulating the displacement via a finite element model is computationally expensive. Hence, Voutchkov et al. suggest replacing it with a surrogate model by estimating the influence of each segment in the weld sequence, based on previously simulated weld sequences. In addition to the objective function values, this surrogate model also exploits intermediate results that can express the impact of individual segments, depending on their position in the weld sequence. Voutchkov et al. [254] find a near-optimal welding sequence by evaluating only 28 out of $46,080$ possible sequences. Variations of this approach for weld sequence optimization are described by Asadi and Goldak [11], and Bonnaud [42]

For binary problems, Verel et al. [252] suggest a model based on the Walsh decomposition. In terms of model quality, they show that the Walsh model tends to outperform Kriging on larger data sets. We categorize the Walsh model as a customized model because it cannot be immediately transferred to other problem classes, such as permutation problems. Still, similar decomposition approaches may be devised for other problem classes.

Another example is presented by Nguyen et al. [192], who use a simplified objective function (a simulation model for job shop scheduling) as a surrogate model in genetic programming. They actively attempt to reduce the complexity of the objective function (e.g., reduced simulation times, reduced number of machines or operations). This approach is also used by Mei et al. [174]. Similarly to the weld sequence optimization approach, a simplification of the simulation model is specific to the considered application. While similar simplifications may be possible with most simulation optimization

problems, it is infeasible to apply this approach to a problem that is not based on a simulation procedure.
Following these examples, many classical approaches in combinatorial optimization could also be labeled as customized surrogate models. Consider for instance the traveling salesperson problem. Here, to optimize the cost of visiting a set of locations, a permutation is optimized such that the traveled distance is minimal. Clearly, this model is only an approximation of reality. It does not account for many aspects of the real-world problem, e.g., stochasticity of actual travel costs or asymmetric travel costs between locations. Hence, the TSP model could be understood as a (customized) surrogate model. In this sense, customized surrogate models could be thought of as a standard approach towards solving combinatorial optimization problems.

3.4 Inherently Discrete Models

Some models are themselves discrete. One example is the class of tree-based models, such as regression trees or random forests. Random Forests have already been employed as surrogate models in the field of algorithm tuning [126, 128, 246, 80, 125, 38, 198, 72].
On the one hand, these models easily deal with discrete data. They are an intuitive way to incorporate binary, categorical, or ordinal integer variables.
On the other hand, this strategy may fail if the discrete structures become more complex (e.g., graphs). Furthermore, in a mixed-variable case, a tree-based model is not ideal to represent the continuous parameters in the mix. This issue may require the coupling of a discrete model and a continuous one, e.g., treed Gaussian processes [106, 107]. Finally, discrete models may not always provide features required by the optimization algorithm. For example, uncertainty estimates can be derived from random forests by estimating the variance from the prediction of all trees in the forest. Unfortunately, such an estimate does not have the same useful properties of the uncertainty estimates of a Kriging model (cf. Section 2.3.3).

3.5 Mapping

Another important approach to discrete surrogate modeling is to map from the complex, discrete space to another space that is easier to handle. Frequently employed approaches are dummy variables and contrasts. They are often applied in the context of classical linear regression models. A categorical variable can be mapped to a set of dummy variables, where each dummy represents a single level of the original variable. This is the same as the one-hot encoding, i.e., only one of several bits can be 1. The n_{cat} levels of the categorical variable are mapped to n_{cat} binary variables. The resulting binary variables are still discrete, but can be handled by standard regression approaches. Alternatively, a map to $(n_{\text{cat}} - 1)$ binaries can be used, where the missing level is represented by all binary variables being zero. This is an example of a contrast, which represents a comparison between different levels. Clearly, both approaches increase the number of model parameters, due to the increased dimensionality of the mapped data. Dummy variables or contrasts are probably among the most frequently applied methods to deal with categorical variables, e.g., in the context of the response surface methodology [188]. A similar approach is often used in genetic algorithms, e.g., to represent integers or real-valued variables with a binary encoding [82].

Another example is the genotype/phenotype mapping that is used to apply continuous evolution strategies to permutation problems, as proposed by Rudolph [216]. This has also been called the random key mapping [29]. The data is first represented by vectors of real numbers from a fixed interval, e.g., the interval $[0, 1]$. To map from this real parameter space to permutation representations, the values are simply sorted. For example, the real vector $x = \begin{bmatrix}0.1 & 0.3 & 0.2 & 0.9\end{bmatrix}$ is mapped to the permutation $\begin{bmatrix}1 & 3 & 2 & 4\end{bmatrix}$ because $x_1 < x_3 < x_2 < x_4$. This mapping approach has also been used in probabilistic models (cf. Section 3.8). Notable drawbacks are redundancies in the real-valued parameter space as well as the non-bijective character of the mapping. Redundancy occurs, e.g., when $\begin{bmatrix}0.1 & 0.3 & 0.2 & 0.9\end{bmatrix}$ and $\begin{bmatrix}0.2 & 0.4 & 0.3 & 0.8\end{bmatrix}$ both map to the same permutation. This induces areas in the real-valued space where observed function values are constant because they represent the same candidate. This may pose a difficulty to some models and optimization algorithms. Non-bijectivity implies that while each real-valued vector can be mapped to a single permutation, the converse is not true.

A more sophisticated example of the mapping strategy are autoencoders. Autoencoders are unsupervised models that try to reconstruct their own inputs with a coupling of an encoder and a decoder. This approach is often used for dimensionality reduction, in which case the coded data (after encoding, before decoding) is lower-dimensional than the original data. But this data may also be of a different type. That is, the original data may be discrete and the coded data continuous. One example is an application of autoencoders in automatic chemical design described by Goméz-Bombarelli et al. [104]. They use an autoencoder to map from the space of discrete molecule sequences to a latent continuous space, and vice versa. Blaschke et al. [40] compare different autoencoder models in this context.

One additional advantage of autoencoders is that mapping to a continuous space allows using gradient information during the search. For example, gradient information is exploited by Mueller et al. [185]. They search for optimal sequences in the context of text revision, making use of an ANN-based autoencoder and gradient ascent.

One potential problem of the autoencoder approach is the feasibility of candidate solutions. For instance, it may be infeasible to synthesize certain suggested molecules. Griffiths and Hernández-Lobato [108] suggest alleviating this issue by modeling the unknown constraint function. Based on such a model, they use an infill criterion that combines the expected improvement and the probability of constraint satisfaction of a candidate solution. Furthermore, autoencoders are fairly complex model structures that may be difficult to configure. Still, recent examples indicate that they are promising solutions for quite difficult search spaces [104, 185, 108, 40].

3.6 Feature Extraction

In many cases, it is possible to extract numeric features of the optimized object. Then, standard models can be trained with these feature values. For example, features of a tree or graph can be path lengths, tree depths, number of nodes, or the frequency of certain node labels.

Of the few approaches that deal with modeling for tree-representations (symbolic regression, genetic programming), many use feature extraction for modeling. Hildebrandt and Branke [121] extract features of the phenotypic behavior of evolved dispatching rules for job shop scheduling. Here, these features are based on rule decisions for a small set of reference situations. The Euclidean distance is computed on these features and is then employed in a k-NN surrogate model. From a different point of view, this process can as well be seen as a part of the (dis-)similarity calculation, belonging to STR-6 rather than STR-5. Hildebrandt and Branke compare the phenotypic approach to a genotypic distance measure: the structural Hamming distance. The phenotypic feature extraction approach has since also been investigated by Nguyen et al. [191] who improve the handling of replications and the selection scheme.

Pilat and Neruda [200] take a slightly different approach. They extract features from the genotype of candidate solutions, e.g., the depth of the tree, summarizing statistics of numeric constants in the tree, or the cardinality of certain arguments in the tree.

Feature extraction is also applied in the field of experimental chemical design. In this context, Balachandran et al. [17] suggest a two stage approach. First, they filter candidates via a classification approach. Then, they use a regression approach and expected improvement to select the most promising candidates. Both the regression model and the classification models are based on data that are derived as continuous features of the chemical compositions.

3.7 Kernel-based Models

Kernel-based models use suitable measures of (dis)similarity to represent structured, discrete data. Depending on their interpretation and use, these measures are referred to as similarity measures, dissimilarity measures, distance measures, correlation measures, or kernels. While this approach is potentially very powerful, it relies on the availability of a suitable measure. This may be especially problematic if a measure has to fulfill further requirements, e.g., being a metric or being positive semi-definite. A more detailed discussion of kernels and related aspects is given in Chapter 4.

Fonseca et al. [92] define similarity-based models as models that keep a memory of data samples and estimate the performance of new samples by comparing them to the memorized samples via a similarity measure. Fonseca et al. list fitness inheritance [236], fitness imitation [148, 136], and k-NN [8] as examples for similarity-based models. In their numerical experiments, they use a k-NN surrogate model in a genetic algorithm to solve a set of numerical, continuous test problems. Bernardino et al. [35] perform similar tests, but use artificial immune systems as solvers. In both cases, the

Hamming distance and the Euclidean distance are used as measures of dissimilarity. This shows that kernel-based models do not depend on a specific data representation or measure. However, the k-NN model is not able to predict whether a candidate solution provides an improvement over existing solutions. Usually, the best-observed value is an upper bound on the prediction of the k-NN model.

To predict potential improvements, more sophisticated kernel-based models are of interest: RBFN, SVM, and Kriging. Hemker [117] describes an approach for dealing with continuous and ordinal integer variables, which is applied to electrical engineering and water resource management problems. Their approach is based on Kriging, essentially treating all parameters with the same distance function. Hence, this approach is not directly applicable to categorical parameters or representations that are more complex. Li et al. [164] propose an adaptation of RBFNs. These adaptations are based on a weighted sum of distance measures, which replaces the usual distance measures that are employed in RBFNs. They apply the distance-based RBFN model to mixed-integer optimization problems. Brownlee and Wright [48] use the same approach for a multi-objective building design optimization problem.

Mixed optimization problems also arise in algorithm tuning. In this context, Hutter et al. [127] propose a Kriging model with a Hamming distance-based correlation function to handle categorical variables. A combination of Kriging and tree-based models called treed Gaussian processes is discussed by Swiler et al. [242]. They apply a treed Gaussian process model, a Kriging model, and a smoothing spline technique to build surrogate models for mixed-integer problems. Coelho et al. [87] and Herrera et al. [119] apply a kernel-based regression method to mixed-variable optimization problems. A different approach with RBFNs for the case of a mixed (discrete and continuous) optimization problem is taken by Bajer and Holena [16]. They use the Hamming distance to cluster the discrete variables, then fit a standard RBFN to the continuous variables for each cluster.

Moraglio and Kattan [179] propose to use arbitrary distances measures in an RBFN to produce surrogate models for combinatorial optimization problems. They applied their approach to quadratic assignment problems [181]. A corresponding extension for Kriging was investigated by Zaefferer et al. [273]. Here, Kriging-based EGO [140] showed positive results when applied to combinatorial problems. Zaefferer et al. [271] also showed that MLE could be used to efficiently select from a set of distance measures. More details with respect to these works can be found in Chapters 5 and 6.

Cáceres et al. [199] report negative results for the application of Kriging and EGO to permutation problems. They report that an ant colony optimization algorithm was not outperformed by a Kriging-based variant of the same algorithm. Smith et al. [235] solve extremely high-dimensional problem instances, employing RBFN models.

All these modeling approaches use distances and kernels in a standard, straightforward way. However, it is often important to consider if the employed distances are Conditionally Negative Semi-Definite (CNSD) or if the derived kernels are Positive Semi-Definite (PSD). Definiteness is a frequent requirement for modeling methods based on distances or kernels. Dealing with the possible indefiniteness of a function is

hence of utmost concern. More details on this are discussed in Chapters 7 and 8.
While kernel-based models are rarely applied in discrete optimization, they are more frequently applied for machine learning with discrete data in general. One early example are the convolution kernels proposed by Haussler [116]. They propose to generate a kernel for discrete data via the convolution of kernels on parts of the discrete objects. Kondor and Lafferty suggest a diffusion kernel to model graphs and discrete input spaces with Gaussian processes or SVMs [151]. The diffusion kernel is closely related to random walks, which are also related to the graph kernels proposed by Smola and Kondor [237]. Focusing on SVMs, Gärtner gives an early survey on kernels for *structured data* [98]. By structured data, they imply any data that is not *"represented in a single table"* [98]. They include (structured) strings, trees, and graphs into this definition. A unified framework for kernels on graphs is presented by Vishwanathan et al. [253]. Bellet et al. [31] give an overview of metric learning for structured data. They also discuss how edit distances can be learned for strings or graphs. They note that the field of structured data is less explored than numerical data. Paassen et al. [194] propose distances and kernels to produce a Gaussian process model of a time series of structured, discrete data. Discrete kernels for strings or graphs are also suggested for Gaussian process models as, e.g., described in the Work by Rasmussen and Williams [206].
Recently, Gaussian process models have also been applied for the optimization of the structure of neural networks (neuroevolution). Gaier et al. propose to use the compatibility distance for that purpose, which is mostly based on a weighted sum of differences between shared edges or nodes [96].
Most of the aforementioned references make use of distance measures in genotype space. For the case of genetic programming, Hildebrandt and Branke [121] show that a distance in phenotype space may be an excellent choice, see also the preceding section. A phenotypic distance for symbolic regression is investigated by Zaefferer et al. [272].

3.8 Alternative Models

Up to now, we did not take the important class of probabilistic models into account. Examples of combinatorial optimization algorithms employing such models are Estimation of Distribution Algorithms (EDA) and Ant Colony Optimization (ACO) [276, 115]. Compared to the earlier presented works, these algorithms usually operate under different assumptions. They are often applied to less expensive (although not necessarily cheap) problems. Still, they can be considered to use surrogate models. If necessary, a mapping between the probabilistic model and objective function values can usually be established (cf. [230, 46]). Examples for expensive optimization with ACO are, e.g., the works by Cáceres et al. [199] and Teixeira et al. [245]. Examples for expensive optimization with EDAs include building design [47] and reactor fuel management [134].
The probabilistic models used in EDAs can also be classified into our six strategies. For instance, there are EDAs that employ a mapping approach (STR-4), e.g., EDAs for

permutation problems using the random key mapping [57]. Other EDAs use problem-specific solutions (STR-2). One example is the EDA based on the (generalized) Mallows model for permutations [168, 59, 58].
Another relevant model class are graph models like Bayesian networks or Markov random fields (MRFs). These are of interest because they may directly encode the graph structure of the candidate solution representations [6]: directed graphs in Bayesian networks and undirected graphs in MRFs. Graph-based models are closely linked to distribution-based approaches, e.g., [197, 157, 230, 46, 47]. Mostly, these approaches can be classified as inherently discrete models (STR-3).

3.9 Conclusions

The six strategies for surrogate modeling in discrete search spaces are not mutually exclusive. For example, kernel-based models (STR-6) can be re-interpreted as mapping approaches (STR-4): As discussed in Section 4.1, kernels can be understood to implicitly map from the original space to a larger, or even infinite-dimensional space.
With that said, the main benefit of our classification into six strategies is not that it provides a sharp distinction between the different strategies. Rather, these different approaches may support the thought process required to design suitable models. Thinking about a specific problem in terms of extracted features, mapping functions, or similarity measures may help to devise an efficient model.
The problem type and application restrictions govern the result of this design process. Even a naive approach (STR-1) may be adequate, if the problem is sufficiently simple. If the necessary prior knowledge is available, customized models are probably well suited (STR-2). If the data structures match to the corresponding models, inherently discrete models (STR-3), such as random forests, are clearly a promising choice.
However, complex search spaces can render STR-1 and 3 infeasible, and the black-box nature of a problem may eliminate the option to use STR-2. Hence, the strategies STR-4, 5 and 6 tend to be more versatile for the problems discussed in this thesis. We focus on kernel-based models (STR-6) and mostly use Kriging. Still, some approaches we employ could easily be interpreted in the context of mapping and feature extraction. For instance, the Arc-kernel later discussed in Section 9.2.1 could be understood to use a mapping from a complex hierarchical search space into a space where a standard kernel can be used (STR-4). Or else, the phenotypic distance examined in Section 9.1.3 can be understood to measure distance in a phenotypic feature space (STR-5).

Chapter 4

Kernels: Terminology and Definitions

As mentioned in Chapter 3, kernel-based models are at the center of this thesis. This chapter provides the necessary foundations, by discussing the background on kernels, similarity measures, distances, and related concepts.
Following Murphy [187], we define a kernel as a real-valued function $\mathrm{k}(x, x')$ with

$$\begin{aligned} \mathrm{k} : \mathcal{X} \times \mathcal{X} &\to \mathbb{R} \\ (x, x') &\mapsto \mathrm{k}(x, x') \end{aligned} \tag{4.1}$$

that is usually symmetric $\mathrm{k}(x, x') = \mathrm{k}(x', x)$ and non-negative $\mathrm{k}(x, x') \geq 0$. In general, we make no further assumptions about $\mathrm{k}(x, x')$ or the nonempty set $\mathcal{X}$. Mostly, the kernels employed in this thesis express some sort of dissimilarity or similarity between the samples or candidate solutions x, x'. In the following, some issues of particular interest are discussed. We start with an important motivation of employing kernels, i.e., the kernel trick. Afterwards, we give some general terms and definitions. Most of the specific kernels used in this thesis are described in Appendix B.

4.1 Motivation: The Kernel Trick

One reason for the popularity of kernel-based models is the so-called kernel trick [224]. When linear models are faced with non-linear learning tasks, the kernel trick allows mapping the original data into a higher-dimensional space (sometimes called the feature space [206]) where the problem becomes solvable by linear methods. The kernel trick allows computing dot products in that higher-dimensional feature space, without having to compute the required mapping directly [224].

In less detail, some of the terms and definitions from this chapter have already been discussed in section 2 of *"An Empirical Approach for Probing the Definiteness of Kernels"* by Zaefferer et al. [266]. Hence, there is some overlap with that section. We added more details, including an illustrative motivation of the kernel trick.

Example 4.1.1. Let us assume that we intend to model the one-dimensional function $f(x) = x^2$ with $x \in [-1, 1]$ using a simple linear regression model with first order effects only, that is,

$$y = \beta_0 + \beta_1 x + \epsilon, \tag{4.2}$$

where β_i are model coefficients, and ϵ is independent, normally distributed error with zero mean and variance σ^2. We sample 11 equidistant points in the given interval and evaluate them with $f(x)$. As was to be expected, the resulting model does not represent this data very well, as is shown in Fig. 4.1.

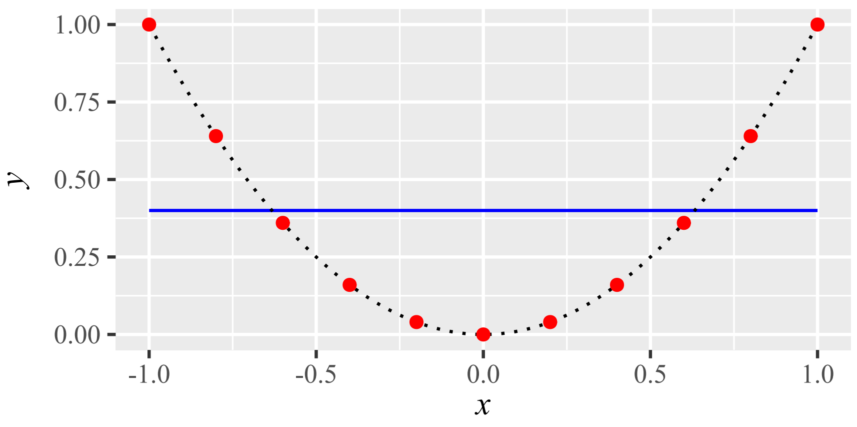


Figure 4.1: *The prediction of the linear regression model from Eq.* (4.2) *(blue solid line) based on 11 data samples (red dots) evaluated with $f(x) = x^2$ (black dotted line).*

Applying the kernel trick, we can map our data into a higher-dimensional space. For this simple example, a two-dimensional feature space suffices. We choose two samples $x^{(a)} = -0.5$ and $x^{(b)} = 0.5$ as centers for our mapping, as well as a kernel function $\mathrm{k}(x, x') = e^{-\theta(x-x')^2}$. Here, we assume that the kernel parameter $\theta = 0.1$ is known a priori. This yields for every $x \in [-1, 1]$ a corresponding value $\mathrm{k}(x, x^{(a)})$, and $\mathrm{k}(x, x^{(b)})$. We can now train a simple linear regression model with these two features, that is,

$$y = \beta_0 + \beta_1 \mathrm{k}(x, x^{(a)}) + \beta_2 \mathrm{k}(x, x^{(b)}) + \epsilon. \tag{4.3}$$

This yields the near-perfect fit depicted in Fig. 4.2. As it is still of a low dimensionality, we can plot the model and the training data in the two-dimensional feature space. The respective plot is given in Fig. 4.3. The original space is embedded as a curve in the feature space, indicated by the dots that represent the training data. Points that do not reside along that curve in feature space have no correspondence in the original, one-dimensional space.

By using more than just two centers, we could easily extend this example to more complex non-linear functions. If we used all training data samples as centers, we would receive a model based on the 11×11 kernel matrix $\mathbf{K}$. This model would be very similar to the Kriging model introduced in Section 2.3.1.

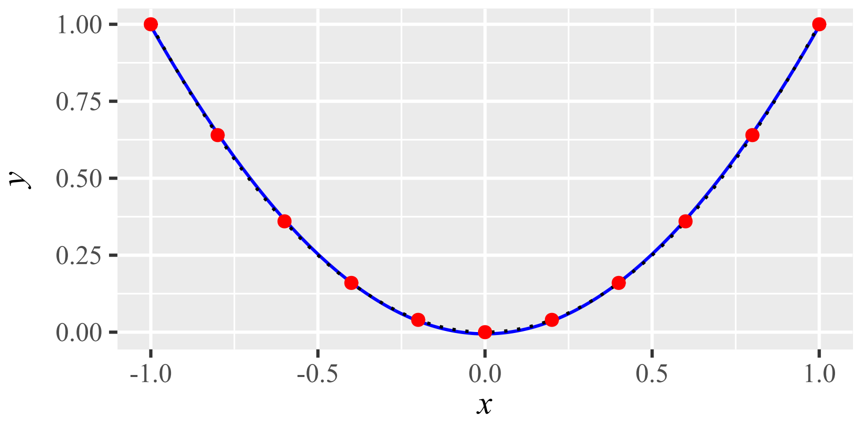


Figure 4.2: *The prediction of the linear regression model from Eq.* (4.3) *employing the kernel trick (blue solid line) based on 11 data samples (red dots) evaluated with $f(x) = x^2$ (black dotted line).*

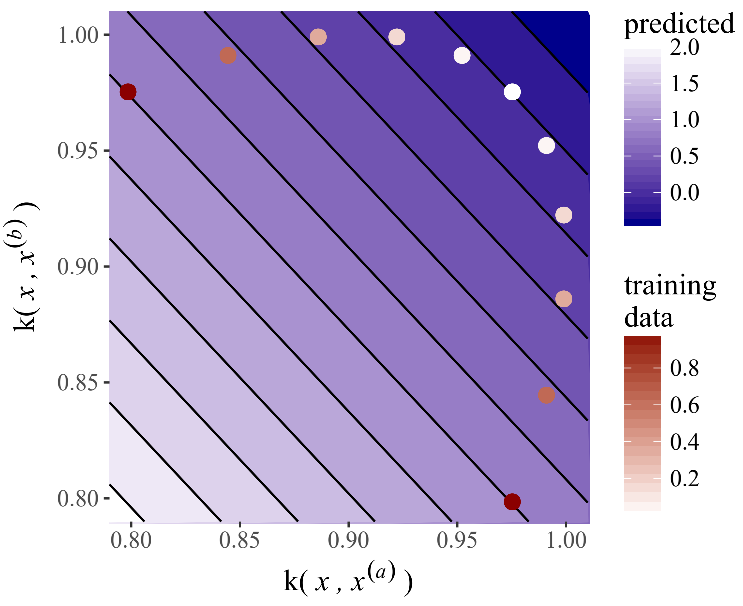


Figure 4.3: *The prediction of the linear model from Eq.* (4.3) *in the two-dimensional feature space spanned by the kernel function. The dots represent the training data mapped into the feature space. The solid lines are isolines of the predicted values.*

4.2 Similarity Measures

Similarity measures are kernels that express how much two objects (samples or candidate solutions) resemble each other. Similarity measures are often employed in methods such as RBFN or SVM [224].
Kriging, which is of particular interest in this thesis, uses a specific type of similarity measure: correlation functions. Correlation functions are kernels $\mathrm{k}(x, x')$ that fulfill certain requirements:

- Correlations are restricted to values $|\mathrm{k}(x, x')| \leq 1$.
- Identical samples are perfectly correlated $\mathrm{k}(x, x) = 1$.
- Correlation functions should be Positive Semi-Definite (PSD) (see Section 4.4.2 for a definition).

The latter can more intuitively be phrased as the requirement that the weighted sum of random variables has a variance, which is necessarily non-negative. More details on the concept of definiteness are discussed in Section 4.4, Chapter 7 and Chapter 8.
In the context of Kriging, correlation functions usually do not yield negative values. This expresses the intuition that correlation decreases the more disparate two samples get and approaches zero for large distances. Correlation functions can also be transferred to covariance functions. In the stationary case, this is simply done by multiplying with the constant variance $\sigma^2\mathrm{k}(x, x')$. As multiplication with a scalar does not affect definiteness, covariance functions should also be PSD. The diagonal of a covariance matrix should be non-negative because it represents a variance.

4.3 Dissimilarity Measures

Another important class of kernels are dissimilarity measures. They are the complement to similarity measures, and express how disparate two objects or samples are. That is, a measure of dissimilarity should become larger the more distinct two compared objects x and x' are.
In the following, two types of dissimilarity measures are introduced: distance measures and distance metrics. They find frequent use in this thesis. Due to the focus on discrete, combinatorial data structures, the concept of edit distance measures is also introduced.

Distance Measure *Distance measures* (or distance functions) compute a dissimilarity $\mathrm{d}: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$ of two objects $x, x' \in \mathcal{X}$.
For a set of $n \in \mathbb{N}$ objects, the $n \times n$ distance matrix $\mathbf{D}$ collects the pairwise distances $\mathbf{D}_{ij} = \mathrm{d}(x^{(i)}, x^{(j)})$ for all $i, j = 1, \ldots, n$. A distance value is often expected to satisfy some intuitive conditions. For instances, it should not become negative.

Distance Metric The term *distance metric* implies a set of more formal conditions. By definition, a distance measure $\mathrm{d}(x, x')$ is a distance metric if

- it is symmetric $\mathrm{d}(x, x') = \mathrm{d}(x', x)$,
- non-negative $\mathrm{d}(x, x') \geq 0$,
- preserves identity $\mathrm{d}(x, x') = 0 \iff x = x'$,
- and satisfies the triangle inequality $\mathrm{d}(x, x'') \leq \mathrm{d}(x, x') + \mathrm{d}(x', x'')$.

For example, the Euclidean distance is a metric, while the squared Euclidean distance is not: it violates the triangle inequality. Distance measures that do not preserve identity are often called pseudo-metrics. That means, a pseudo-metric can yield zero, although $x \neq x'$.

Edit Distance One specific class of distance measures are edit distance measures. Edit distance measures count the minimal number of edit operations required to transform one object into another. These edit operations are small, discrete changes of the respective object (or sample). Their exact nature depends on the application. For instance, tree structures may require different operations than strings or sequences. An edit distance measure may concern one specific edit operation (e.g., only swaps of elements in a sequence) or a set of different operations (e.g., the Levenshtein distance with substitution, deletion, and insertion operations). For combinatorial or structured data, edit distance measures play an important role because they are often an intuitive choice to express distance between samples.
Due to their definition, it is usually simple to show that edit distance measures are metrics:

- Non-negativity: Edit distance measures count operations. A count cannot be negative.
- Identity preservation: Identical objects do not require any edit operations to transform one into the other. Hence, their edit distance is zero. Non-identical objects always require some edit operation(s). Thus, their distance is non-zero.
- Symmetry: Clearly, turning object x into x' or x' into x requires the same set of edit operations, only in reverse. Hence, edit distance measures are nearly always symmetric. An exception could be caused by an edit operation that is not reversible by a single operation.
- Triangle inequality: Due to the definition of edit distance measures (*minimal* number of operations), the triangle inequality cannot be violated. Any distance along an indirect path via a third point $\mathrm{d}(x, x') + \mathrm{d}(x', x'')$ can be seen as a combined set of consecutive edit operations. As the distance along the direct path $\mathrm{d}(x, x'')$ is by definition a minimal number of operations, it cannot be larger than the indirect path.

There are special cases where edit distance measures operate under certain restrictions that let them violate the triangle inequality (e.g., the restricted Damerau Levenshtein distance [44]).

4.4 Definiteness

Kriging usually requires kernels to be definite. This requirement is also common to many other kernel-based models, such as SVMs [223]. This section first introduces the concept of definite matrices, and then extends the concept to functions. We refer to the literature for more in-detail descriptions and proofs that are the foundation of the following considerations [32, 223, 53].

4.4.1 Definiteness of Matrices

A symmetric, square matrix $\mathbf{A}$ of dimension $n \times n$ ($n \in \mathbb{N}$) is Positive Definite (PD) if and only if

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \mathbf{A}_{ij} > 0, \tag{4.4}$$

for all $\mathbf{c} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, where $\mathbf{0}$ is a vector of zeros. This is equivalent to $\mathbf{c}^{\mathrm{T}}\mathbf{A}\mathbf{c} > 0$, and also equivalent to all eigenvalues being positive, $0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. Due to symmetry, the eigenvalues are $\lambda \in \mathbb{R}^n$.
Respectively, a negative definite (ND) matrix implies negative eigenvalues ($\lambda_n < 0$) and $\mathbf{c}^{\mathrm{T}}\mathbf{A}\mathbf{c} < 0$. If some eigenvalues are zero, the matrix is called Positive or Negative Semi-Definite (PSD, NSD), i.e., $\mathbf{c}^{\mathrm{T}}\mathbf{A}\mathbf{c} \geq 0$ or $\mathbf{c}^{\mathrm{T}}\mathbf{A}\mathbf{c} \leq 0$.
If mixed signs are present, the matrix may be called indefinite. Kernel or correlation matrices are examples of matrices that are often required to be PSD.
A broader set of matrices are Conditionally PSD or NSD (CPSD, CNSD) matrices. Here, the coefficients $\mathbf{c}$ have to satisfy

$$\sum_{i=1}^{n} c_i = 0, \tag{4.5}$$

with $n > 1$. All PSD (NSD) matrices are CPSD (CNSD). Conditionally definite matrices do also have a relation to eigenvalues. CNSD matrices have at most one positive eigenvalue [18]. Symmetric, non-zero, non-negative CNSD matrices (e.g., distance matrices) have exactly one positive eigenvalue [18]. According to Ikramov [130], we can check whether $\mathbf{A}$ is CNSD as follows. Let the $n \times n$ matrix $\mathbf{P}$ be[1]

$$\mathbf{P} = \begin{bmatrix} \mathbf{I}_{n-1} - \mathbf{1}\,\mathbf{1}^{\mathrm{T}}/n & -\mathbf{1}/n \\ \begin{bmatrix}0 & \ldots & 0\end{bmatrix} & 1 \end{bmatrix}, \tag{4.6}$$

[1] The definition of $\mathbf{P}$ in Eq. (4.6) is only one option. Any non-singular matrix is feasible, if all but the last row sums are zero, and the last row sum is one.

with $\mathbf{1} = \begin{bmatrix}1 & \dots & 1\end{bmatrix}^{\mathrm{T}}$ and $\mathbf{I}_{n-1}$ is the $(n-1) \times (n-1)$ identity matrix. Then, the $n \times n$ matrix $\mathbf{B}$ is

$$\mathbf{B} = \mathbf{P}\mathbf{A}\mathbf{P}^{\mathrm{T}},$$

then $\mathbf{A}$ is CNSD if and only if the matrix

$$\hat{\mathbf{A}} = \mathbf{B}_{n-1} \tag{4.7}$$

is NSD [130]. Here, $\mathbf{B}_{n-1}$ is $\mathbf{B}$ without its last row and column. The largest eigenvalue of $\hat{\mathbf{A}}$, which determines whether $\mathbf{A}$ is CNSD, will be denoted by $\hat{\lambda}$.

4.4.2 Definiteness of Kernels

The definition of definiteness can be extended from matrices to functions, which is especially important for kernels. In the literature, the respective notations and definitions are not used consistently. These inconsistencies are due to historical reasons. For instance, in linear algebra, *positive definite* is used to describe the relation (... > 0) and *positive semi-definite* for $(\ldots \geq 0)$. In functional analysis *positive definite* is used for $(\ldots \geq 0)$ and the relation $(\ldots > 0)$ is denoted with *strictly positive definite*. We use the notation from linear algebra in this thesis. Similarly, *conditionally negative definite* is often abbreviated with *negative definite*. We always indicate the conditional nature of a kernel (or matrix), so that the notation respects the basic symmetry between the definitions of *negative definite* and *positive definite*. The upcoming description roughly follows the definitions and notations from the relevant literature [32, 223, 53].
For the nonempty set $\mathcal{X}$, a symmetric kernel $\mathrm{k}(x, x')$ is called PSD if and only if

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \mathrm{k}(x^{(i)}, x^{(j)}) \geq 0,$$

for all $n \in \mathbb{N}$, $x^{(1)}, \ldots, x^{(n)} \subseteq \mathcal{X}$ and $c_1, \ldots, c_n \subseteq \mathbb{R}$. A PSD kernel always yields PSD kernel matrices.
Equivalently, a kernel is negative semi-definite if and only if

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \mathrm{k}(x^{(i)}, x^{(j)}) \leq 0, \tag{4.8}$$

for all $n \in \mathbb{N}$, $x^{(1)}, \ldots, x^{(n)} \subseteq \mathcal{X}$ and $c_1, \ldots, c_n \subseteq \mathbb{R}$.
Analogous to the matrix case, conditionally definite functions imply that the coefficients $\mathbf{c}$ satisfy the condition in Eq. (4.5). The importance of CNSD functions is because the distance measure $\mathrm{d}(x, x')$ is CNSD if and only if the kernel $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x'))$ is PSD $\forall\ \theta > 0$ [223]. One example of a CNSD function is the Euclidean distance.
Another important fact is that the product of two PSD kernels is also PSD [32]. This also implies that the linear combinations discussed in Chapter 6 lead to PSD kernels.

It is often stated that kernels must be PSD [206, 70]. In fact, PSDness of a kernel implies that there exists a mapping to an infinite-dimensional feature space, in which the kernel acts as a dot product [224, 206] (cf. the kernel trick in Section 4.1). Still, even an indefinite kernel function may yield a PSD kernel matrix. This may depend on the specific data set used to train the model [50, 162] as well as the parameters of the kernel function. Some frequently used kernels are known to be indefinite. Examples are the sigmoid kernel [238, 55] or time-warp kernels for time series [171].
To summarize this section, Fig. 4.4 depicts the most important relations with respect to definiteness, as used throughout this thesis.

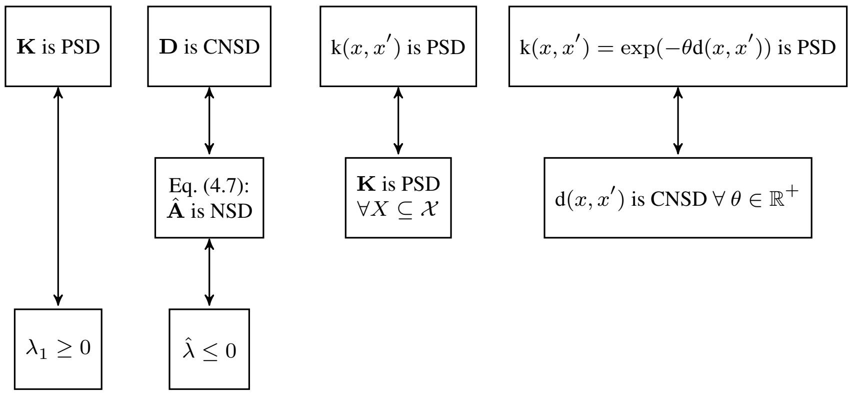

$\mathbf{K}$ is PSD
$\mathbf{D}$ is CNSD
$\mathrm{k}(x,x')$ is PSD
$\mathrm{k}(x,x') = \exp(-\theta \mathrm{d}(x,x'))$ is PSD
Eq. (4.7): $\hat{\mathbf{A}}$ is NSD
$\mathbf{K}$ is PSD $\forall X \subseteq \mathcal{X}$
$\mathrm{d}(x,x')$ is CNSD $\forall\, \theta \in \mathbb{R}^+$
$\lambda_1 \geq 0$
$\hat{\lambda} \leq 0$

Figure 4.4: *An overview of the most important relations with respect to definiteness of matrices and kernel functions, as considered in this thesis.*

4.5 Conditioning of Matrices

A concept that is related to the definiteness of matrices is the conditioning of matrices. The condition number of a symmetric matrix is the ratio of the largest and smallest eigenvalue, $\kappa = |\lambda_n|/|\lambda_1|$. A matrix is ill-conditioned if the condition number κ is large. Here, large usually implies that the reciprocal of κ *"approaches the machine's floating-point precision"* [203]. Conversely, if κ is small, the matrix is well-conditioned. We do not focus on ill-conditioning, but will occasionally remark on it because it affects numerical stability and model quality.
We usually do not discuss the condition number of distance matrices. The numerical algorithms required for models like Kriging operate on the correlation matrix. The correlation matrix is usually based on a distance matrix, but ill-conditioning of one does not imply ill-conditioning of the other (unless we would use a different definition of κ for distance matrices).

Part II

Main Contributions

Chapter 5

Kriging-based Combinatorial Optimization

This chapter comes back to the Kriging-based SMBO algorithms discussed in Chapter 2. In the following, we discuss how Kriging can be applied to arbitrary combinatorial, discrete search spaces. Furthermore, we describe experiments with a set of artificial test-problems, to demonstrate that this approach may indeed improve upon model-free optimization algorithms. Hence, the core research questions of this chapter are:

Question 5.1. Applicability. Can SMBO algorithms like EGO be applied to combinatorial or discrete search spaces?

Question 5.2. Comparison. How well do the respective algorithms compare to model-free optimization techniques?

5.1 Kriging for Combinatorial Data

With the definitions of Kriging in Chapter 2 and kernels in Chapter 4, the step from a real, vector-valued Kriging model to a discrete model is straightforward. We can make comparable steps with most other kernel-based models, but focus on Kriging due to its beneficial features, as described in Chapter 2.

The following sections of this chapter are partially based on the article *“Efficient Global Optimization for Combinatorial Problems”* by Zaefferer et al. [273]. Occasionally, text elements have been adopted verbatim from that publication. Overall, the text was significantly rewritten and extended before its inclusion into this thesis. We also added an illustrative example. The described experiments were repeated in a more thorough way. This included a more flexible, self-adaptive evolutionary algorithm, a broader set of test functions, and a parameter sensitivity study.

Essentially, we exploit the fact that Kriging is mostly based on a kernel (or correlation function), e.g., the exponential kernel

$$\mathrm{k}(x, x') = \exp(-\theta ||x - x'||).$$

Our goal is to modify or replace these functions, so that x is no longer restricted to be a real-valued vector. In fact, we can write this specific kernel function as

$$\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x')), \tag{5.1}$$

where $\mathrm{d}(x, x')$ is a distance measure. Many kernel functions can be rewritten in a similar way. Intuitively, the underlying distance measure $\mathrm{d}(x, x')$ can be replaced by any measure that is well suited for the respective discrete search space $\mathcal{X}$, e.g., an edit distance measure for sequences or graphs. Thus, Kriging is applicable to any search space $\mathcal{X}$ where a respective kernel or distance measure is available.
Of course, these considerations rely on a Kriging model that only exploits the correlations, and not the underlying input data. There are Kriging variants that employ polynomial trend functions, which describe the behavior of the data when predicting far away from any training samples. In our case, the prediction would tend to the process mean μ in these cases. More complex, polynomial trend functions are not directly transferable to arbitrary search spaces.

Example 5.1.1. Let us assume that we are dealing with a problem where the ordering (or permutation) of actions has to be optimized such that some objective value is minimal. For this simple example, the artificial objective function is the swap distance to a reference permutation $x_{\text{ref}} = \begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix}$, that is, $f(x) = \mathrm{d}_{\text{Swa}}(x, x_{\text{ref}})$. The swap distance is the minimal number of swaps (exchanges of neighboring elements) required to turn one permutation into another. The optimization starts with four initial samples,

$$\begin{matrix} x^{(1)} = [1 & 2 & 4 & 3] \\ x^{(2)} = [1 & 4 & 3 & 2] \\ x^{(3)} = [2 & 1 & 3 & 4] \\ x^{(4)} = [3 & 2 & 4 & 1] \end{matrix} \quad \text{with} \quad \begin{matrix} f(x^{(1)}) = 1 \\ f(x^{(2)}) = 3 \\ f(x^{(3)}) = 1 \\ f(x^{(4)}) = 4 \end{matrix}.$$

To model this data with Kriging, we need to compute a distance matrix. Again, we choose the swap distance. We evaluate the swap-distance between the four permutations. The matrix $\mathbf{D}$ collects all pairwise swap distances,

$$\mathbf{D} = \begin{bmatrix} 0 & 2 & 2 & 3 \\ 2 & 0 & 4 & 3 \\ 2 & 4 & 0 & 3 \\ 3 & 3 & 3 & 0 \end{bmatrix}.$$

These distances can now be plugged into a corresponding kernel or correlation function, $\mathbf{K}_{i,j} = \exp(-\theta \mathbf{D}_{i,j})$. Next, we use maximum likelihood estimation to derive the kernel parameter θ, which is estimated to be $\theta \approx 1.96$. The other parameters of the model are the process mean $\hat{\mu} \approx 2.26$ and the process variance $\hat{\sigma}^2 \approx 1.68$. These are used in the predictor in Eq. (2.4) and the uncertainty estimator in Eq. (2.5). For a new sample $x^* = \begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix}$, the prediction is $\hat{y}(x^*) \approx 1.91$ and the uncertainty estimate is $\hat{s}^2(x^*) \approx 1.62$.

5.2 Combinatorial Efficient Global Optimization

With the corresponding adaptation of Kriging for combinatorial data, it is straightforward to apply the EGO algorithm (see Section 2.4) to combinatorial search spaces. We refer to this combinatorial variant of EGO as CEGO. It is outlined in Algorithm 4. CEGO is closely related to the earlier introduced SMBO algorithm (Algorithm 3). In fact, the general procedure is largely identical. The differences occur mainly in the choice of the operators: `init()`, `model()`, and `optimizer()`.

The `init()` operator generates an experimental design, which is an initial set of candidate solutions. Since many classical design of experiment methods do not apply to arbitrary combinatorial data structures, the design is often based on some form of random sampling. This can be extended by choosing designs that maximize the minimum distance between samples, to avoid closely spaced solutions in the initial design X. This is denoted as a max-min design.

Models based on suitable kernels or distance functions for combinatorial data are generated by `model()`. In this thesis, the model of choice is Kriging, yet most consider-

Algorithm 4 The CEGO algorithm

```
1: function CEGO($f(x)$, init(), k$(x, x')$, model(), optimizer(), infill, terminate())
2:     $X$ = init();                              ▷ create initial design with $X \subset \mathcal{X}$
3:     $n = |X|$;
4:     for $j = 1$ to $n$ do
5:         $y_j = f(x^{(j)})$;                    ▷ evaluate design
6:     end for
7:     $\mathbf{y} = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^\mathrm{T}$;
8:     while not terminate() do
9:         $M$ = model $(X, \mathbf{y}, \mathrm{k})$;        ▷ create / update the kernel-based model
10:        $x'$ = optimizer($M$,infill);          ▷ determine promising candidates
11:        $y' = f(x')$;                          ▷ evaluate candidate
12:        $X = X \cup \{x'\}$;                   ▷ add candidate to data set
13:        $\mathbf{y} = \begin{bmatrix} \mathbf{y} \\ y' \end{bmatrix}$;   ▷ add objective function value of candidate
14:    end while
15: end function
```

ations may as well apply to other kernel-based models. In addition, the kernel function $\mathrm{k}(x, x')$ has to be supplied to the algorithm since it is a requirement of the `model()`.

A suitable optimization algorithm is indicated by `optimizer()`. This algorithm is often a variant of an EA (see Section 2.1) because EAs are easily adapted to arbitrary discrete data types. While other `infill` criteria are possible, the `optimizer()` usually maximizes the EI infill criterion.

Finally, `terminate()` checks whether the CEGO algorithm should stop, e.g., based on the limited budget of evaluations of $f(x)$ or based on a desired objective value.

Example 5.2.1. Following up on Example 5.1.1, we use the derived model to perform a minimization of $f(x)$. To that end, we first minimize the predicted mean of the Kriging model. Since we deal with permutations with $m = 4$ elements, we can simply use brute force to predict all 24 permutations. This yields two potential local optima, $x^* = \begin{bmatrix}1 & 2 & 4 & 3\end{bmatrix}$ and $x^* = \begin{bmatrix}2 & 1 & 3 & 4\end{bmatrix}$. Both are predicted to have exactly $\hat{y}(x^*) = f(x^*) = 1$. This is no surprise, since both candidate solutions are part of the training data and have already been observed.

To account for that, we could exclude the training data and select the most promising solution from the remaining 20 candidates. Or else, we follow the concept of CEGO, using the expected improvement criterion. With Eq. (2.6), we determine that the largest expected improvement is estimated for $x^* = \begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix}$, which is in fact the true optimum of $f(x)$. The complete data with all samples, predictions, and true values is given in Table 5.1.

Table 5.1: *All permutations with $m = 4$ elements, the predicted mean $\hat{y}(x)$, the uncertainty estimate $\hat{s}^2(x)$, the negative logarithm of the EI, and the underlying objective function values $f(x)$, based on Examples 5.1.1 and 5.2.1. The best predicted value (excluding training data), the best function value, and the best EI are printed in bold (lower values are better).*

x	$\hat{y}(x)$	$\hat{s}^2(x)$	-log(EI)	$f(x)$	x	$\hat{y}(x)$	$\hat{s}^2(x)$	-log(EI)	$f(x)$
[1 2 3 4]	**1.91**	1.62	**0.75**	**0.00**	[4 3 2 1]	2.30	1.69	0.97	6.00
[1 2 4 3]	1.00	0.00	∞	1.00	[3 4 2 1]	2.28	1.69	0.96	5.00
[1 4 2 3]	2.36	1.65	1.02	2.00	[3 2 4 1]	4.00	0.00	∞	4.00
[4 1 2 3]	2.24	1.69	0.93	3.00	[3 2 1 4]	2.24	1.69	0.93	3.00
[4 1 3 2]	2.29	1.69	0.96	4.00	[2 3 1 4]	2.26	1.69	0.94	2.00
[1 4 3 2]	3.00	0.00	∞	3.00	[2 3 4 1]	2.50	1.65	1.11	3.00
[1 3 4 2]	2.22	1.62	0.94	2.00	[2 4 3 1]	2.40	1.65	1.05	4.00
[1 3 2 4]	2.23	1.68	0.92	1.00	[4 2 3 1]	2.51	1.65	1.12	5.00
[3 1 2 4]	2.08	1.65	0.84	2.00	[4 2 1 3]	2.26	1.69	0.94	4.00
[3 1 4 2]	2.46	1.65	1.09	3.00	[2 4 1 3]	2.28	1.69	0.95	3.00
[3 4 1 2]	2.27	1.69	0.94	4.00	[2 1 4 3]	1.95	1.62	0.77	2.00
[4 3 1 2]	2.27	1.69	0.95	5.00	[2 1 3 4]	1.00	0.00	∞	1.00

5.3 Proof of Concept

With the adaptation introduced in the previous sections, Kriging-based SMBO techniques can be extended to combinatorial search spaces, and, more generally, to any search space for which a suitable kernel is available. Since Kriging has been originally established for continuous domains, this may seem counterintuitive. To demonstrate that this can still be successful in practice, this section provides an empirical study: We compare a model-free EA with a Kriging-based EGO algorithm that internally employs the same model-free EA.
To that end, a set of artificial test-problems has been selected. These are neither fully black-box, nor are they expensive to evaluate. This allows us to perform a sufficient number of experiments to determine the performance of the tested algorithms.

5.3.1 Case Study: Kriging with Hamming Distance

Test Problems Test functions for two different data types were selected: permutations and binary strings. The permutation problems are listed in the following.

- One instance of the Traveling Salesperson Problem (TSP) was chosen, the bayg29 instance with 29 locations from the TSPLIB [211]. In the TSP, the cost or length of a route through several locations is minimized. Each location has to be visited once. Here, the route should be circular, that is, the traveler returns to the start location after visiting all other locations.

- An instance of the Asymmetric TSP (ATSP) was generated. Therefore, a distance matrix for 20 locations was generated by uniform random sampling. In contrast to the TSP, the cost of traveling between two locations depends on the direction. A non-circular variant was chosen. That is, each location is visited exactly once, without a return to the start location.

- The Quadratic Assignment Problem (QAP) instance tho30 from the QAP Library (QAPLIB) [52] was chosen. In the QAP [51] m facilities have to be assigned to m locations. The cost of an assignment is based on the flow between facilities and the distance between locations. Hence, the optimization problem is to find a permutation that minimizes the assignment cost.

- The Flow-shop Scheduling Problem (FSP) [243] concerns the sequencing of m jobs on several machines. An optimal sequence should minimize the makespan, that is, the production time required for the whole set of jobs. It depends on the processing time of each job on each machine. We chose the FSP instance reC05 [209] from the OR-Library [30].

- An instance of the single-machine total Weighted Tardiness Problem (WTP) [1] is chosen from the OR-Library [30] (the second of length 40, i.e., wt40b). Here, m jobs are sequenced on one machine that can handle one job at a time. The tardiness of a schedule for all jobs (weighted by a set of m given weights) is minimized. The optimal tardiness depends on the given processing times and due dates of each job.

- Moraglio et al. [181] proposed so-called unimodal (UNI) problems as simple and transparent test problems for combinatorial surrogate models. In the UNI problems, the distance of a candidate solution to some reference solution has to be minimized. For permutations, the distance to the fixed reference permutation $x_{\text{ref}} = \begin{bmatrix}1 & 2 & \dots & m\end{bmatrix}$ has to be minimized. This is identical to the test function used in Examples 5.1.1 and 5.2.1. That is, the objective function is $f(x) = \mathrm{d}(x, x_{\text{ref}})$. The individual problem instances chosen for this study are pUNIH30 (based on Hamming distance), pUNIS30 (swap distance), and pUNII30 (interchange distance). Here, the number 30 indicates the number of elements in the permutation, m.

For the experiments with binary strings, we chose the following problem instances.

- Three NK-Landscape instances were generated. NK-Landscapes (NKL) are fitness landscapes based on bit strings. They have been proposed by Kauffman [146]. Here, the fitness of a string is the sum of contributions of all $N = m$ bits, each affected by $K = n_{\text{bit}}$ other bits. For each bit, a function $g_i(x)$ assigns a real-valued weight to each possible combination of the bit and its neighbors. The weights are typically uniform random samples from the interval $[0, 1]$. In the present work, the n_{bit} bits that impact the contribution of the i-th bit x_i are the neighbors given by the sequence $\begin{bmatrix}x_{i+1} & \dots & x_{i+n_{\text{bit}}}\end{bmatrix}$. The three generated instances are N30K2, N30K4 and N30K8, with $m = 30$ and $n_{\text{bit}} \in \{2, 4, 8\}$.

- Two Max-Cut problems were generated [103], one with $m = 30$ (MC30) and one with $m = 40$ (MC40) bits. The Max-Cut problem is a graph problem, where the goal is to cut a graph into two subgraphs, such that the number of edges between them is maximal. Each bit indicates to which subgraph a specific node belongs.

- Three unimodal problems (UNI) were generated, similarly to the permutation case. The individual problem instances are bUNIH30 (binary, Hamming distance), bUNILE30 (binary, Levenshtein distance), and bUNILC30 (binary, Longest Common Substring distance (LCStr)).

An overview of all these problem instances is given in Table 5.2.

Model-Free Optimization Settings As a baseline, we used Random Search (RS) and a model-free EA to solve the test problem instances. The EA is based on the descriptions in Section 2.1. Details of the variation operators can be found in Appendix C. The following configuration was employed:

Table 5.2: *Overview of the problem instances used in the experiments. The problems are intended for a first test of the applicability of SMBO to combinatorial problems.*

data type	problem	instance	dimensionality
permutation	TSP	bayg29	29
permutation	ATSP	atsp20	20
permutation	QAP	tho30	30
permutation	WT	wt40b	40
permutation	FSP	reC05	20
permutation	UNI	pUNIH30	30
permutation	UNI	pUNIS30	30
permutation	UNI	pUNII30	30
binary string	NKL	N30K2	30
binary string	NKL	N30K4	30
binary string	NKL	N30K8	30
binary string	MC	MC30	30
binary string	MC	MC40	40
binary string	UNI	bUNIH30	30
binary string	UNI	bUNILE30	30
binary string	UNI	bUNILC30	30
binary string	UNI	pUNII30	30

- **budget:** The budget of fitness function evaluations was set to 100, under the artificial assumption that evaluations are expensive.

- **replications:** Each algorithm run was repeated 20 times, with different random number generator seeds.

- **archiving:** If the EA suggested a solution that had already been tested during that specific run, the fitness function was not evaluated. Instead, the fitness was read from an archive of previous evaluations.

- **mutation:** In case of the permutation problems, swap, interchange, insert and reversal mutation operators were used (see Section C.1.1). In case of the binary strings, the chosen mutation operators were block inversion, cycle and bit flip (see Section C.2.1). The mutation rate and the selection of the mutation operator were controlled in a self-adaptive way (see Algorithm 2 in Section 2.1). Initially, the mutation rate was set to $r_{\text{mut}} = 1/m$. The operators were randomly initialized for each individual in the initial population. The parameters τ and p_{s} of the self-adaptation scheme were determined with a parameter tuning study that was part of our experiments. We tested the values $\tau \in \{1/(2\sqrt{2}), 1/\sqrt{2}, 2/\sqrt{2}\}$ and $p_{\text{s}} \in \{0.2, 0.5, 0.8\}$.

- **recombination:** Like the mutation operators, the recombination operators were randomly initialized and evolved in a self-adaptive fashion. For permutations, the potential operators were cycle crossover, order crossover, position-based crossover, and alternating position crossover (see Section C.1.2). For binary strings, we chose the operators uniform crossover, 1-point crossover, 2-point crossover, and AND-crossover (see Section C.2.2).
- **selection:** Two selection schemes were available, tournament selection or truncation selection. Truncation selection deterministically selects the best subset, while tournament selection is a probabilistic operation based on fitness ranks [82]. Tournament selection was configured to use a tournament probability of 0.8 and a tournament size of one-half of the population size n_{pop}. Both selection procedures were tested separately in the experiments.
- **population size:** The population size n_{pop} was also investigated in the parameter tuning study, during our experiments. We tested the values $n_{\text{pop}} \in \{5, 10, 20\}$.
- **number of offspring:** The number of offspring was set to $n_{\text{off}} = \lfloor 0.5\, n_{\text{pop}} \rfloor$ (brackets denote the floor function).

Model-based Optimization Settings The CEGO algorithm (see Algorithm 4) was configured as follows.

- **budget:** The objective function evaluation budget was set to 100.
- **replications:** Each run was repeated 20 times.
- **initial design:** The CEGO algorithm was configured to create an initial max-min design of 10 samples. The max-min design was created as follows. 100 different sets of 10 samples were created by uniform random sampling. Of these 100 sets, the one with the largest minimum pairwise Hamming distance was selected.
- **model:** Kriging was used as a surrogate model, with the kernel $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x'))$. Here, $\mathrm{d}(x, x')$ was the Hamming distance (that is, the number of elements that are not equal in x and x'). The nugget effect was not used in the model ($\eta = 0$). The Dividing Rectangles (DIRECT) algorithm [139] was chosen to optimize the model parameter θ during MLE. The employed DIRECT implementation is part of the NLopt library. We used the NLOPT_GN_DIRECT_L variant [138, 94]. Two hundred likelihood evaluations were allowed. A relative tolerance of $1\mathrm{e}{-6}$ was used to detect earlier convergence.
- **optimizer:** An EA was used to optimize the expected improvement criterion provided by the Kriging model. Largely, the EA had identical settings as the model-free EA. The budget was enlarged to 5000 evaluations of the surrogate model (in each CEGO iteration) since the model evaluations were assumed to be considerably cheaper than the objective function evaluations. Due to the larger number of evaluations, no archive was used.

Parameter tuning To allow for a fair comparison of CEGO and the model-free EA, a study on the effect of algorithm parameters was performed. We investigated the sensitivity to some of the algorithms' most important parameters. To that end, the following parameters of the EA were tested: the self-adaptation parameters ($\tau \in \{1/(2\sqrt{2}), 1/\sqrt{2}, 2/\sqrt{2}\}$ and $p_s \in \{0.2, 0.5, 0.8\}$), the choice of selection operator (tournament or truncation), and the population size ($n_{\text{pop}} \in \{5, 10, 20\}$). All combinations of these parameter settings were tested, for the model-free and the model-based optimization algorithm.

5.3.2 Results and Analysis

For the analysis, we followed the procedure described in Appendix A.

Visual Analysis Firstly, we performed an exploratory, visual analysis of the results, focusing on the best performing algorithm configurations. To that end, the algorithm configuration with the best median performance was selected for each test instance. The respective algorithm configurations are listed in Table 5.3 and their performances are compared in a box plot in Fig. 5.1. In terms of algorithm parameters, especially the model-free EA seemed to perform better with tournament selection, and it preferred smaller population sizes. Furthermore, the visual inspection seemed to confirm our expectations: In most cases, CEGO outperformed the model-free EA.

Table 5.3: *Parameter configurations of the EA and CEGO (with Hamming distance) with best median performance for each problem instance. Note that these choices were sometimes arbitrary since the attained objective function value was insensitive to the parameters.*

CEGO				EA				
p_s	τ	selection	n_{pop}	p_s	τ	selection	n_{pop}	instance
0.2	0.35	truncation	20	0.2	1.41	tournament	5	N30K2
0.2	1.41	tournament	10	0.5	0.35	tournament	5	N30K4
0.5	1.41	truncation	20	0.5	0.35	tournament	5	N30K8
0.8	1.41	tournament	20	0.2	0.35	tournament	10	MC30
0.5	0.71	truncation	10	0.5	0.35	tournament	10	MC40
0.2	0.35	tournament	5	0.2	0.71	tournament	5	bUNIH30
0.2	0.71	tournament	5	0.5	0.71	tournament	5	bUNILE30
0.2	0.71	truncation	5	0.2	0.35	tournament	5	bUNILC30
0.2	0.35	tournament	10	0.5	0.35	tournament	10	tho30
0.5	1.41	tournament	20	0.8	0.35	tournament	10	reC05
0.2	0.35	tournament	20	0.2	0.71	tournament	10	bayg29
0.5	1.41	tournament	20	0.5	0.35	tournament	5	atsp20
0.2	0.71	tournament	5	0.8	0.71	truncation	5	wt40b
0.5	1.41	truncation	5	0.2	0.35	tournament	5	pUNIH30
0.2	1.41	tournament	10	0.2	0.35	tournament	5	pUNIS30
0.2	0.71	truncation	5	0.2	0.71	tournament	5	pUNII30

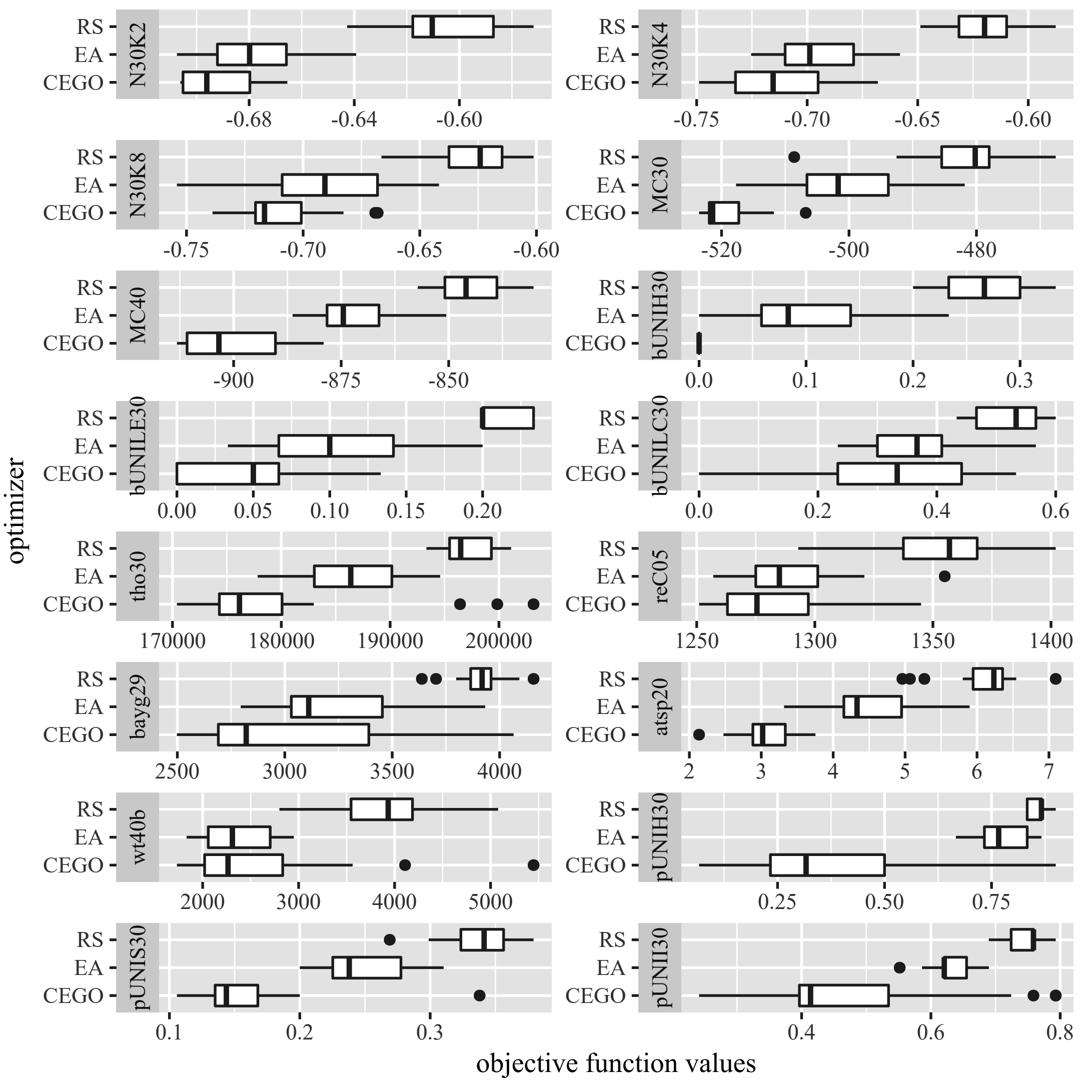


Figure 5.1: *Comparing the best configurations (according to median performance) of random search (RS), a model-free EA, and CEGO with Hamming distance. The x-axis shows the best-observed values of each run. Lower values are better. The dark gray boxes give the test problem IDs.*

Statistical Analysis Next, we trained a linear regression model to determine the influence of the algorithm parameters. Performance was measured by computing the gap to the best-observed objective function value for each problem instance.

Separate linear models were trained for each Algorithm (EA, CEGO) and each data type (permutation, binary). The models included main effects for the algorithm parameters as well as the problem instances. An additional model included effects for the algorithm and the data type, based on the complete data set.

The residuals of the resulting models did not seem to fit to the assumptions of homoscedasticity and normality. This was mostly remedied by using weighted least squares instead of ordinary least squares (see Appendix A). That is, the weighted least squares approach was used to train the linear regression model, where the weights were derived from the variances estimated from the replications of each algorithm run. Figure 5.2 compares QQ-plots of these linear regression models fitted with ordinary and weighted least squares. Clearly, weighted least squares was to be preferred.

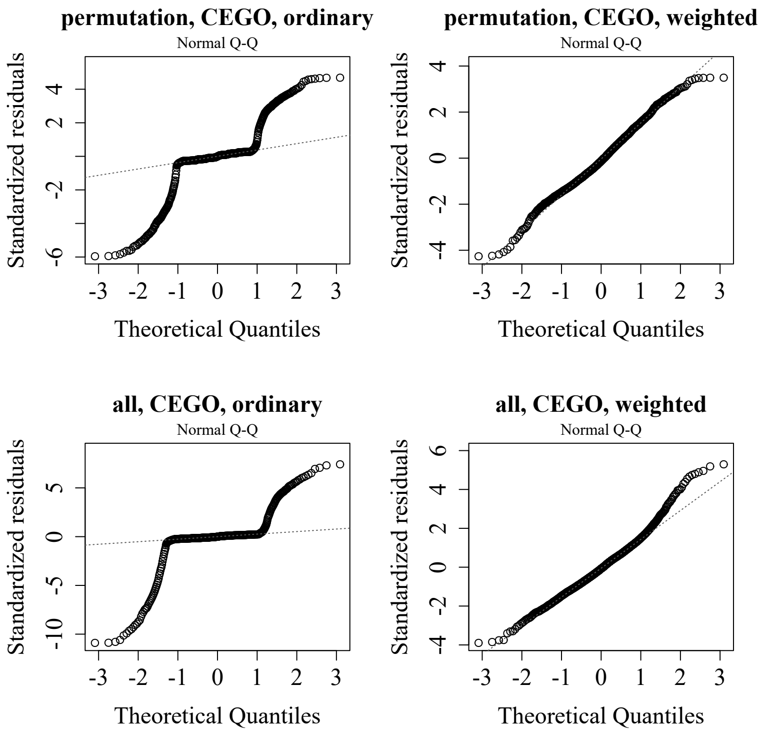


Figure 5.2: *QQ Plots for two of the linear regression models that are listed in Table 5.4, relating algorithm parameters to optimization performance. The upper row shows the model for the subset of permutation problems, the lower model concerns all experiments (permutation and binary). Each case is depicted once with ordinary least squares (left) and once with weighted least squares (right).*

The coefficients and corresponding p-values of the models (weighted least squares) are depicted in Table 5.4. The model-based CEGO runs performed better in general (lower intercepts). This was confirmed by the main effect of the *optimizer* contrast in the encompassing model (last row of Table 5.4). The positive value of that effect implies that changing from the model-based CEGO to the model-free EA decreased performance. Of all other parameters, the selection process (column: truncation) usually had the largest effect. Tournament selection generally performed better than truncation selection (positive contrast effect). To a lesser amount, the parameters p , τ and n_{pop} seemed to yield better performance when set to lower values (effects are positive, if significant). A preference for small population sizes n_{pop} was expected, due to the restricted number of objective function evaluations.

Table 5.4: *Coefficients of linear models, with p-values in brackets. The modeled variable is the optimization performance of two algorithms (EA, CEGO), tested on two problem classes (binary, permutation). The model shown in the last row includes the choice of optimizer as an effect. The columns* optimizer *and* selection *represent contrasts. That is, the column* optimizer *shows the effect of changing from CEGO to the model-free EA. The column* selection *shows the effect of changing from tournament to truncation. The effects of different problem instances are included in the model, but are not shown for the sake of brevity. The last column provides the adjusted coefficient of determination, a measure of model quality [154].*

	(intercept)	optimizer	p_s	τ	selection	n_pop	$r^2_\text{adj.}$
bin-EA	0.120 (0.00)	/	0.003 (0.05)	0.007 (0.00)	0.037 (0.00)	0.002 (0.00)	0.89
bin-CEGO	0.000 (0.82)	/	0.001 (0.41)	0.000 (0.60)	0.000 (0.80)	0.000 (0.83)	0.75
per-EA	3.228 (0.00)	/	0.000 (0.89)	0.005 (0.00)	0.048 (0.00)	0.002 (0.00)	0.96
per-CEGO	1.649 (0.00)	/	0.040 (0.00)	-0.003 (0.40)	0.068 (0.00)	0.000 (0.06)	0.84
all	2.671 (0.00)	0.066 (0.00)	0.006 (0.00)	0.003 (0.00)	0.020 (0.00)	0.001 (0.00)	0.92

In case of the binary runs with CEGO (second row of Table 5.4), the choice of parameters seemed to play little role in the determination of algorithm performance: All main effects received large p-values. This observation may be due to a poor model (the lowest of all $r^2_\text{adj.}$ values). Another reason may be that the binary problems mostly dealt with smaller search spaces than the permutation problems. Since the budgets were identical, this implies that the binary optimization problems might have been easier to solve. Hence, even a model-based EA with sub-optimal configuration could perform reasonably well, so that the parameters of the underlying EA had little impact on the performance. In fact, this may also explain the poor quality of the linear analysis model, since several binary runs converged to the optimum, and produced ties in the observed performances.

5.3.3 Case Study: Changing the Distance Function

SMBO seemed to be quite successful in most of the experiments. As an exception, no differences could be observed, e.g., for bUNILC30 (binary), reC05 (permutation) and wt40b (permutation). One potential explanation is that the Hamming distance is not the ideal distance measure for these problems. For example, bUNILC30 is based on the LCStr distance. At least for this case, using the LCStr distance in the model can be expected perform better. Therefore, the experiments with the model-based approaches were repeated, employing the four distance measures that were also employed in the UNI problem instances.

Figure 5.3 shows a subset of the results. Clearly, the choice of the distance measure was crucial. The figure shows that the LCStr distance performed best on bUNILC30. This issue is even more obvious for the swap distance on the pUNIS30 instance. Selecting the wrong distance caused CEGO to perform equal to or worse than the model-free EA. While the choice of distance is obvious for these simple unimodal problems, it may not be obvious for some black-box real-world application. Hence, we need a method that selects the right kernel.

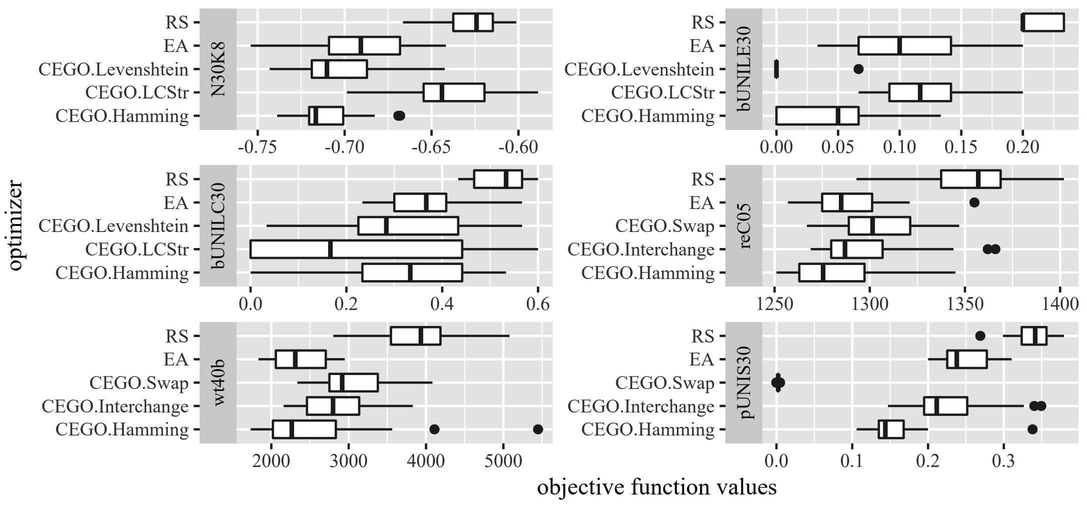


Figure 5.3: *Comparing the best configurations (according to median performance) of a model-free EA, random search (RS), and CEGO with different distance functions. The x-axis shows the best-observed values of each run. Lower values are better, the dark gray boxes contain test problem IDs.*

5.4 Conclusions

In this chapter, we discussed how SMBO could be extended to combinatorial or discrete search spaces.

Answer 5.1. Applicability. Can SMBO algorithms like EGO be applied to combinatorial or discrete search spaces?

Yes, if appropriate kernels are available. The Kriging variants we discuss only rely on correlations, and hence do not require the input data to be of a specific data type.

Answer 5.2. Comparison. How well do the respective algorithms compare to model-free optimization techniques?

In most of our experiments, the model-based CEGO algorithm clearly outperformed the model-free EA. However, a success may depend on the selection of a good kernel.

Our results raise two important follow-up questions. Firstly, we need an approach that is able to handle multiple kernels or distances. This necessity became obvious in Section 5.3.3. In the corresponding experiments, the selection of the kernel was crucial: A model based on the right kernel yielded the best performance, whereas other kernels sometimes produced results that are no better than random search. Hence, we need approaches to effectively select or combine kernels. This issue is discussed in the following Chapter 6.

Secondly, kernels for Kriging models should usually be PSD, as described in Section 2.3.1 and Section 4.4. Thus, we should be able to determine whether a kernel is definite or not. This is dealt with in Chapter 7. We discuss how Kriging can employ indefinite kernels in Chapter 8.

Chapter 6

How to Find the Right Kernel

The main modeling technique employed in this thesis, i.e., Kriging, relies on different measures of (dis-)similarity or kernels. Often, several notions of (dis-)similarity are conceivable. In practice, it may be unclear which measure works best. Clearly, the selection of the right kernel has a major impact on model quality. This was also demonstrated in the results presented in Section 5.3.3. Two questions result from this.

Question 6.1. Selection. How can the best kernel be chosen without prior knowledge?

Question 6.2. Combination. Can several kernels be combined?

This chapter tries to provide answers to these questions. In that sense, methods for the selection and combination of kernels or distances are described. We use the term *multi-kernel* method to indicate methods that either choose or combine several kernels. In contrast, some fixed kernel that has been chosen a priori is referred to as *single-kernel*.

These ideas are closely related to the concepts of multiple kernel learning [173] and metric learning [31]. Multiple kernel learning has already received much attention, especially in the context of support vector machines. These approaches consider multiple kernels at once. An alternative to combining existing kernels is to learn new ones from scratch. For instance, new kernels can be assembled from simple building blocks via genetic programming, as discussed by Gagné et al. [95] and Koch et al. [150]. In this chapter, we will focus on dealing with existing kernels.
Gönen and Alpaydı [173] provide an overview of multiple kernel learning algorithms. They distinguish algorithms by six different key properties:

The following sections are partially based on *"Distance Measures for Permutations in Combinatorial Efficient Global Optimization"* by Zaefferer et al. [271]. Some text elements have been adopted verbatim from the original contribution. The content of the publication has been extended significantly. Firstly, several additional methods for dealing with multiple kernels are discussed, including a more comprehensive explanation of the original ideas. Secondly, a more thorough experimental study now includes these methods as well as additional distance measures and test functions. Furthermore, the results are subject to a more in-depth analysis.

- Type of learning method: fixed rules, heuristic approaches, optimization approaches, Bayesian approaches, and boosting (ensemble) approaches.
- Functional form of the combination: linear combinations, non-linear combinations, and data-dependent combination.
- Target function: Similarity-based, structural risk, and Bayesian.
- Training method: How parameters of the combination function are determined, in a one-step or iterative fashion.
- Base models: Kriging, support vector machines, or other kernel-based learning methods.
- Computational complexity.

The methods in the focus of this chapter can be categorized with respect to type as heuristic approaches, optimization approaches and Bayesian approaches. Instead of boosting, stacked generalization (stacking) is proposed as an additional ensemble technique for combining learners based on different kernels. Furthermore, we discuss an approach based on the superposition of individual Kriging models.
Regarding the functional form, we stick to linear combinations. One exception is the superposition approach, which could also be classified as data-dependent. The selection of a single kernel can be viewed as an extreme case of a linear combination, where all but one weight are zero. With respect to target functions, we focus on similarity-based and Bayesian approaches. This chapter mostly deals with Kriging models that are trained in a one-step procedure, but it is straightforward to apply most of the discussed approaches to arbitrary other models.
Our contributions are:

- We consider selection as well as combination methods.
- We focus on kernels with a distance interpretation.
- We suggest different variants to combine or select kernels based on correlation, likelihood, linearity, ensemble methods, and superposition.
- We account for our main goal, which is optimization.
- To generate kernel combinations that are interpretable and allow for an efficient implementation, we propose to use linear models with variable selection and non-negative coefficients.
- Based on numerical experiments, we investigate how multi-kernel methods perform in the context of combinatorial SMBO.

With respect to the last point, this chapter provides an empirical study for permutation problems. This choice is made due to the large number of problems that involve permutation representations [5]. One frequent example are production processes that have to be scheduled such that processing times or other costs are minimal. In addition to the large number of applications, permutations also give rise to numerous distance measures, which can be employed in a kernel, see Appendix B. Hence, this presents an excellent test case for the questions that have been raised.

6.1 Selecting Kernels

The methods in this section intend to select a kernel from a set of kernels. The selected kernel has to be distinguished from other, sub-optimal kernels that are less suited for the given problem. Our methods are data-driven, i.e., based on a set of given training data. We do not consider cases where prior knowledge is used to select a kernel.

A data-dependent kernel selection process cannot be perfectly accurate. Its performance depends on the training data as well as the accuracy of the selection procedure. A wrong decision may occur, especially while data is still very sparse at the beginning of an SMBO run. Therefore, a data-dependent kernel selection process may be worse than a model based on the single best kernel, for a specific data set. However, this single best kernel is usually not known in advance.

Even more, a single-best kernel may be outperformed by kernel selection if the behavior of the model becomes more dynamic. That means, different kernels may work best in different phases of an SMBO run. For example, a specific kernel may work best for sparse data (start of the run), while another kernel may be better for increasingly dense data sets (near the end of the run).

In the following, we introduce different selection methods based on correlation, maximum likelihood, and cross-validated errors. Before we come to these more complex kernel selection methods, we require a simple baseline against which sophisticated methods can be compared.

6.1.1 Random Selection

We suggest uniform random selection as a baseline for our experiments. Random selection implies that whenever a model is trained, the employed kernel is chosen randomly from the set of available kernels. This baseline may even perform satisfactory in some cases, e.g., when the number of available kernels is rather small. Since the optimization algorithms we employ are iterative procedures, it may be sufficient to select the correct kernel occasionally. We refer to this simple procedure as `selectRandom`.

6.1.2 Selection with Correlation Measures

A straightforward idea to deal with multiple kernels is to select one kernel based on a measure of correlation between kernel values and observed values. For the sake of simplicity, we mostly employ Pearson correlation. Other measures, such as the rank-based Spearman correlation [79], could be used to replace Pearson correlation in the following approaches.
The methods discussed here are similar to the least squares fitting of variogram models [69]. The main difference is that we use correlation instead of the sum of squared deviations. In addition, variogram models usually consider the fitting of kernel parameters rather than selecting from a discrete set of kernels.

Fitness Distance Correlation When the selection of kernels can be simplified to a selection of a distance measure, a very intuitive and conceptually simple idea may be used: Fitness Distance Correlation (FDC) [141]. FDC evaluates the linear Pearson correlation between the distances to the global optimum and the respective fitness values. That means, given n candidate solutions with objective values $\mathbf{y} = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}$ and distances to the optimum $\mathbf{d} = \begin{bmatrix} d_1 & \dots & d_n \end{bmatrix}$, with respective means and standard deviations ($\bar{y}$, $\bar{d}$, σ_y, σ_d) the FDC is

$$\text{FDC} = \frac{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})(d_i - \bar{d})}{\sigma_y \sigma_d}$$

This definition varies slightly in the numerator, compared to the definition by Jones and Forrest [141]. It uses the sample covariance with $1/(n-1)$ instead of the population covariance with $1/n$.
When the global optimum is unknown, the local FDC can be computed instead. It uses the best-observed solution instead of the global optimum [144]. Since we deal with black-box problems and do not know the global optimum, we use the local FDC.
The FDC has previously been used to analyze fitness landscapes. In the context of minimization, a problem is considered easy if small distances to the optimum coincide with low fitness values, i.e., positive FDC. This is because positive FDC implies smoothness of the fitness landscape, at least in the direction of the optimum. In other words, candidates close to the optimum do not have poor fitness values. It is simple to extend this idea to the selection of a distance measure for a model. The distance measure with the largest FDC value is the most promising choice, as it perceives a smoother landscape.
Previous work on fitness landscape analysis advises caution since FDC was shown to be potentially misleading [7]. Still, the FDC is comparatively simple to compute, independent of the chosen surrogate model. Whether the potentially misleading information actually translates into decreased performance for our use case needs to be investigated.
Overall, the FDC selection procedure works as follows. A data set of $n \in \mathbb{N}$ pairs $\{x^{(i)}, y_i\}$ with $i = 1, ..., n$ is given, as well as a set of $r \in \mathbb{N}$ distance measures

$\mathrm{d}_j(x,x')$ with j=$1,...,r$. First, the candidate solution x_{opt} that yields the minimal objective value $y_{\text{opt}} = \min(\mathbf{y})$ is chosen from the data set. Next, we compute the distances $\mathbf{d} = \left[\mathrm{d}_j(x^{(1)},x_{\text{opt}}) \;\; \ldots \;\; \mathrm{d}_j(x^{(n)},x_{\text{opt}})\right]$ and the corresponding correlation $\text{FDC}_j = \text{corr}(\mathbf{d},\mathbf{y})$. Finally, the distance measure $\mathrm{d}_j(x,x')$ that yielded the maximal FDC_j is chosen.
Clearly, the situation becomes slightly more complex when the distance measures have parameters. Then, the selection process can be redefined as an optimization problem with respect to the parameters: for each distance measure, the parameters that maximize FDC are determined. Then, the distance that yields the maximal FDC is selected. We refer to the FDC-based selection procedure as `selectFDC`.

Fitness-Difference Distance Correlation FDC considers the distances to a reference solution, usually the best-observed solution. Since the goal is to use the resulting model for optimization, this may be reasonable. We are interested in using our model to find the optimum, so the distances in promising directions (i.e., towards optimal solutions) are of great interest. In other words, the FDC measure reflects the goal of the optimization process.
However, a model like Kriging considers all pairwise distances during training, regardless of any reference solutions. Thus, parts of the information used by the model are not considered by the FDC criterion.
To alleviate this, all pairwise differences should be considered during the selection procedure. This idea can be used to develop an alternative to FDC-based selection, which we refer to as Fitness-Difference Distance Correlation (FDDC). Here, we calculate *all* pairwise distances and all pairwise differences in fitness. Then, the Pearson correlation of these distances and the fitness-differences is calculated. The resulting FDDC values are used just like the FDC values.
In summary, we start by computing all differences between the observations $\mathbf{y}$, i.e., $\Delta y_{il} = |y_i - y_l|$, for all $i,l \in \{1,...,n\}$. Next, for each distance j, $d_{il} = \mathrm{d}_j(x^{(i)},x^{(l)})$ and $\text{FDDC}_j = \text{corr}(\mathbf{d},\boldsymbol{\Delta}\mathbf{y})$ are computed. Finally, the distance measure $\mathrm{d}_j(x,x')$ that yields the maximal FDDC_j is chosen. We refer to this selection procedure as `selectFDDC`.

6.1.3 Selection with Maximum Likelihood Estimation

While the structure of common Kriging kernels for real-valued samples, e.g., $\mathrm{k}(x,x') = \exp(-\theta|x-x'|^p)$, is indeed fixed, the respective kernels are not completely inflexible. Rather, the exact shape can be changed via the parameters, e.g., θ or p. The configuration of parameters such as p may be included in the MLE process during model training.
We can exploit this idea when we are faced with selecting different kernels. A choice between different kernels can be understood as a categorical parameter of an encompassing kernel. For the case where we use an exponential kernel $\mathrm{k}(x,x')$ with a number

of potential distance measures $\mathrm{d}_i(x, x')$, this idea can be presented as follows,

$$\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}_\gamma(x, x')).$$

Here, the parameter γ decides which of the available measures is selected. This new categorical parameter can be optimized alongside the other kernel parameters during MLE. This could be done with an appropriate mixed-integer optimization algorithm such as the MIES [84, 163]. Alternatively, for the sake of simplicity, the usual model training procedure can be performed for each kernel separately. Afterwards, the kernel that yielded the largest likelihood value is chosen. In terms of the survey of Gönen and Alpaydı [173], this MLE-based approach could be categorized as a Bayesian or optimization approach to multiple kernel learning. We refer to this selection procedure as `selectMLE`.

In comparison to other selection approaches, the MLE-based selection has the advantage that it is natural to Kriging. It is easily integrated into standard frameworks. Further, it is also straightforward to optimize additional parameters of the distance measure or kernel function in the MLE procedure. A drawback of this approach is the fact that it cannot be applied to arbitrary models, since it relies on some form of stochastic model where likelihoods can be computed.

6.1.4 Selection with Cross-Validation

Different kernels may as well be chosen by selecting the one that yields a minimal, cross-validated error. Cross-validation (CV) is a standard procedure for model evaluation and selection in the field of machine learning [132]. It is used to approximate how well a model performs on unseen data.

We use k-fold CV. Here, the training data set is randomly split into k non-overlapping subsets of equal size. The model is built k times, each time excluding a different subset, using the remaining data for training. The model's error is estimated with the excluded data. CV has the advantage of being applicable to any model type. It has the drawback of potentially increasing the computational effort for model training: Models have to be trained for every tested kernel and all k subsets of the data. We refer to this selection procedure as `selectCV`.

6.2 Combining Kernels

Selecting a kernel may be a good solution in some cases. In other cases, there may be no single, best performing kernel. Rather, several kernels could contribute complementary information with regard to the similarity of solutions. In such a case, a combination of kernels is more promising than selection.

A trivial example are mixed-integer problems. Here, different distance measures can be used for the continuous and the categorical parameters, e.g., Euclidean and Hamming distance. Clearly, it makes little sense to choose between them. Each distance contains information about a different set of variables. In previous works by Hutter [126] and Li et al. [164], the respective distance measures were hence combined. Li et al. [164] used a fixed rule for combination. They combined the individual measures by computing the square root of their sums. Hutter [126] proposed to use a weighted sum of the individual distances, with weights determined by MLE.

In this work, we also deal with weighted sums of individual distance measures, although not restricted to learning the weights via MLE. Weighted sums have the advantage of being interpretable and relatively simple to compute. In our use case, they also avoid potential problems with respect to definiteness since products of PSD kernels are also PSD[1]. Definiteness may become a problem if more complex, nonlinear combinations are considered. Weighted sums of kernels are among the more frequently used methods in multiple kernel learning [173].

In the following, we propose four methods for combining distance measures or kernels. Three of them can be linked to the selection methods earlier described in Section 6.1.

- The linear regression approach described in Section 6.2.1 is connected to the correlation measures from Section 6.1.2.
- The MLE-based combination in Section 6.2.2 corresponds to the MLE-based selection from Section 6.1.3.
- Finally, the stacked generalization approach in Section 6.2.3 is analogous to the CV-based selection discussed in Section 6.1.4.

In addition, we consider a combination based on the superposition of Kriging models in Section 6.2.4.

6.2.1 Weighted Sum via Linear Regression

The methods for kernel selection in Section 6.1.2 exploit a potential linear relationship between fitness-differences and distances to *select* a distance measure. We propose to use the same information to *combine* different distance measures.

Similarly to selectFDDC, the idea is to first compute all pairwise differences in the objective value $\Delta y_{il} = |y_i - y_l|$ as well as all pairwise distances $d_{il,j} = \mathrm{d}_j(x^{(i)}, x^{(l)})$ (for each j). Then, a linear regression model can be built to learn the relationship between Δy_{il} and $d_{il,j}$:

$$\Delta y_{il} = \sum_j \beta_j d_{il,j} + \epsilon_{il} \tag{6.1}$$

with $\epsilon_{il} \sim N(0, \sigma^2)$. Correspondingly, the combined distance measure is then

$$\mathrm{d}_*(x, x') = \sum_j \beta_j \mathrm{d}_j(x, x') \tag{6.2}$$

[1] Since we usually consider exponential kernels in this thesis, the weighted sum of the CNSD distances implies a product of the encompassing PSD kernels, e.g., $\exp\left(-\theta_1 d_1(x,x') - \theta_2 d_2(x,x')\right) = \exp\left(-\theta_1 d_1(x,x')\right) \exp\left(-\theta_2 d_2(x,x')\right)$.

Intuitively, the combined distance should be zero if all individual distances are zero. Hence, the linear model in Eq. (6.1) has no intercept term. Furthermore, the coefficients β_j can be understood to encode the relevance of each single distance measure. Large β_j imply that the j-th distance has a strong linear relationship with the fitness-differences. Otherwise, distances that have no observable effect should receive coefficients close to zero.
Clearly, it is desirable that the coefficients β_j are non-negative. This has three reasons. Firstly, negative β_j would deteriorate the interpretability of the model. Secondly, the combined distance measure could otherwise yield negative distance values. Thirdly, the combined measure could potentially become indefinite. Therefore, the coefficients in Eq. (6.1) are constrained with $\beta_j \geq 0$.
This linear modeling approach can be used regardless of the chosen surrogate model (e.g., Kriging) that employs the combined distance measure. It is also fairly simple to compute. One decisive drawback in comparison to the selection approaches is the additional computational effort: The resulting model predictor requires the evaluation of all distance measures (e.g., for prediction), rather than just the one that was selected. To partially alleviate this last issue, we suggest enabling the model to exclude distances that do not contribute relevant information. That means, some coefficients β_j should be set to exactly zero. The corresponding distance measures can be disregarded for prediction. To achieve this, we can use the *least absolute shrinkage and selection operator* (LASSO), which comprehends a variable selection procedure [247]. We refer to our approach of combining kernels with linear regression as `combineLinear`.
This process could be extended with non-linear regression models. However, great care would have to be taken, to avoid issues such as indefiniteness of the resulting combination.
A very similar, classical approach from the field of stochastic process modeling is known as Minimum Norm Quadratic (MINQ) estimation [69]. MINQ estimation attempts to learn linear combinations of covariance matrices via a least-squares approach.

6.2.2 Weighted Sum via MLE

A weighted sum of distance functions as described in Eq. (6.2) can also be generated with MLE. To that end, the respective distance measures have to be embedded into a combined kernel, e.g.,

$$\mathrm{k}_*(x, x') = \exp(-\sum_j \beta_j \mathrm{d}_j(x, x')), \tag{6.3}$$

which is subsequently subject to MLE. That means, the weights β_j are kernel parameters that are optimized numerically via MLE (cf. Section 2.3.1). In terms of the categorization of Gönen and Alpaydı [173], this combination approach can be denoted as Bayesian or optimization with a Bayesian target function.

The structure of the combined kernel in Eq. (6.3) could also be subject to optimization. However, choosing some standard structure like the exponential kernel is recommendable, to avoid encumbering the MLE procedure with too many parameters. This also extends to the number of different distance measures $\mathrm{d}_j(x, x')$: Weeding out unlikely candidates where possible (e.g., via some kind of expert knowledge) reduces the burden of the MLE procedure and enables to find better weights. We refer to the approach of combining kernels by MLE as `combineMLE`.

6.2.3 Weighted Sum via Stacking

Similar to the CV-based selection described in Section 6.1.4, a k-fold segmentation of the training data can also be used to learn how to correct the corresponding generalization error of different candidate models. Stacked generalization or stacking [261, 22] uses such an approach, to combine different models (or learners) in an ensemble. Stacking was recently suggested for combining diverse surrogate models for optimization [22].

For two reasons, stacking is the most expensive of our combination methods. Firstly, it combines models rather than just kernels or distance measures. Secondly, like k-fold CV, stacking requires to fit each model k times.

Loosely following the descriptions by Bartz-Beielstein [22], our stacking implementation is depicted in Fig. 6.1. Firstly, the training data is randomly split into k subsets of equal size. For each subset $i = 1, \ldots, k$ and each distance measure $\mathrm{d}_j(x, x')$ (or else, kernel), a model is built excluding the corresponding subset of the training data (hold-out data). These models are denoted as level-0 models. They are used to predict the hold-out data. The resulting predictions are the input of an encompassing model, denoted as level-1 model. The level-1 model attempts to correct the level-0 predictions. When predicting new data, the stacked ensemble first predicts with all level-0 models, which have been re-trained with the complete training data. Then, we use the individual predictions as inputs of the level-1 model, which produces the ensemble prediction.

In this study, the level-0 models are Kriging models with different kernels. For the level-1 model, we chose a linear model with non-negative coefficients and no intercept, similarly to the approach described in Section 6.2.1. Since complete models are combined instead of kernels, negative distances or indefiniteness are not an issue (unless the underlying kernels themselves are indefinite). Still, non-negative coefficients are recommendable, due to better interpretability of the resulting ensemble. In addition, linear level-1 models with non-negative coefficients have been reported to work well in stacking ensembles [45, 159].

Furthermore, we use LASSO for variable selection, to exclude redundant or irrelevant models. Due to the cost of combining many individual models via stacking, the LASSO variable selection is especially important to reduce the computational effort of the ensemble predictor.

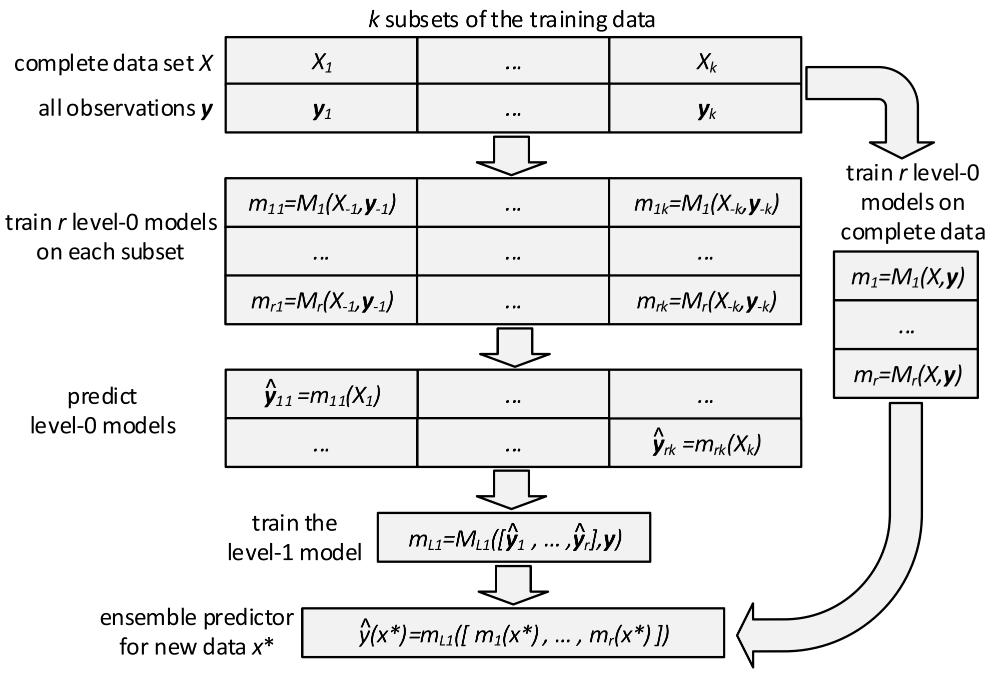

k subsets of the training data
complete data set X
all observations y
X1
...
Xk
y1
yk
train r level-0 models on each subset
m11=M1(X-1,y-1)
m1k=M1(X-k,y-k)
mr1=Mr(X-1,y-1)
mrk=Mr(X-k,y-k)
train r level-0 models on complete data
m1=M1(X,y)
mr=Mr(X,y)
predict level-0 models
ŷ11 =m11(X1)
ŷrk =mrk(Xk)
train the level-1 model
mL1=ML1([ŷ1 , ... ,ŷr],y)
ensemble predictor for new data x*
ŷ(x*)=mL1([m1(x*), ... , mr(x*)])

Figure 6.1: *Training and prediction of a stacking ensemble with k data subsets, and r different level-1 models (e.g., using r different kernels). $M_i(x, y)$ indicates a training procedure with input data x and output y for an individual model, and $m_i(x)$ is the predictor of a trained instance of the model. Data indices are used somewhat differently than in the rest of this thesis: X_{-1} and $\mathbf{y}_{-1}$ indicates the complete data, excluding the subsets X_1 and $\mathbf{y}_1$.*

For this specific level-1 model, the combined predictor of the ensemble is

$$\hat{y}_{\text{ensemble}}(x^*) = \sum_{j=1}^{r} \beta_j \hat{y}_j(x^*),$$

where $\hat{y}_j(x)$ is the predictor from Eq. (2.4) based on the distance function (or kernel) j, and β_j are the non-negative coefficients of a linear model with LASSO.
Stacking combines models rather than kernels. This causes a problem that is not present in the approaches that combine kernels. When the ensemble model is employed in the CEGO framework, we require an uncertainty estimate to compute the expected improvement criterion, see Section 2.4. If the level-1 model is some complex non-linear learner, determining the ensemble uncertainty may be quite complex or even infeasible. Since we use a linear, non-negative combination, we can use this structure to combine the individual uncertainties in a straightforward way. First, we assume that the predictive distributions from all level-0 models are not correlated to each other. Whether this is always true is doubtful, but the assumption is nevertheless useful as the correlations between level-0 models are unknown to us. We exploit that the variance of the sum of uncorrelated random variables z_1 and z_2 is $\text{var}(z_1 + z_2) = \text{var}(z_1) + \text{var}(z_2)$,

and var$(\beta z) = \beta^2$var(z). Thus, the uncertainty of the ensemble is estimated to be

$$\hat{s}^2_{\text{ensemble}}(x^*) = \sum_{j=1}^{r} \beta_j^2 \hat{s}_j^2(x^*).$$

We refer to the approach of combining kernels via stacking as `combineStack`. Other ensemble methods could be used, such as bagging or boosting [132]. We decided to use stacking because i) it is easy to interpret the linear level-1 model in terms of each kernel's contribution to the ensemble, ii) the uncertainty estimate is straightforward to compute, and iii) a combination via stacking is analogous to selection via cross-validation.

6.2.4 Superposition of Kriging Models

Another approach that combines models rather than kernels is the superposition of Kriging models. This approach has previously been used to combine models built from data subsets, to reduce the computational complexity of the model building procedure [250, 258]. The resulting superposition is a weighted sum of the individual models, but with weights that depend on the location in the search space. The weights are based on the uncertainty estimates of each individual model. That means, a model that is very certain about a specific sample has more impact on the prediction than a model that is rather uncertain. Following the distinctions in the survey by Gönen and Alpaydı [173], this could be termed a linear, data-dependent combination. The predictor and uncertainty estimates of the resulting combination can be written as [250, 258]:

$$\hat{y}_{\text{super}}(x^*) = \sum_{j=1}^{r} \beta_j(x^*)\hat{y}_j(x^*),$$

$$\hat{s}^2_{\text{super}}(x^*) = \sum_{j=1}^{r} \beta_j(x^*)^2 \hat{s}_j^2(x^*),$$

with

$$\beta_j(x^*) = \hat{s}_j^{-2}(x^*) / \sum_{i=1}^{r} \hat{s}_i^{-2}(x^*).$$

The weights $\beta_j(x^*)$ are non-negative, similarly to the stacking ensemble proposed in Section 6.2.3.

In contrast to stacking, the weights are data-dependent. In fact, the superposition could react to non-stationary properties of the data. This also has a negative repercussion. Since the weights may change from location to location, the exclusion of a certain model is not straightforward. In effect, this means that the predictor of the superposition approach is usually more expensive than the LASSO-based stacking approach. On the plus side, fitting the superposition model is cheaper because a model is trained only once for each kernel. In case of stacking, a model is trained k times for each kernel.

Of course, using superposition of Kriging models assumes that the uncertainty estimates of each single-kernel model are at least roughly correct. Since the uncertainties are used to determine the weights, an error in the uncertainties leads to an error in the linear combination. We refer to the approach of combining kernels via superposition as `combineSuper`.

6.3 Experimental Validation

The various multi-kernel methods all have their own advantages and drawbacks. It is of interest how their different properties translate into optimization performance. To that end, this section investigates the performance of the methods in a set of experiments with artificial benchmark functions. The goal is to determine whether the selection or combination approaches actually work, how they compare to each other, and how they interact with different test problems.

6.3.1 Experimental Setup

Test Problems To allow for a detailed, yet focused study of the proposed multi-kernel methods, the experiments focused on permutation problems. Permutations are a promising choice since numerous distance measures are available. We mostly used the same test problem classes as described in Section 5.3.1: TSP, ATSP, FSP, WTP, QAP and UNI. However, a larger number of problem instances was selected for each problem class. These instances are listed in Table 6.1. The listed UNI instances were based on Hamming distance (pUNIH*), swap distance (pUNIS*) and interchange (pUNII*).

Model-Free Optimization Settings As a baseline experiment, we tested a model-free EA as described in Section 2.1. Mostly, the same configuration as in Section 5.3.1 was used, with the following exceptions.

- **budget:** The budget of fitness function evaluations was set to 200.
- **self-adaptation:** Based on the statistical analysis in Section 5.3.2, the learning parameter was set to $\tau = 1/(2\sqrt{2})$. The probability of changing to a different operator was specified with $p_\text{s} = 0.2$.
- **selection:** Also following from Section 5.3.2, tournament selection was chosen, with a tournament probability of 0.8 and a tournament size of $n_\text{pop}/2$.
- **population size:** Accordingly, the population size was set to 5.

Model-based Optimization Settings The various multi-kernel methods were tested by integrating them into the CEGO algorithm (see Algorithm 4), which used the following configuration. Again, we only list those configurations that differ from Section 5.3.1.

Table 6.1: *Overview of the permutation problem instances used in the multi-kernel experiments.*

Problem	Instance Name	m
TSP	bayg29	29
TSP	fri26	26
TSP	gr24	24
ATSP	atsp10	10
ATSP	atsp20	20
ATSP	atsp30	30
ATSP	atsp40	40
QAP	tho30	30
QAP	kra33	32
QAP	nug30	30
QAP	nug12	12
WTP	wt40a	40
WTP	wt40b	40

Problem	Instance Name	Dim
WTP	wt40c	40
WTP	wt40d	40
FSP	reC05	20
FSP	reC13	20
FSP	reC19	30
FSP	reC31	50
UNI	pUNIH20	20
UNI	pUNIH30	30
UNI	pUNIS20	20
UNI	pUNIS30	30
UNI	pUNII20	20
UNI	pUNII30	30

- **budget:** The objective function evaluation budget was set to 200.
- **initial design:** 10 samples were created by uniform random sampling.
- **model:** The distance function $\mathrm{d}(x, x')$ used in the Kriging model was either chosen from a set of distances, or a combination of those. Each distance was also tested separately, without any multi-kernel approach. The employed distance measures are listed in Appendix B.3. The employed multi-kernel methods were selectFDC, selectFDDC, selectFDC, selectMLE, selectCV, combineLinear, combineMLE, combineStack, and combineSuper. The nugget effect (determined by MLE) was used in the model, in combination with re-interpolation (cf. Section 2.3.4). A relative tolerance of $1\mathrm{e}{-16}$ was used to detect earlier convergence of the likelihood maximization.
- **optimizer:** The model-free EA was used to optimize the expected improvement criterion computed by the Kriging model. Following the results described in Section 5.3.2, the population size was set to $n_{\text{pop}} = 10$, and the self-adaptation probability was set to $p_{\text{s}} = 0.2$.

6.3.2 Results and Discussion

Visual Analysis To interpret the results, we started with a visual analysis. As an example, Fig. 6.2 shows box plots of the best-observed values in case of the ATSP instances. The remaining box plots can be found in the appendix, in Figs. D.1 to D.5. The visual analysis indicates that the multi-kernel methods worked quite well: they usually managed to be among the best performing algorithm configurations.

The best performing single distances seemed to be problem specific. The adjacency distance worked well for the TSP instances, while the R distance worked well for the ATSP instances. This makes sense since the R distance is the asymmetric variant of the adjacency distance, and is hence better suited for the asymmetric TSP. Other distances that worked well are, e.g., the Position distance for WTP and FSP or the Hamming distance for the QAP. While not always optimal, the Hamming and Position distances performed robustly, usually outperforming the model-free EA.

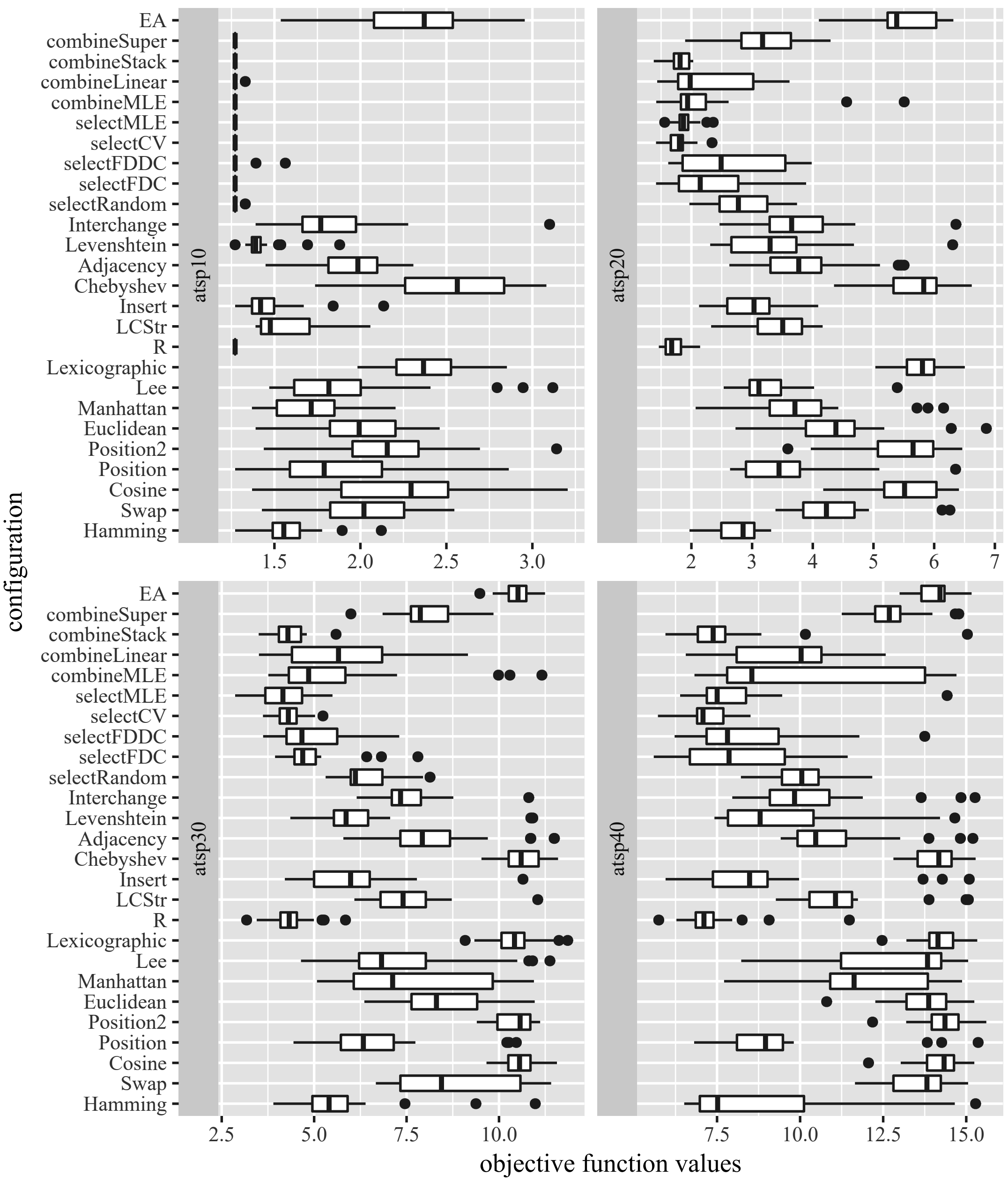


Figure 6.2: *Results of different optimization runs with different kernels for permutations (or different methods to select/combine kernels) on ATSP instances. Smaller values are better.*

Statistical Analysis To confirm these observations and to receive a more complete view of the results, we applied the statistical analysis tools described in Appendix A. The visual analysis already indicated that the data was not homoscedastic. In some cases (e.g., problem instance atsp10), variances were zero for some algorithms: The respective algorithms always converged to the optimum. Some of the suboptimal algorithms had rather large variances.

Further, the box plots indicated several cases where the distributions were quite skewed. These observations were confirmed by the model analysis plots, which are shown in Fig. 6.3. These plots concern a linear model that includes main effects for chosen kernels, multi-kernel approaches, and problem instances. Transformations and weighted least squares did not improve this situation sufficiently. Therefore, we used non-parametric tests for the further analysis.

The initial tests for the presence of significant differences (Kruskal, Friedman) yielded p-values $< 10^{-16}$. Hence, we rejected the null hypothesis (no significant differences). There seemed to be evidence for differences between the algorithms and post hoc tests were warranted. The resulting ranks based on pairwise multiple comparison tests are shown in Table 6.2 (cf. Appendix A).

Largely, the statistical tests confirmed the visual analysis: Multi-kernel methods generally performed equal to or better than single-kernel algorithms, but there was no individual problem instance where the multi-kernel methods performed significantly better than the best single kernel. The best single kernel varied, depending on the problem, but Hamming and Position distance received the best ranks overall. The model-free EA was consistently outperformed by most model-based methods, especially by the multi-kernel approaches.

The exact choice of multi-kernel method seemed to be less important. No decisive

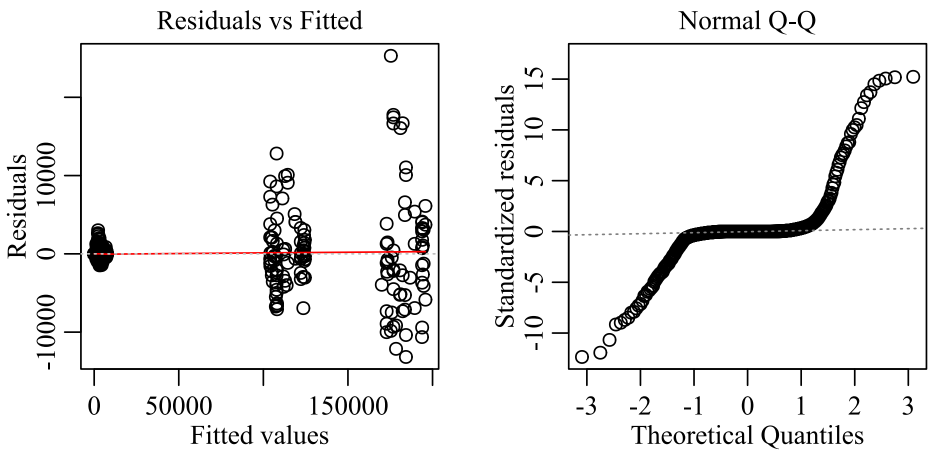


Figure 6.3: *Residual plot (left) and QQ-plot (right) for a linear regression model of the optimization performance dependent on two variables: 1) chosen kernels (or multi-kernel approaches) and 2) problem instances.*

differences were found between most of the multi-kernel methods. Even the simplest multi-kernel procedure, selectFDC, produced competitive results. Overall, selection based on CV received the best rank. It never performed worse than its competitors. Only the random selection baseline and the superposition approach performed poorly. While superposition worked well enough in some individual cases (atsp10, reC05 and most WTP instances), it performed worse than many of the competitors in all other cases. This may be explained by the way this method combines models: It relies on using the uncertainty estimates as weights. If these estimates are wrong (e.g., because an individual kernel or distance does not represent the problem behavior well), the quality of the combined model deteriorates.

Frequencies and Weights In addition to optimization performance, it is of interest to investigate how the multi-kernel methods operated during each optimization run:

Table 6.2: *Ranks for the experimental comparison of model-based optimization with multiple kernels, based on the non-parametric test procedures described in Appendix A. Rows are test problem instances, columns are different algorithm configurations, where the name either identifies the used kernel, or a method that was used to select / combine multiple kernels. The row "All" shows the combined test for all test problem instances. The rightmost column (EA) is the model-free baseline algorithm. The best algorithms / kernels are denoted with green background color and bold font.*

	Hamming	Swap	Cosine	Position	Position2	Euclidean	Manhattan	Lee	Lexicographic	R	LCStr	Insert	Chebyshev	Adjacency	Levenshtein	Interchange	selectRandom	selectFDC	selectFDDC	selectCV	selectMLE	combineMLE	combineLinear	combineStack	combineSuper	EA
All	5	9	11	4	9	9	6	6	12	9	11	7	12	10	6	8	4	2	3	**1**	2	3	4	2	6	12
atsp10	2	4	4	3	4	4	3	3	5	**1**	2	2	5	3	2	3	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5
atsp20	2	4	5	3	5	4	4	3	5	**1**	3	3	5	4	3	4	2	2	2	**1**	**1**	**1**	2	**1**	3	5
atsp30	2	3	4	2	4	3	3	3	4	**1**	3	2	4	3	2	3	2	**1**	**1**	**1**	**1**	2	2	**1**	3	4
atsp40	**1**	3	3	2	4	3	3	3	3	**1**	2	**1**	3	2	2	2	2	**1**	**1**	**1**	**1**	2	2	**1**	3	3
wt40a	2	4	4	**1**	2	3	2	3	4	4	4	**1**	4	4	**1**	3	2	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	3
wt40b	2	3	3	**1**	2	3	2	2	4	3	3	**1**	3	3	**1**	2	2	2	**1**	**1**	**1**	**1**	**1**	2	**1**	3
wt40c	2	3	3	**1**	**1**	3	2	2	3	3	3	**1**	3	3	**1**	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	3
wt40d	**1**	3	3	**1**	**1**	3	2	2	3	3	3	**1**	3	3	**1**	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	3
pUNIH20	**1**	4	5	**1**	5	4	**1**	**1**	5	5	5	4	5	5	2	3	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	2	5
pUNIS20	3	**1**	3	4	6	**1**	**1**	2	7	7	7	6	5	7	4	5	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	3	6
pUNII20	**1**	3	3	2	4	3	2	2	4	4	4	4	4	4	3	2	2	**1**	**1**	**1**	**1**	**1**	2	**1**	2	4
pUNIH30	**1**	5	6	2	6	5	2	3	6	6	6	5	6	6	3	4	2	**1**	2	**1**	**1**	**1**	3	**1**	3	6
pUNIS30	4	**1**	3	5	6	3	**1**	4	6	6	7	6	5	6	5	5	**1**	**1**	2	**1**	**1**	**1**	2	**1**	3	6
pUNII30	**1**	3	3	2	3	2	2	2	3	3	3	3	3	3	2	**1**	2	**1**	**1**	**1**	**1**	2	2	**1**	2	3
tho30	**1**	2	2	**1**	3	**1**	**1**	**1**	3	3	3	3	3	3	2	2	2	**1**	**1**	**1**	2	2	2	**1**	2	3
kra32	**1**	2	3	**1**	3	2	**1**	**1**	4	4	4	3	4	4	2	2	2	**1**	**1**	**1**	**1**	**1**	2	**1**	2	3
nug30	**1**	3	4	**1**	3	3	2	2	3	4	3	2	4	3	**1**	2	2	2	**1**	**1**	2	2	2	**1**	2	3
nug12	**1**	2	2	**1**	2	2	**1**	**1**	3	3	3	2	3	2	**1**	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	2	3
reC05	2	3	3	**1**	**1**	2	2	2	3	3	3	**1**	3	3	**1**	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	3
reC13	**1**	3	3	**1**	2	3	2	2	3	4	3	2	4	4	**1**	2	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	2	4
reC19	2	3	4	**1**	2	3	2	2	4	4	4	**1**	4	4	2	3	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	3	4
reC31	2	4	3	**1**	2	3	3	3	4	4	4	2	4	4	2	3	2	**1**	**1**	**1**	**1**	2	2	**1**	2	3
bayg29	2	4	5	3	4	4	4	3	5	2	3	3	5	**1**	3	4	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	3	5
fri26	3	3	4	3	4	4	2	3	3	**1**	4	3	5	**1**	3	4	**1**	2	**1**	**1**	**1**	**1**	**1**	**1**	3	5
gr24	3	4	5	3	4	4	4	4	5	2	4	3	5	**1**	3	4	2	**1**	2	**1**	**1**	2	**1**	**1**	3	5

When was which kernel chosen (or when was which kernel given a large weight)? And did the largest weights determined by the combination approaches correspond to the kernels chosen by the selection approaches?
The superposition method (combineSuper) is ignored here, because it did not perform well. It is also the only approach that employs a local, data-dependent combination approach. Furthermore, we only present the respective figures for one of the correlation-based selection methods (selectFDC).
The figures discussed in the following visualize the behavior of the multi-kernel methods on two problem instances. For the selection methods, the relative frequency of selecting a specific kernel is reported. For the combination methods, the average of the weights, divided by the sum of the weights is reported. Hence, the plotted values scale from zero to one. In both cases, averages over five iterations of CEGO are reported.
The respective plots for the WTP instance wt40a are shown in Fig. 6.4. Clearly, the two variants of the position distance were favored by the selection methods. While there was some variance in the beginning, the choice converged to the two position distances as more data became available. The weights of the combination approaches had a different pattern, with much more variance. While the position distances received some weight, other distances, such as LCStr and Hamming, seemed to be more important. At the same time, the optimization performances of all these algorithms were quite similar. Only selectFDC received worse ranks, see Table 6.2. A potential reason may be the occasional selection of the R or Adjacency distance, even late in the runs.
A visualization for the pUNIS30 instance is shown in Fig. 6.5. Here, the behavior of the selection methods was more diverse. All selection approaches started with a preference for the cosine distance. Later, selectFDC and selectCV showed more preference for the swap distance. Interestingly, combineMLE did the opposite: The cosine distance received large weights at the end of the run. The stacking approach mostly focused on the swap distance, and the behavior of combineLinear was comparatively random.
It may seem surprising that the observed performances of the multi-kernel methods were fairly similar, while their internal behavior in terms of weights and selection frequencies differed considerably. But in fact, several of the proposed kernels provide very similar, potentially redundant information. This is obvious, e.g., for the two variants of the position distance. Some permutation distances strongly correlate with each other, as discussed by Schiavinotto and Stützle [221].

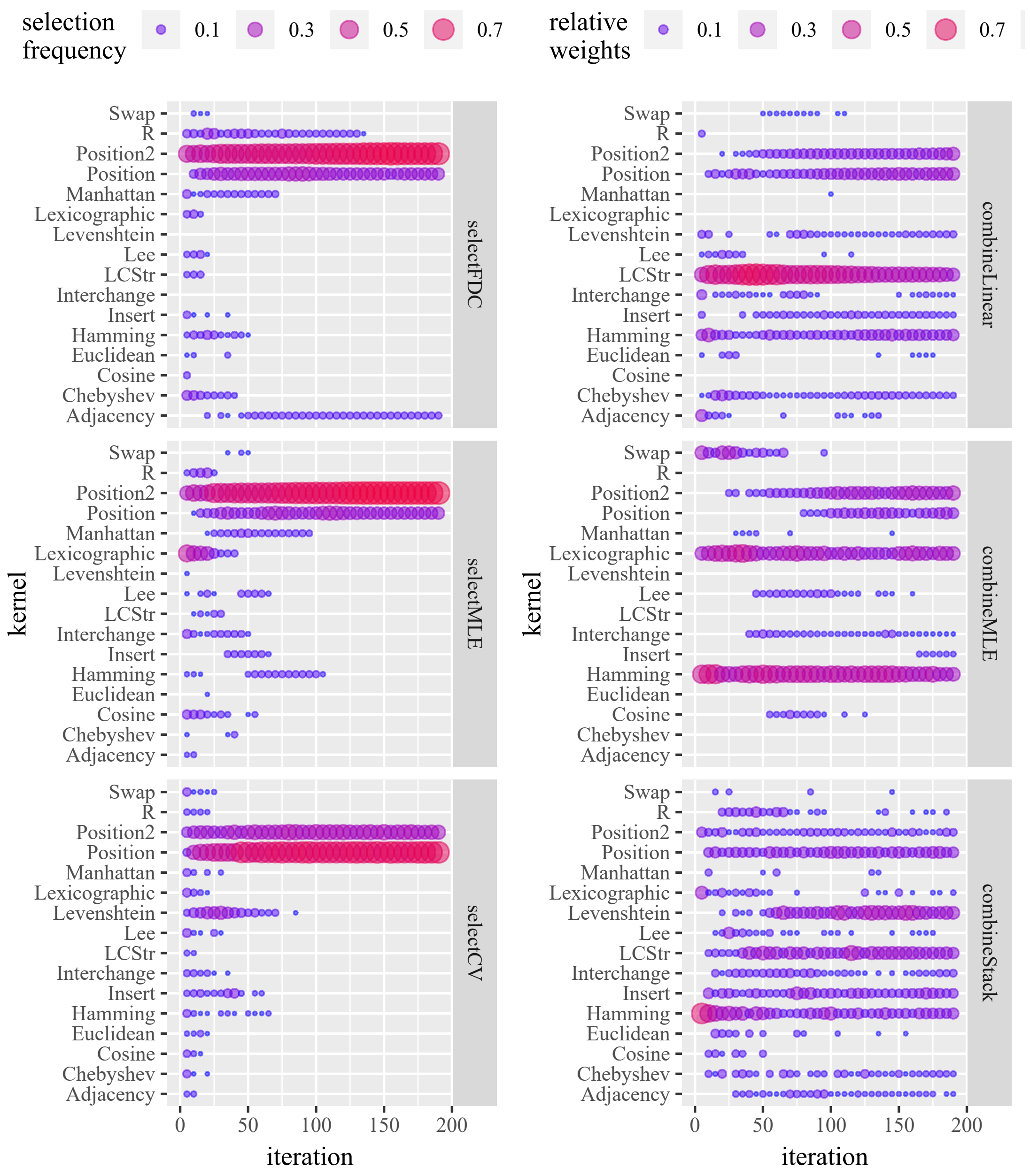


Figure 6.4: *Frequency of selection and relative weights assigned to different kernels during an optimization run on problem wt40a. The weights and frequencies are reported every five iterations and as an average over these five iterations.*

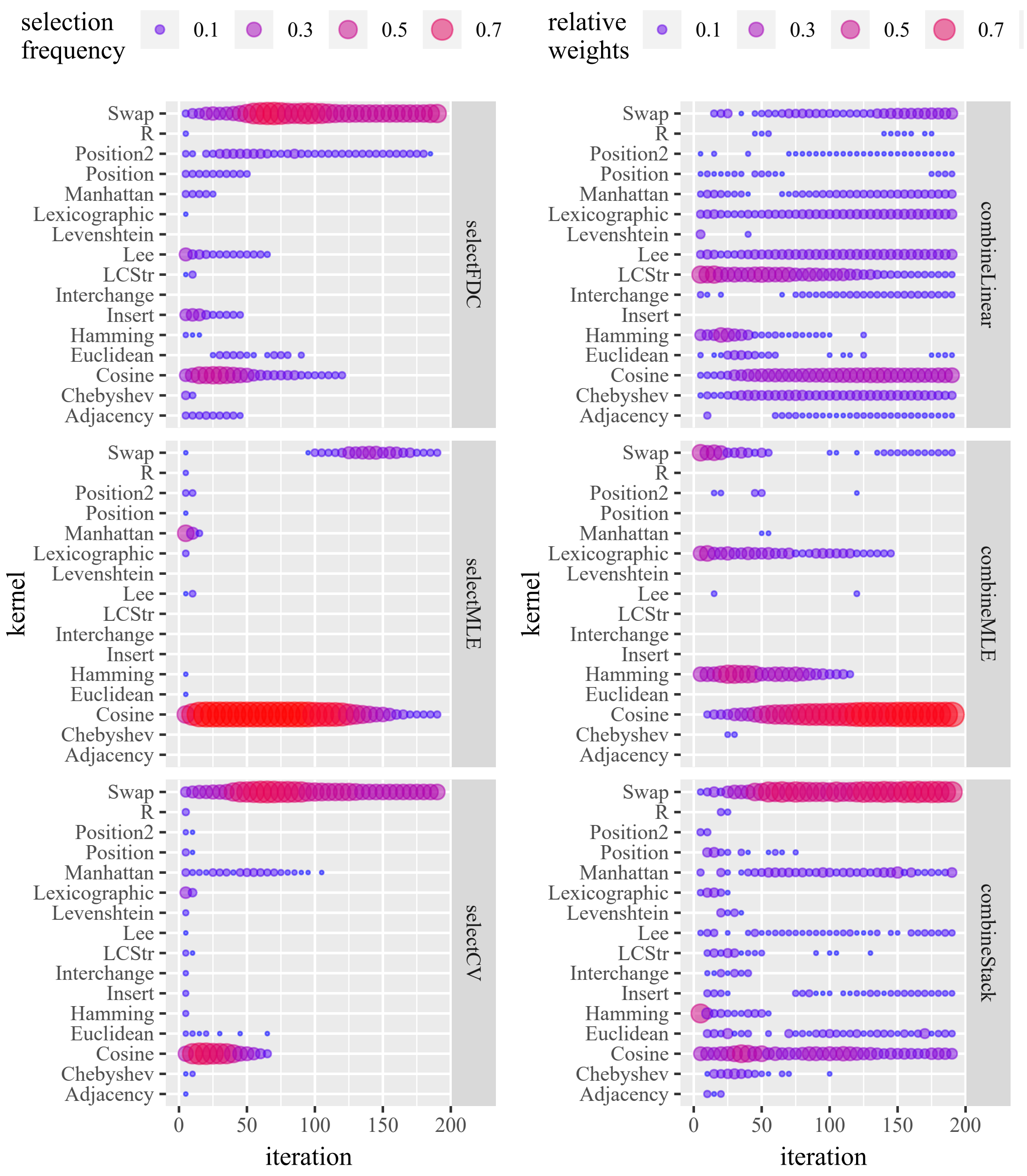


Figure 6.5: *Frequency of selection and relative weights assigned to different kernels during an optimization run on problem pUNIS30. The weights and frequencies are reported every five iterations and as an average over these five iterations.*

Runtimes One final aspect of our experiments is the runtime. We list the algorithms and some of their key features in Table 6.3, including runtimes from the optimization experiments. Clearly, the runtime depends on the method as well as the problem. If a problem requires a fast-to-compute kernel (e.g., Hamming distance), a selection approach that identifies this kernel correctly takes less time than a sub-optimal kernel (e.g., Lexicographic distance). This situation may be reversed if the computational effort of the best kernel is large. Of course, the implementation and employed hardware also have an impact. Therefore, the runtimes given in Table 6.3 should be considered with care.
Still, some general observations can be made, which are supported by our understanding of the underlying methods. The computational effort was especially large for all methods that combined or selected fully trained models rather than just kernels. Most affected by this were the methods that employed models trained on several subsets of the data, i.e., cross-validation and stacking. Finally, the combination approaches required more time than comparable selection approaches.
In terms of the listed values, CV and FDC worked best in these specific experiments. All other methods were worse than or equal to CV or FDC. Please note that a difference in the listed performance ranks does not necessarily imply a statistically significant difference between two specific algorithms (cf. Appendix A).

Table 6.3: *Comparison of key features of various multi-kernel methods. The first row gives ranges of the corresponding values for single-kernel runs. The column "acts on" indicates whether the respective method selects / combines from a set of individual kernels, or from a set of trained models. The runtime is the mean runtime of an algorithm run in minutes. The last column captures the ranks from the experimental analysis, i.e., from the first row of Table 6.2.*

method	type	acts on	runtime	performance
single-kernel			20-49	4-12
FDC	selection	kernels	22	2
FDDC	selection	kernels	22	3
CV	selection	models	98	1
MLE	selection	kernels	46	2
MLE	combination	kernels	253	3
linear regression	combination	kernels	99	4
stacking	combination	models	1298	2
superposition	combination	models	257	6

6.4 Conclusions

In this chapter, we investigated how multiple kernels can be selected or combined to provide better models for SMBO. The *how* was answered by introducing various multi-kernel methods that select or combine kernels.

From a performance point of view, the experimental investigation revealed that it was more important to select some multi-kernel approach rather than selecting the best multi-kernel approach. The best performing multi-kernel approach on all test instances was cross-validation (selectCV). It was closely followed by several other multi-kernel methods. Only the approach based on the superposition of Kriging models had unsatisfactory performance for this use case.
While the selection methods seemed to perform well, other problem classes may require the combination rather than the selection of kernels. It is likely that the kernels tested in our experiments did not complement each other, at least for these specific problem instances. In practice, it may be recommendable to use a combination approach if it is unclear whether the available kernels complement each other. If there are indications that they do not complement each other, a selection approach may be more appropriate especially due to the lower computational cost.
To summarize this chapter, we answer the initially raised questions:

Answer 6.1. Selection. How can the best kernel be chosen without prior knowledge?
We introduced several methods for the selection of kernels based on correlation measures, MLE, and cross-validated errors. In an experimental study, these methods seemed to be successful at selecting the right kernel. The cross-validation method performed best.

Answer 6.2. Combination. Can several kernels be combined?
Similarly to selection, various approaches can be used to combine kernels. These methods require a significantly larger computational effort. While this additional effort did not pay off in our experiments, combination approaches may be preferable for other problem types.

Finally, definiteness may be a reason why some individual kernels performed poorly in our experiments. At this point, we do not know whether the employed kernels are definite. The following Chapter 7 shows how the definiteness of a kernel can be determined.

Chapter 7

Definiteness of Distance-based Kernels: An Empirical Approach

The kernels discussed throughout this thesis are usually required to be positive semi-definite or conditionally negative semi-definite (cf. Sections 2.3 and 4.4). This is not guaranteed for arbitrary kernels. Of course, this issue is not exclusive to SMBO. In general, definiteness plays an important role in statistics and machine learning [86, 251, 224].
As outlined in Section 4.4, the definiteness of a kernel matrix can be determined, e.g., by analyzing its eigenvalues. It is more difficult to determine the definiteness of a kernel function. Section 7.1 briefly discusses some ways to determine the definiteness of a kernel by analytical or theoretical procedures. In practice, it may not always be feasible to approach this problem analytically. Therefore, this chapter tries to answer the following two questions:

Question 7.1. Discovery. Is there an efficient, empirical approach to determine the definiteness of kernel functions that are based on arbitrary distance measures?

Question 7.2. Measurability. Can we quantify to what extent a lack of definiteness is problematic in practice?

As a solution, we propose an empirical approach in Section 7.2 that is based on sampling or optimization of eigenvalues. If the approach detects an indefinite matrix, the respective kernel function is demonstrated to be indefinite. Our approach may also yield insights into measurability. Firstly, a probability of finding indefinite matrices can be estimated. Secondly, the impact of a kernel's indefiniteness on model accuracy

The following sections are based on the article *"An Empirical Approach for Probing the Definiteness of Kernels"* by Zaefferer et al. [266]. The material was revised to embed it into the context, notation, and structure of this thesis. Otherwise, a majority of the text has been adopted verbatim from the original document. The experiments and analysis were not changed.

can be linked to the magnitude of eigenvalues. As in the previous chapters, we use permutation distance measures to demonstrate our ideas.

7.1 Dealing with Definiteness

To handle the issue of a kernel's definiteness, different (not mutually exclusive) approaches can be found in the literature.

- **Proving:** Sometimes, definiteness of a specific function can be *proven* (or disproven) by theoretical considerations (cf. [32]). One example of such a proof is discussed in Section 9.2. In practice, this may be an infeasible approach for more complex kernels [193].
- **Designing:** Functions can be *designed* to be definite [116, 99, 171]. Especially noteworthy are the so-called convolution kernels by Haussler [116]. They provide a method to construct PSD kernels for structured data. For a similar purpose, Gärtner et al. [100] show how to design a syntax-based PSD kernel for structured data. However, convolution kernels may be hard to design [100]. Kernels and distance measures may also be predetermined for a certain application. Then, the laborious design of a new, definite kernel may not be economical.
- **Adapting:** Algorithms or kernel functions may be *adapted* to be usable despite a lack of definiteness. This may affect the performance of the derived model negatively. More details can be found in Chapter 8.

To supplement these techniques, we propose an empirical approach to detect whether a function is definite. An empirical approach may help to overcome the difficulty of theoretical considerations or designed kernels. Empirical results may also be a starting point for a more formal approach. Furthermore, it may give a quick answer on whether the modeling algorithm has to be adapted to indefinite kernels.

7.2 Probing Definiteness

As throughout this thesis, we consider the definiteness of a distance-based exponential kernel, $\mathrm{k}(x, x') = \exp(-\theta d(x, x'))$. The kernel is definite if the underlying distance function is CNSD. For a given distance matrix $\mathbf{D}$, CNSDness is determined by the largest eigenvalue $\hat{\lambda}$ of $\hat{\mathbf{D}}$, as described in Section 4.4.1. $\mathbf{D}$ is not CNSD if $\hat{\lambda} > 0$. We could also probe the kernel matrix $\mathbf{K}$, but in this case, the kernel parameter θ would have to be dealt with.
Of course, it does not suffice to check the definiteness of a single matrix $\mathbf{D}$. This would be only one possible outcome of the respective kernel or distance function. Hence, numerous data sets with their respective distance matrices have to be generated to determine whether any of the matrices are non-CNSD (research question Q1: discovery) and to what extent this may affect a model (Q2: measurability). For smaller,

finite spaces, a brute force approach may be viable. All potential matrices $\mathbf{D}$ can be enumerated and checked for CNSDness. Since this quickly becomes computationally infeasible, we propose to use sampling or optimization instead.

7.2.1 Random Sampling for Probing Definiteness

We propose a simple random sampling approach, which we refer to as Random Sampling for Probing Definiteness (RSPD). While we illustrate this for probing CNSDness of a distance measure, this could easily be extended to probing PSDness of, e.g., a correlation measure. The RSPD approach is presented in Algorithm 5.

Algorithm 5 Random sampling for probing the definiteness of a distance measure.

```
1: function RSPD(distance(), $t \in \mathbb{N} \setminus 0, n \in \mathbb{N} \setminus 0$)
2:     $n_{\lambda+} = 0$;
3:     for $i = 1$ to $t$ do
4:         create $X_i = \{x^{(i1)}, \ldots, x^{(in)}\}$;        ▷ uniform random sampling
5:         replace duplicates in $X_i$ with unique samples;
6:         $\mathbf{D}_i$ =distance($X_i$);        ▷ all pairwise distances of the set $X_i$
7:         compute $\hat{\mathbf{D}}_i$;        ▷ see Eq. (4.7)
8:         compute $\hat{\lambda}$ of $\hat{\mathbf{D}}_i$;        ▷ largest eigenvalue
9:         if $\hat{\lambda} > 0$ then
10:            $n_{\lambda+} = n_{\lambda+} + 1$;        ▷ count indefinite matrices
11:        end if
12:    end for
13:    $p_{\lambda+} = n_{\lambda+}/t$;
14: end function
```

First, t sets of n samples are randomly generated. Next, the distance matrix is computed for each set. Then, the largest eigenvalue $\hat{\lambda}$ of $\hat{\mathbf{D}}$. This eigenvalue determines whether $\mathbf{D}$ is CNSD. The number of times that the largest eigenvalue is positive is retained as $n_{\lambda+}$. Accordingly, the proportion of non-CNSD matrices is determined with $p_{\lambda+} = n_{\lambda+}/t$.
Obviously, all distance measures that yield $p_{\lambda+} > 0$ are proven non-CNSD. For the case $p_{\lambda+} = 0$, CNSDness is not proven or disproven. That means, RSPD is analogous to a Monte Carlo algorithm for problems of the complexity class RP (randomized polynomial time) [101]. It stops after a fixed number of samples, and allows for a one-sided error. While a rejection result (non-CNSD) is always true, acceptance (CNSD) may err with a probability that depends on the number of samples.
The parameter $p_{\lambda+}$ is an estimator of how likely a non-CNSD matrix is to occur, for the specified set size n. To determine definiteness, the calculation of $\hat{\lambda}$ of $\hat{\mathbf{D}}$ is not mandatory, but it may be useful to distinguish between a pathological case and cases where the matrix $\mathbf{D}$ is just barely non-CNSD ($\hat{\lambda}$ close to zero). We show in Section 7.4.3 that $\hat{\lambda}$ of $\hat{\mathbf{D}}$ can be linked to model quality.

Note that inaccuracies of the numerical algorithm used to compute the eigenvalues might lead to an erroneous sign of the largest eigenvalue. To deal with that, one could try to use exact or symbolic methods or else use a tolerance when checking whether the largest eigenvalue is larger than zero. In the latter case, a matrix $\hat{\mathbf{D}}$ is assumed to be non-NSD if $\hat{\lambda} > \epsilon_{\text{tol}}$, where ϵ_{tol} is a small positive number.

7.2.2 Optimization for Probing Definiteness

RSPD may fail if the number of sets X that yield indefinite matrices is small in relation to the size of $\mathcal{X}$. In such cases, it may be more efficient to replace random sampling with a directed search. We refer to such an approach as Optimization for Probing Definiteness (OPD). Here, a set X is viewed as a candidate solution of an optimization problem. The largest eigenvalue $\hat{\lambda}$ of the transformed distance matrix $\hat{\mathbf{D}}$ is the objective function to be maximized. By maximizing the largest eigenvalue, a positive $\hat{\lambda}$ may be found more quickly (and more reliably) than by random sampling.
This optimization problem is strongly dependent on the kind of solution representation used. Evolutionary Algorithms (EAs) are a good choice to solve this problem because they are applicable to a wide range of solution representations (cf. Section 2.1). For OPD, we use the simple EA from Algorithm 1, with the following configuration.

- **Individual:** X**.** A set $X = \{x^{(1)}, \ldots, x^{(n)}\}$ with set size n is considered as an individual, or candidate solution. Set elements $x^{(i)}$ are samples in the actual search or input space in which the investigated distance operates.

- **Search space:** S^n**.** All possible sets X of size n, i.e., $X \in S^n$.

- **Population:** Z**.** A population or candidate solution set of size t, containing $X_j \in Z$ with $j \in \{1, \ldots, t\}$ and $Z \subseteq S^n$.

- **Objective function:**

$$\begin{aligned} f:\ S^n &\to \mathbb{R} \\ X &\mapsto \hat{\lambda} \end{aligned} \tag{7.1}$$

  where $\hat{\lambda}$ is the largest eigenvalue of the transformed distance matrix $\hat{\mathbf{D}}$ as described in Section 4.4.1. The objective function f is maximized.

- **Mutation:**
  $X' = \texttt{mut}(X) = \{x^{(1)}, \ldots, x^{(j-1)}, \texttt{sub-mutation}(x^{(j)}), x^{(j+1)}, \ldots, x^{(n)}, \}$, with $j \in \{1, \ldots, n\}$. For the sub-mutation operator, any edit operation that works for a sample x can be chosen. For example, in case of permutations, one permutation $x^{(j)}$ is randomly chosen and mutated with typical permutation edit-operations (swap, interchange, reversal, see also Appendix C). The specific edit-operation is called sub-mutation operator, to distinguish between mutation of the individual set X and the sub-mutation of a single permutation $x^{(j)} \in X$.

- **Recombination:** Two sets X are randomly split, and the parts of both sets are joined to form a new set of the same size.
- **Duplicate removal:** The variation operators may create duplicates ($x^{(i)} = x^{(j)}$ with $i \neq j$). These are undesirable and are replaced by randomly generated, unique samples $x^* \notin X$.
- **Stopping criterion:** The optimization run can stop when it finds some solution set X that yields $\hat{\lambda} > \epsilon_{\text{tol}}$, where ϵ_{tol} is a small positive number. Or else, the optimization run can stop when the budget of evaluations of the objective function is exhausted.

7.3 Experimental Validation

As a proof-of-concept, we chose to perform experiments with permutation distances, which are frequently used in this thesis. These distance measures were used indiscriminately in Sections 5.3 and 6.3, without knowing whether they are actually CNSD. This section intends to investigate this question. We consider relatively small set sizes n, due to our context of SMBO.

7.3.1 Random Sampling

We decided to use $t = 10,000$ randomly generated sets of permutations for each RSPD run. Each set contained n permutations and each permutation had m elements. As a summary statistic of all t sets, the largest $\hat{\lambda}$ as well as the average of $p_{\lambda+} = n_{\lambda+}/t$ were recorded (see Algorithm 5). The tolerance value was set to $\epsilon_{\text{tol}} = 10^{-10}$. Each RSPD run was repeated 10 times, to receive a robust estimate of the recorded statistics.
Two batches of experiments were examined. The first included all sixteen distance measures, with $n \in \{4, \ldots, 20\}$ and $m \in \{4, \ldots, 15\}$. In the second batch, larger sizes $n \in \{21, \ldots, 40, 45, 50, 60, 70, 80, 90, 100\}$ were tested, but the permutations were restricted to $m \in \{5, \ldots, 15\}$, and the distance measures were limited to LCStr, Insert, Chebyshev, Levenshtein, and Interchange.

7.3.2 Optimization

To be comparable to RSPD, the budget for each OPD run was set to $10,000$ fitness function evaluations. A run stopped if the budget was exhausted or if $\hat{\lambda} > \epsilon_{\text{tol}} = 10^{-10}$. The population size of the EA was set to $n_{\text{pop}} = 100$, and the respective number of offspring was $n_{\text{off}} = \lfloor 0.5\, n_{\text{pop}} \rfloor$. Truncation selection was used.
To identify bias introduced by the choice of the sub-mutation operator (which may have a strong interaction with the distance measures), each EA run was repeated with three different sub-mutation operators: swap, interchange, and reversal (see Appendix C).

All sixteen distance measures were tested, with $n \in \{4, \ldots, 20\}$ and $m \in \{4, \ldots, 15\}$. With ten repeats and the three different sub-mutation operators, an overall number of $97,920$ experiments are run, each with $10,000$ fitness function evaluations.

7.3.3 Other Distance Measures

To show that RSPD and OPD are not limited to permutation distance measures, we also explored other search domains and their respective distances. However, these are not discussed in further detail. Instead, we generated a list of minimal examples: the smallest (with respect to dimension) indefinite distance matrix found for each tested distance measure. The examples include distance measures for permutations, signed permutations, labeled trees, and strings.

7.4 Observations and Discussion

We split the summary of our results and their discussion into four parts. First, we discuss the results of RSPD in Section 7.4.1, then those of OPD in Section 7.4.2. We extend this by linking the value of $\hat{\lambda}$ to model quality in a smaller test study in Section 7.4.3. Finally, we briefly discuss in Section 7.4.4 that this approach can also be used for other data types.

7.4.1 Sampling Results

The proportions $p_{\lambda+}$ of sets with positive eigenvalues are summarized in Fig. 7.1. The eigenvalues $\hat{\lambda}$ are depicted in Fig. 7.2. Only the five distance measures that achieved positive $\hat{\lambda}$ are shown. That means, five of sixteen distance measures were proven non-CNSD: Longest Common Substring, Insert, Chebyshev, Levenshtein, and Interchange. No counter-examples were found for the remaining eleven measures. That did not prove that they are CNSD (although some are known to be CNSD, e.g., the Euclidean distance, the Swap distance [135], and the Hamming distance [128]), but indicated that it may be unproblematic to use them in practice. Conversely, previous results indicated that even non-CNSD distances might produce good results in practice. For example, the experiments in Section 6.3 showed that the Levenshtein and Insert distance seemed to work well for the WTP instances, despite being non-CNSD. Even a non-CNSD distance measure *may* yield PSD kernel matrix $\mathbf{K}$, depending on the specific data set and the kernel parameters. We do not propose that non-CNSD distance measures should be avoided, but that their application should be handled with care.
Regarding $p_{\lambda+}$, some trends can be observed in Fig. 7.1. For indefinite distance measures, increasing the set size (n) led to larger values of $p_{\lambda+}$, up to $p_{\lambda+} = 100\%$. Obviously, a larger set is more likely to contain combinations of samples that yield positive eigenvalues $\hat{\lambda}$. Moreover, the upper bound of these eigenvalues increases with increasing matrix size [67].

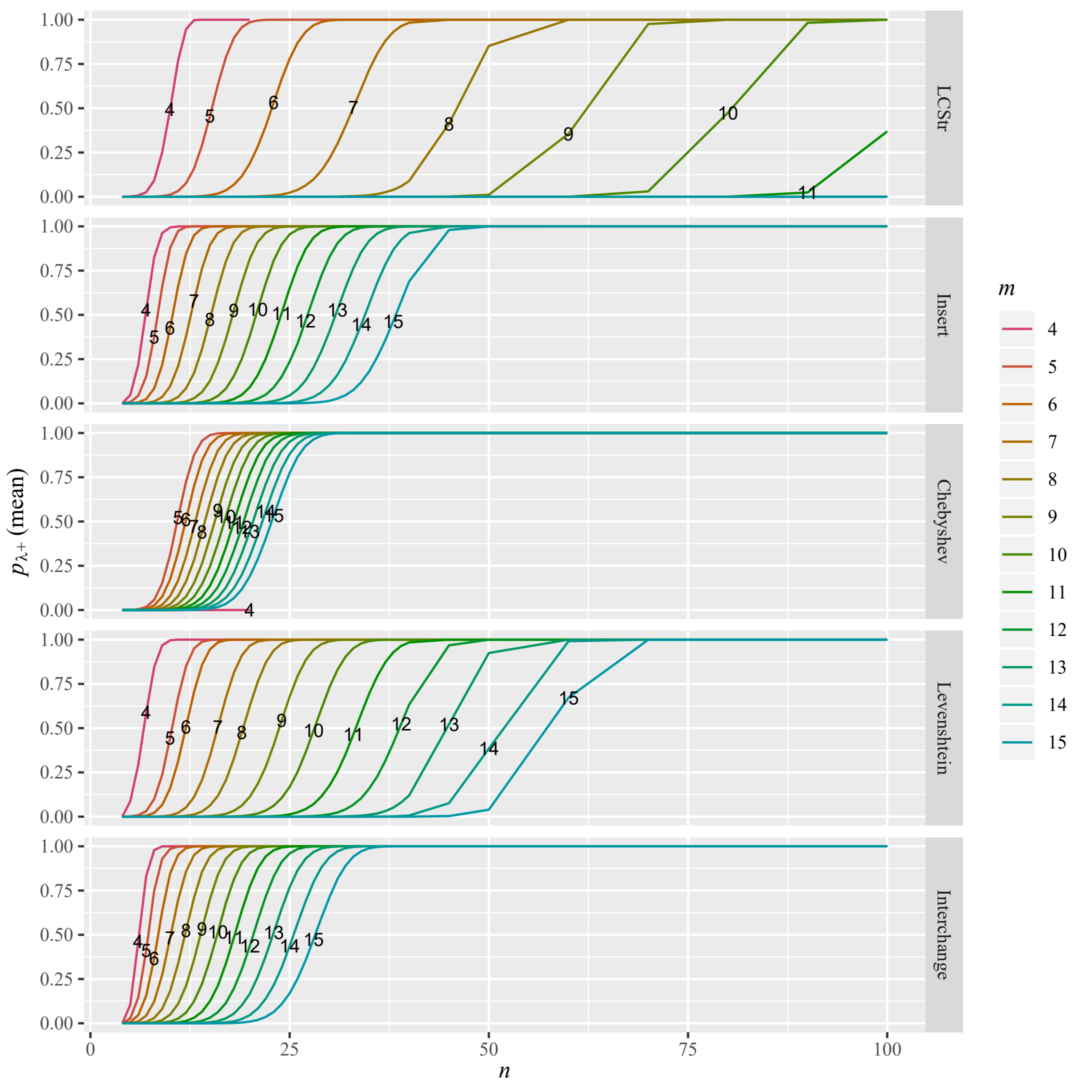


Figure 7.1: *Mean proportion of sets yielding matrices $\hat{\mathbf{D}}$ with positive $\hat{\lambda}$ found with random sampling. Only distance measures that achieved positive $\hat{\lambda}$ are shown. The numeric labels and the color indicate the respective value of m. The figure is taken from [266].*

In contrast to the set size, increasing the number of permutation elements (m) decreased the proportion of positive eigenvalues $p_{\lambda+}$ in all five cases. This can be attributed to a larger search space. Overall, none of the distance measures showed exactly the same behavior. The LCStr distance had the least problematic behavior. Only few sets (of comparatively large size) yielded positive eigenvalues with the LCStr distance. The Interchange, Levenshtein, and Insert distance all yielded relatively large $p_{\lambda+}$ for small set sizes n. On the other hand, the Chebyshev distance started to produce non-zero $p_{\lambda+}$ for relatively large n. The number of permutation elements had a weaker influence on the Chebyshev distance. It was the only measure where no positive eigenvalues were found for $m = 4$. Analogous to $p_{\lambda+}$, the magnitude of the eigenvalues

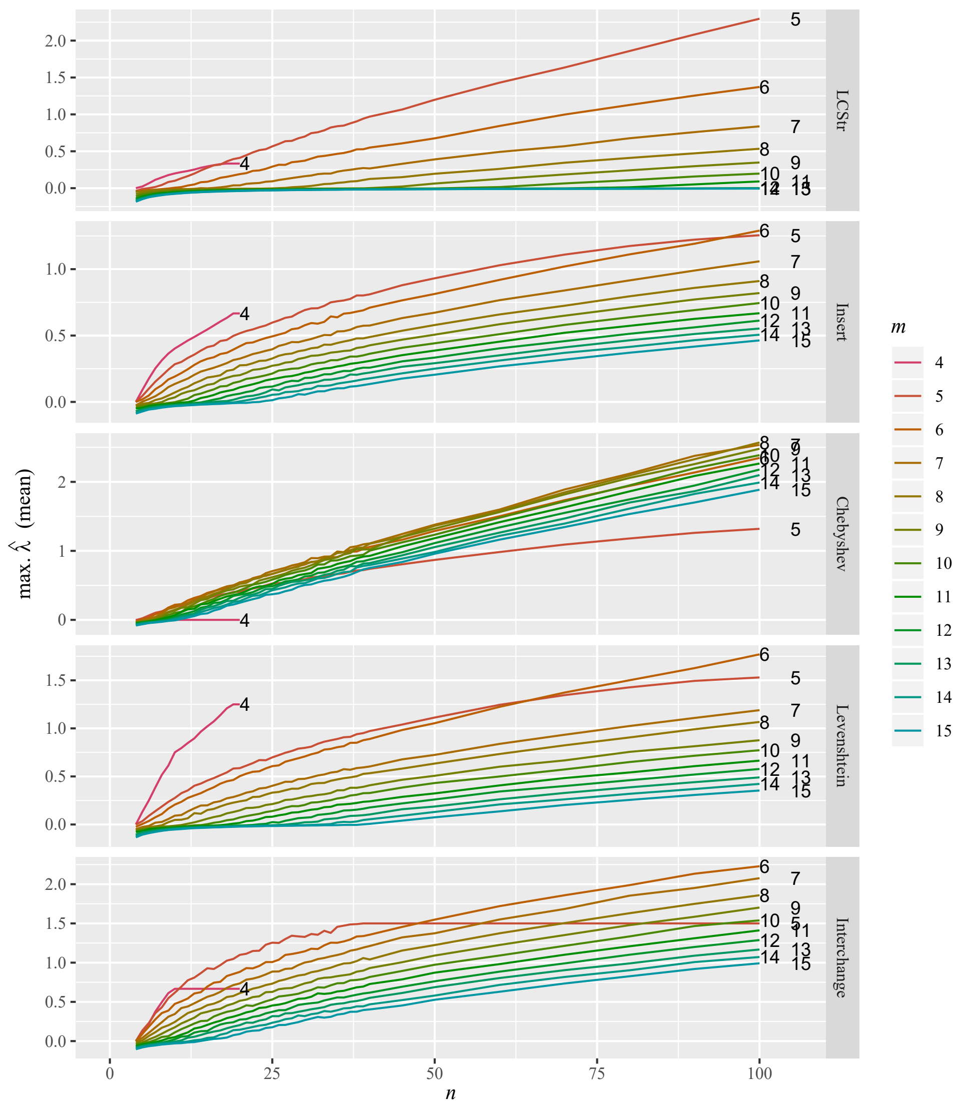


Figure 7.2: *Mean of maximal $\hat{\lambda}$ of $\hat{\mathbf{D}}$ found with RSPD. The maximum is determined over all sets in a single experiment. The mean is determined over ten repeats of the sampling. Only distance measures that achieved positive $\hat{\lambda}$ are shown. The numbers inside the plot and the color indicate the respective value of m. The figure is taken from [266].*

increased with increasing n and decreased with increasing m.

Our findings are confirmed by some results from the literature. Cortes et al. [68] have shown that an exponential kernel function based on the Levenshtein distances is indefinite for strings of more than one symbol. Our experiments show that this result can be easily rediscovered empirically, for the case of permutations. At the same time, these findings also provide a reason for numerical problems observed in a previous study [271].

7.4.2 Optimization Results

The average number of fitness function evaluations that was required to find a matrix $\hat{\mathbf{D}}$ with positive eigenvalues is depicted in Fig. 7.3. OPD and RSPD had similar behavior with respect to n and m. Increasing m led to an increased number of fitness function evaluations. That means, finding positive eigenvalues became more difficult with larger m. Conversely, increasing n reduced the number of required fitness function evaluations. That is, finding positive eigenvalues became easier. In some cases, this effect disappeared for large values of m, e.g., for the LCStr distance, where the averages were more or less constant over n, if $m \geq 10$.
Importantly, the comparison of OPD and RSPD clearly shows that the EA had some success in optimizing the largest eigenvalue of the transformed distance matrix $\hat{\mathbf{D}}$. In several cases, OPD found solutions with positive eigenvalues while RSPD failed to find any. It seemed that the fitness landscapes based on $\hat{\lambda}$ of $\hat{\mathbf{D}}$ were sufficiently smooth to allow for optimization. The eigenvalue $\hat{\lambda}$ seemed to be a good indicator of how close a solution set was to yielding an indefinite kernel matrix.
As expected, the results showed that the choice of sub-mutation operator introduced some bias to the results. For the Insert and LCStr distance, the EA with swap mutation worked considerably better than the other two variants. Therefore, comparisons of these values across different distance measures should be handled with caution. Clearly, other aspects of the optimization algorithm (e.g., selection criteria or recombination operators) may introduce a similar bias.

7.4.3 Verification: Impact on Model Quality

Earlier, we discussed two values that may express the effect of the lack of definiteness in practice, i.e., $p_{\lambda+}$ and $\hat{\lambda}$ of $\hat{\mathbf{D}}$. But what do these values imply?
An advantage of $p_{\lambda+}$ is that it is rather easy to interpret. It can be seen as a probability of generating indefinite matrices. If we assume that a model is unable to deal with indefinite data, the fraction $p_{\lambda+}$ is an estimate of how likely a modeling failure is. For other cases, it may be interpreted as a probability to receive worse results. Still, it is difficult to link $p_{\lambda+}$ to a model performance measure, such as accuracy, without making too many assumptions. We argue that $p_{\lambda+}$ provides useful information, especially when the kernel is designed and probed before sampling any data (e.g., when planning an experiment). We suggest using $p_{\lambda+}$ to support an initial decision (e.g., whether to spend additional time on fixing or otherwise dealing with the indefinite kernel).
In contrast, the parameter $\hat{\lambda}$ is more difficult to interpret. It has the advantage that it may be estimated for a single set or matrix. Hence, we want to determine whether the magnitude of this eigenvalue affects model performance. We expect an influence that depends on the choice of model. Consider, e.g., the Kriging models used throughout this thesis (see Sections 2.3 and 5.1). Kriging may be able to mitigate the problematic eigenvalue by assigning larger θ values to the kernel $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x'))$. Mostly, very large $\hat{\lambda}$ require very large θ. Large values of θ lead to kernel matrices that approximate the identity matrix. Clearly, the identity matrix is PSD. A model based

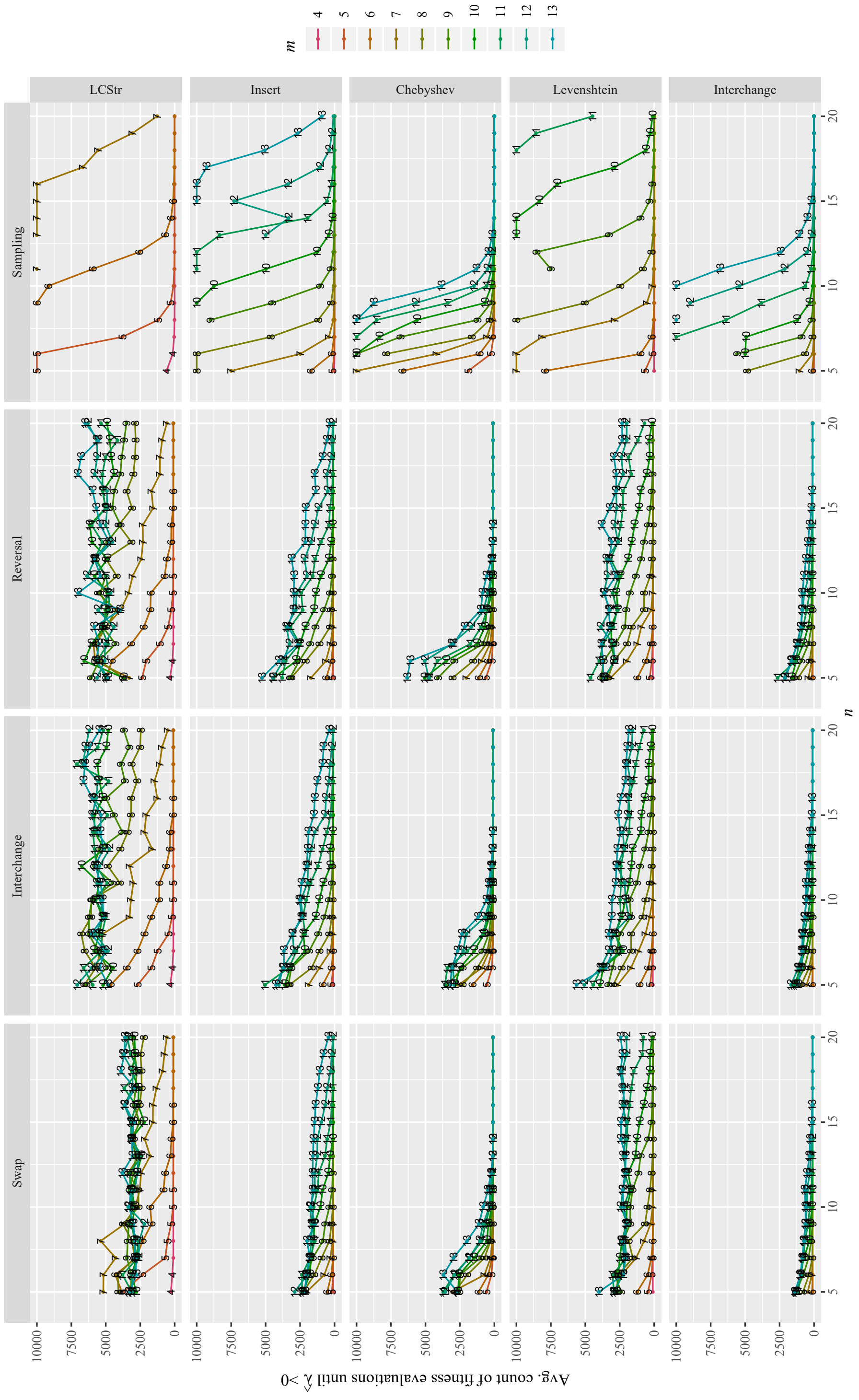


Figure 7.3: *Average number of fitness function evaluations until a positive $\hat{\lambda}$ was found. Missing line segments indicate that no positive $\hat{\lambda}$ were found in any experiment, within the given budget. Averages were computed after removal of cases where no positive values are found. The four columns show results from OPD with each of the three sub-mutation functions as well as RSPD (last column). The five rows depict different distances measures. The figure is taken from [266].*

on the identity matrix would be able to reproduce the training data, but would predict the process mean for all other data points. That means, larger θ values might correct a non-CNSD distance matrix, but could produce a rather poor model.
An experimental test has to consider the potential bias of the used data set. We need to be able to reasonably assume that differences in performance are actually due to the properties of the employed kernel (i.e., a kernel performs poorly because the corresponding $\hat{\lambda}$ is large) rather than properties of the data set (i.e., a kernel performs poorly because it does not fit well to the ground truth of the data set). To that end, we suggest deriving the observations for a test data set from the same distances that are used in the model. That is, we use the unimodal test problems employed in Section 5.3.1.
Hence, we performed the following experiment. We randomly created data sets X of size n with permutations of dimension m, similarly to the random sampling performed earlier. Then, we created training observations by evaluating the distance of each permutation in X to a reference permutation $x_{\text{ref}} = \begin{bmatrix} 1 & \dots & m \end{bmatrix}$, i.e., $y = \mathrm{d}(x, x_{\text{ref}})$. A Kriging model was then trained with this data. We chose Kriging for the sake of a transparent and interpretable test case. Similar experiments could easily be made with support vector regression.
The model was trained with the kernel $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x'))$, and the model parameters (e.g., θ) were determined by maximum likelihood estimation, via the locally biased version of DIRECT [94] with 1,000 likelihood evaluations. For each test, the distance chosen to produce the observations y and the distance used in the Kriging model's kernel were identical.
The Root Mean Square Error (RMSE) of the model was evaluated on 1,000 randomly chosen permutations. The resulting RMSE values and the corresponding eigenvalues $\hat{\lambda}$ are shown in Fig. 7.4. A trend can be observed in the figure, which confirms our expectation. Distances associated to larger $\hat{\lambda}$ tended to produce larger errors.

7.4.4 Additional Simple Examples

To highlight the usefulness of the proposed methods, this section lists small example data sets and the respective indefinite distance matrices that were found in experiments with RSPD and OPD. Besides the permutation distances, we also tested the reversal distance on signed permutations, the edit distance for labeled trees, the Jaro-Winkler distance for strings, and the optimal string alignment distance. For more details, see Appendix B. The respective results are listed in Table 7.1. All the listed distance measures are shown to be non-CNSD. Further examples for non-CNSD distances can be found in Sections 9.1.3 and 9.2.1.

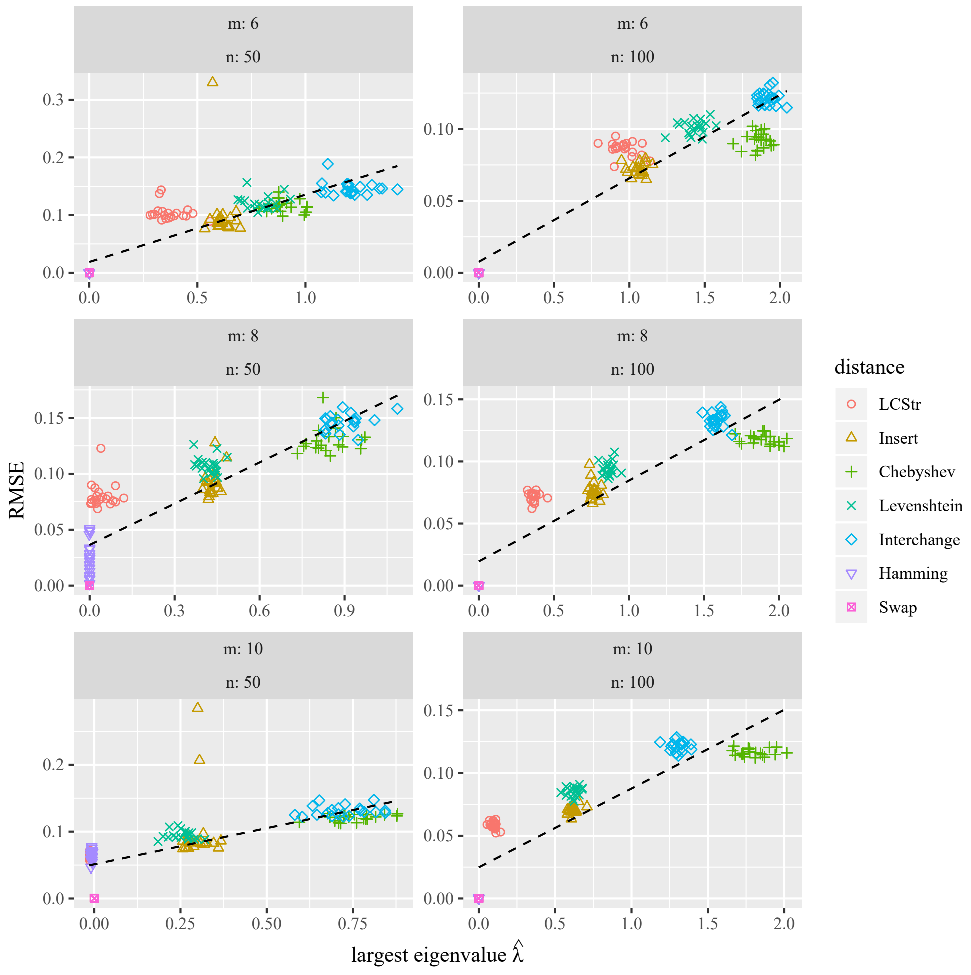


Figure 7.4: *RMSEs of a Kriging model plotted against the eigenvalue $\hat{\lambda}$ of $\hat{\mathbf{D}}$. The denoted distance measures are used in the underlying data set as well as the model itself. The dashed line depicts a linear trend. The figure is taken from [266].*

Table 7.1: *Simple examples for sets that produce indefinite distance matrices, with the respective largest eigenvalue. The matrix in the table is the actual distance matrix, while the eigenvalue refers to the transformed matrix* $\hat{\mathbf{D}}$ *derived from Eq.* (4.7)*. The lower triangular matrix is omitted due to symmetry. Some distances are scaled to* $[0, 1]$*.*

(a) *Unsigned permutations*

Permutations, Insert, $n = 5$, $m = 4$, $\hat{\lambda} \approx 0.090$

i	$x^{(i)}$	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$	$d_{i,5}$
1	[1 2 3 4]	0	$1/3$	$1/3$	$2/3$	$1/3$
2	[1 3 4 2]		0	$2/3$	$1/3$	$2/3$
3	[2 3 4 1]			0	$1/3$	$2/3$
4	[3 4 1 2]				0	$1/3$
5	[4 1 2 3]					0

Permutations, Interchange, $n = 5$, $m = 4$, $\hat{\lambda} \approx 0.090$

i	$x^{(i)}$	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$	$d_{i,5}$
1	[1 2 3 4]	0	$1/3$	$1/3$	$2/3$	$1/3$
2	[1 2 4 3]		0	$2/3$	$1/3$	$2/3$
3	[1 3 2 4]			0	$1/3$	$2/3$
4	[1 3 4 2]				0	$1/3$
5	[1 4 3 2]					0

Permutations, Levenshtein, $n = 5$, $m = 4$, $\hat{\lambda} \approx 0.135$

i	$x^{(i)}$	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$	$d_{i,5}$
1	[1 2 4 3]	0	1	$1/2$	$1/2$	1
2	[2 3 1 4]		0	$1/2$	$1/2$	1
3	[2 4 3 1]			0	1	$1/2$
4	[3 1 2 4]				0	$1/2$
5	[3 4 2 1]					0

Permutations, LCStr, $n = 5$, $m = 4$, $\hat{\lambda} \approx 0.023$

i	$x^{(i)}$	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$	$d_{i,5}$
1	[1 3 2 4]	0	$2/3$	$1/3$	$1/3$	$2/3$
2	[2 4 1 3]		0	$1/3$	$1/3$	$2/3$
3	[3 2 4 1]			0	$2/3$	1
4	[4 1 3 2]				0	$2/3$
5	[4 2 1 3]					0

Permutations, Chebyshev, $n = 5$, $m = 5$, $\hat{\lambda} \approx 0.034$

i	$x^{(i)}$	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$	$d_{i,5}$
1	[1 5 3 4 2]	0	$1/4$	$3/4$	$3/4$	1
2	[2 5 3 4 1]		0	1	1	$3/4$
3	[4 2 3 1 5]			0	$2/4$	$1/4$
4	[4 3 1 2 5]				0	$1/4$
5	[5 3 2 1 4]					0

(b) *Other*

Signed Perm., Reversal, $n = 5$, $m = 5$, $\hat{\lambda} \approx 0.016$

i	$x^{(i)}$	$d_{i,j=1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$	$d_{i,5}$
1	[4 5 -1 -2 -3]	0	$4/6$	$5/6$	$3/6$	$2/6$
2	[2 1 3 -4 -5]		0	$2/6$	$3/6$	$5/6$
3	[-2 1 3 5 4]			0	$5/6$	$3/6$
4	[4 -2 3 1 -5]				0	$2/6$
5	[4 -2 1 -5 -3]					0

Labeled Trees, Edit dist., $n = 5$, $\hat{\lambda} \approx 0.026$

i	$x^{(i)}$	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$	$d_{i,5}$
1	$\{b\{c\{b\}\}\}$	0	2	1	3	1
2	$\{b\}$		0	1	3	1
3	$\{b\{c\}\}$			0	2	2
4	$\{a\{c\}\{a\}\}$				0	3
5	$\{c\{b\}\}$					0

Strings, Optimal String Alignment, $n = 5$, $\hat{\lambda} \approx 0.102$

i	$x^{(i)}$	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$	$d_{i,5}$
1	abc	0	1	2	3	1
2	acc		0	3	2	2
3	cba			0	1	2
4	caa				0	2
5	bac					0

Strings, Jaro-Winkler, $n = 4$, $\hat{\lambda} \approx 0.046$

i	$x^{(i)}$	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$
1	bbbb	0	1	$1/6$	$3/6$
2	aaaa		0	$3/6$	$1/6$
3	bbba			0	$3/6$
4	aaab				0

7.5 Conclusions

The main goal of this chapter was to investigate the definiteness of kernels or distances. Two empirical approaches were suggested. RSPD samples from the space of solution sets, to find any sets that yield indefinite kernel or distance matrices. In the alternative OPD approach, a directed search with an EA is used instead. This directed search is useful if indefinite matrices are rare. The EA maximizes the eigenvalue that is critical for the definiteness of the corresponding matrix.
As a proof-of-concept, the approaches where applied to distance measures for permutations. Five non-CNSD distance measures were identified: the LCStr, Insert, Chebyshev, Levenshtein, and Interchange distance.
It seemed that the optimization approach was successful. It was able to outperform the sampling approach in discovering sets with indefinite kernel matrices. The success of the EA indicates that the fitness landscape posed by the corresponding eigenvalue is sufficiently smooth and has an exploitable structure. This suggests that the largest eigenvalue is an indicator of how far a certain solution set is from producing an indefinite matrix. In an additional set of experiments, we further verified that larger eigenvalues could be linked to a decrease in model quality. This is the basis of the following responses to our research questions:

Answer 7.1. Discovery: Is there an efficient, empirical approach to determine the definiteness of kernel functions that are based on arbitrary distance measures?

Random sampling (RSPD) from the space of potential candidate solution sets allows identifying solution sets that lead to non-CNSD distance matrices. Where indefinite matrices are rare (and hence more likely to be missed by RSPD), an optimization approach (OPD) may be more successful. While neither RSPD nor OPD are able to prove definiteness, both are able to disprove it. At the same time, if no negative results are found it is reasonable to assume that using the respective distance/kernel function is feasible in practice.

Answer 7.2. Measurability: Can we quantify to what extent a lack of definiteness is problematic?

The sampling approach (RSPD) yields a proportion of potentially non-CNSD matrices, which in turn yields an estimate of how problematic a distance measure is. The number of evaluations required by OPD gives a similar estimate, but it is potentially biased by configuration of the optimization algorithm. Moreover, the success of the optimization approach (OPD) suggests that the critical eigenvalue $\hat{\lambda}$ indicates how close certain sets are to producing an indefinite matrix. Additional experiments showed that this eigenvalue could be linked to model performance.

One problem of RSPD and OPD is that separate experiments have to be run for different set sizes. Yet, SMBO usually deals with varying data set sizes, starting from small sets of initial solutions, which grow during an SMBO run. For future research, it may be interesting to allow the EA to change the set size. Clearly, one issue would be that enlarging the sets might quickly lead to a trivial solution. Larger sets naturally lead to

larger $\hat{\lambda}$ of $\hat{\mathbf{D}}$. Hence, there is a trade-off between the largest eigenvalue and the set size. A multi-objective EA (e.g., NSGA-II [74] or SMS-EMOA [36]) may be used to handle this issue by simultaneously maximizing $\hat{\lambda}$ and minimizing the set size n.
In addition, the herein described kernels and distances are not the full story. There may be other kernels where the relation between distance measure and kernel function is not as straightforward, or where parameters of the distance measure or the kernel function have more complicated influence than θ in the exponential kernel. In those cases, it may be necessary to adapt the proposed method to, e.g., include parameters in the RSPD and OPD procedures.
Finally, it remains unclear what the best course of action is if a kernel is determined to be indefinite. The following Chapter 8 shows how Kriging models may deal with this case.

Chapter 8

Kriging-based Optimization with Indefinite Kernels

Chapter 7 described methods that determine whether a kernel or distance is indefinite, employing an empirical approach. Alternatively, it may be possible to show by theoretical means that a kernel is indefinite. Regardless of how we discover that information, we need some way to deal with indefinite kernels.
Clearly, even indefinite kernels may still be used in most models, depending on the specific data sets and the model's configuration. For instance, the algorithms presented in Chapter 6 made use of distances that have been shown to be indefinite in Chapter 7.
There are several ways how models can deal with indefinite kernels. Firstly, some data sets may be unproblematic, especially if the sets are small (cf. Chapter 7). Secondly, a larger value of the kernel parameter (θ) may produce a PSD kernel matrix despite an indefinite distance matrix. Thirdly, the employed Kriging models used the nugget effect. The nugget effect is one potential way of dealing with numerically problematic (close to singular) or even indefinite correlation matrices. But does this provide well performing models, and is it the only solution? In general:

Question 8.1. Indefinite Kriging. How can Kriging deal with indefinite kernels?

To answer this question, this chapter first considers some existing methods from the field of SVM learning. Then, these methods are transferred and adapted to Kriging, showing how indefinite kernels may be treated in this context. A connection of the

This chapter is partially based on the article *"Efficient Global Optimization with Indefinite Kernels"* by Zaefferer et al. [264]. It has been extended and rewritten. Some parts are taken verbatim from the original publication. An illustration with a one-dimensional example was added. Further, the experimental setup was extended to deal with test problems that are more varied and of a higher dimension. Correspondingly, the experimental analysis had to be adapted. The remarks on non-stationarity are also not part of the original article. Finally, some additional repair methods based on a linear combination and a nearest-neighbor approach were added.

presented approaches to non-stationary models is outlined. Finally, a numerical study compares the different approaches that are proposed.

8.1 Correcting Indefinite Kernels

Several recent studies dealt with indefinite kernels in the context of SVMs and related machine learning methods [105, 170, 111, 262, 186, 60, 61, 167, 165, 109, 4, 166, 222, 3]. A survey is given by Schleif and Tino [222]. Indefinite kernels are only rarely discussed in connection with Kriging or Gaussian process regression [169, 41, 12].
Four different approaches to handling indefinite kernels can be identified in the literature.

- *Spectrum transformations* transform the kernel matrix such that all of its eigenvalues (i.e., the spectrum) become positive (or negative, if desired). They have mostly been applied in SVMs [222] and, to a lesser extent, for Gaussian Processes [12]. Spectrum transformations are outlined and extended by repair methods in Sections 8.1.1 to 8.1.3.

- *Nearest matrix* algorithms (Section 8.1.4) try to find matrices that fulfill some conditions and are as close as possible to the original matrix (under some norm) [120, 102].

- *Feature embedding* (Section 8.1.5) interprets the values produced by an indefinite kernel as features. Then, it uses a standard, definite kernel to compute a surrogate similarity based on these features. As the name implies, this approach embeds the indefinite kernel in a definite one.

- *Method modifications* imply changes to the modeling method itself (e.g., SVM), such that the kernel is no longer required to be definite. One example for SVMs is to convert the quadratic programming problem to a linear one (LP-SVM or 1-norm SVM [170, 275, 166]). Most approaches of this type are not directly applicable to Kriging. Hence, they are not discussed in the following.

The distinction between these categories is ambiguous. A method may be a combination from several approaches, or fit into multiple categories. Still, it is a useful basis, which is in similar form used by other authors, e.g., [166, 222].
In the following discussion, $\tilde{\mathbf{K}}$ denotes the modified, definite variant of the kernel matrix $\mathbf{K}$. Respectively, $\tilde{\mathbf{k}}$ denotes the modified variant of the vector of similarities $\mathbf{k}$ between training samples and a new sample (as employed in Eq. (2.4)). Similarly, corrected distances are denoted with $\tilde{\mathbf{D}}$ and $\tilde{\mathbf{d}}$.

8.1.1 Spectrum Transformation: Kernel

Transformation of the Kernel Matrix The spectrum transformation is based on the eigen-decomposition

$$\mathbf{K} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}.$$

Here, $\mathbf{U}$ is the matrix of eigenvectors of $\mathbf{K}$, and $\mathbf{\Lambda} = \mathrm{diag}(\boldsymbol{\lambda})$ is the diagonal matrix that contains the corresponding eigenvalues. The goal of the transformation is to turn an indefinite $\mathbf{K}$ into a PSD $\tilde{\mathbf{K}}$. Chen et al. [61] describe this with a linear transformation based on a vector $\mathbf{a} \in \mathbb{R}^n$:

$$\begin{aligned} \tilde{\mathbf{K}} &= \mathbf{A}\mathbf{K} \\ &= \mathbf{U}\mathrm{diag}(\mathbf{a})\mathbf{U}^{\mathrm{T}}\mathbf{K} \\ &= \mathbf{U}\mathrm{diag}(\mathbf{a})\mathbf{U}^{\mathrm{T}}(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}) \\ &= \mathbf{U}\mathrm{diag}(\mathbf{a})\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}} \end{aligned} \tag{8.1}$$

with

$$\mathbf{A} = \mathbf{U}\mathrm{diag}(\mathbf{a})\mathbf{U}^{\mathrm{T}}. \tag{8.2}$$

Various choices for $\mathbf{a}$ are possible [262, 61, 222], e.g.,

- **I)** Spectrum *flip* takes the absolute of the eigenvalues, i.e., $\tilde{\lambda}_i = |\lambda_i|$. This yields the vector $\mathbf{a}_{\text{flip}} = \mathrm{sign}(\boldsymbol{\lambda})$. In combination with Eq. (8.4) the flip transformation is very similar to the approach for SVMs in Krein spaces as described by Loosli et al. [166].

- **II)** Spectrum *clip* sets all negative eigenvalues to zero, with $\tilde{\lambda}_i = \max(\lambda_i, 0)$ and $\mathbf{a}_{\text{clip}} = \begin{bmatrix}\mathbb{I}(\lambda_1) & \dots & \mathbb{I}(\lambda_n)\end{bmatrix}$, where the indicator function is $\mathbb{I}(\lambda_i) = 1$ if $\lambda_i \geq 0$ else $\mathbb{I}(\lambda_i) = 0$. Spectrum clip is related to the Moore–Penrose pseudoinverse [176], which is sometimes used in case of ill-conditioned $\mathbf{K}$. Spectrum clip can be understood to deal with negative eigenvalues as if they are artifacts caused by measurement noise. Therefore, it is also called the *denoise* spectrum transformation [262].

- **III)** Spectrum *shift* adds a constant to the diagonal of $\mathbf{K}$, i.e., $\tilde{\lambda}_i = \lambda_i + \eta$ with $\eta \in \mathbb{R}^+$ and $\tilde{\mathbf{K}} = \mathbf{K} + \eta\mathbf{I}$. The spectrum shift does not change the differences between consecutive eigenvalues and only acts on the diagonal of the original matrix. Hence, it does not disturb the cross-similarity values.

  Shifting is equivalent to the nugget effect that is often used in the Kriging model (see Section 2.3.4), where η is an additional parameter determined by MLE. Clearly, if η is determined by MLE, spectrum shift does not require the computation of the eigenvalue decomposition. The nugget effect is often used to deal with noise or with ill-conditioned $\mathbf{K}$ [176]. Since noise and ill-conditioning may also occur with indefinite kernels, it may be necessary to use the other spectrum transformations in combination with the nugget effect.

- **IV)** Spectrum *square* takes the square of the eigenvalues, i.e., $\tilde{\lambda}_i = (\lambda_i)^2$ and $\mathbf{a}_{\text{sqr}} = \boldsymbol{\lambda}$. In addition, the spectrum square is equivalent to computing the square of the kernel matrix, that is, $\tilde{\mathbf{K}} = \mathbf{K}\mathbf{K}$. When $\lambda_i \approx 0$, the spectrum square will further reduce the magnitude of λ_i, which may aggravate potential numerical problems.

- **V)** Spectrum *diffusion* takes the exponential of the eigenvalues, with $\tilde{\lambda}_i = \exp(\lambda_i)$ and $\mathbf{a}_{\text{diff}} = \exp(\boldsymbol{\lambda})/\boldsymbol{\lambda}$. Here, $\tilde{\mathbf{K}} = \exp(\mathbf{K})$, where $\exp(\mathbf{K})$ is the matrix exponential function [262]. Spectrum diffusion can produce numerical problems when any $\lambda_i \approx 0$. Spectrum diffusion is closely related to the diffusion kernel [151, 262], which has been proposed to generate PSD kernels for discrete structures, such as graphs.

Of all these transformations, only spectrum shift (cf. nugget effect [93, 176]) and clip (cf. pseudoinverse [176] or multi-dimensional scaling [41]) have been used in Kriging models. They are usually not used to correct definiteness but to deal with noise or ill-conditioning.

Discussion of Extreme Cases Let us now consider how the spectrum transformations behave for two extreme cases.

- Let us assume that a negative definite kernel matrix $\mathbf{K}$ is given, and we would like to produce a PSD kernel matrix $\tilde{\mathbf{K}}$. Of course, this is a trivial case since $\tilde{\mathbf{K}} = -\mathbf{K}$ is PSD. Spectrum clip fails in this case. It sets all eigenvalues to zero. This is clearly undesirable and stresses that spectrum clip should only be used if few eigenvalues are negative. While all other methods provide usable results, spectrum flip is of special interest: Setting all eigenvalues to their absolute value is equivalent to negating the matrix $\mathbf{K}$.

- As a second case, we assume that the matrix $\mathbf{K}$ is already PSD. Both clip and flip do not change the matrix in this case, whereas square and diffusion do. In that sense, clip and flip seem to be preferable if the kernel may occasionally produce definite as well as indefinite kernel matrices.

  Of course, if the original matrix $\mathbf{K}$ is already determined to be PSD, square and diffusion do not need to be applied. Then however, an inconsistency may occur when the matrix is changed only slightly. If a single eigenvalue becomes negative, all positive eigenvalues are changed by spectrum square and diffusion. In the same case, spectrum clip and flip only change the single negative eigenvalue. They behave more consistently when a small change occurs.

Handling New Data Samples To guarantee that new data samples x and training data X are handled consistently (e.g. in the predictor given in Eq. (2.4)) , the transformation $\mathbf{A}$ has to be applied not only to $\mathbf{K}$ but to $\mathbf{k}$ as well [61]:

$$\tilde{\mathbf{k}} = \mathbf{A}\mathbf{k}. \tag{8.3}$$

Equation (8.3) is not required for the spectrum shift, which only affects self-similarities $\mathrm{k}(x,x')$ with $x = x'$. While Eq. (8.3) is a consistent way to treat training samples and new samples [61], its computational effort has been noted as a drawback for prediction [166]. This drawback can be remedied by combining the transformation $\mathbf{A}$ and the resulting inverse of the kernel matrix $\mathbf{K}^{-1}$. The Kriging predictor given in Eq. (2.4) computes $\mathbf{k}^{\mathrm{T}}\mathbf{K}^{-1}$. We can easily show that:

$$\tilde{\mathbf{k}}^{\mathrm{T}}\tilde{\mathbf{K}}^{-1} = (\mathbf{A}\mathbf{k})^{\mathrm{T}}\tilde{\mathbf{K}}^{-1} = \mathbf{k}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\tilde{\mathbf{K}}^{-1}. \tag{8.4}$$

The last equality in Eq. (8.4) is because $\mathbf{A}$ is a symmetric matrix, as can be seen from Eq. (8.2). The computation of $\mathbf{A}^{\mathrm{T}}\tilde{\mathbf{K}}^{-1}$ needs to be performed only once since it is independent of the new sample. Therefore, the computational effort of the Kriging predictor is not increased. With Eqs. (8.1) and (8.4) and $\mathbf{U}^{\mathrm{T}} = \mathbf{U}^{-1}$, we can show that the predictor only requires the untransformed, original kernel:

$$\begin{aligned}\tilde{\mathbf{k}}^{\mathrm{T}}\tilde{\mathbf{K}}^{-1} =& \mathbf{k}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\tilde{\mathbf{K}}^{-1}\\ =&\mathbf{k}^{\mathrm{T}}\mathbf{U}\mathrm{diag}(\mathbf{a})\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathrm{diag}(\begin{bmatrix}\frac{1}{a_1\lambda_1} & \cdots & \frac{1}{a_n\lambda_n}\end{bmatrix})\mathbf{U}^{\mathrm{T}}\\ =&\mathbf{k}^{\mathrm{T}}\mathbf{U}\mathrm{diag}(\begin{bmatrix}\frac{1}{\lambda_1} & \cdots & \frac{1}{\lambda_n}\end{bmatrix})\mathbf{U}^{\mathrm{T}}\\ =&\mathbf{k}^{\mathrm{T}}\mathbf{K}^{-1}.\end{aligned} \tag{8.5}$$

Similarly to the predictor, the uncertainty estimate in Eq. (2.5) computes $\mathbf{k}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{k}$. With Eq. (8.1), Eq. (8.3), and Eq. (8.4) this yields:

$$\tilde{\mathbf{k}}^{\mathrm{T}}\tilde{\mathbf{K}}^{-1}\tilde{\mathbf{k}} = \mathbf{k}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\tilde{\mathbf{K}}^{-1}\mathbf{A}\mathbf{k}. \tag{8.6}$$

Hence, $\mathbf{A}^{\mathrm{T}}\tilde{\mathbf{K}}^{-1}\mathbf{A}$ only needs to be computed once, after training the model. Storing the result requires additional memory. This should usually be negligible since we perform expensive optimization where training data sets can be expected to be rather small. Moreover, the matrix $\mathbf{A}$ itself does not require being stored. In the following, *PSD-correction* refers to all methods that generate PSD matrices via spectrum transformation, with $\tilde{\mathbf{K}} = \mathrm{SPEC}_{\mathrm{PSD}}(\mathbf{K})$.

8.1.2 Spectrum Transformation: Distance

It is straightforward to extend spectrum transformations to distance matrices, by generating NSD matrices with $\tilde{\mathbf{D}} = \mathrm{SPEC}_{\mathrm{NSD}}(\mathbf{D}) = -\mathrm{SPEC}_{\mathrm{PSD}}(-\mathbf{D})$. Unfortunately, Eqs. (8.4) to (8.6) are not applicable in this case. Thus, the computational effort for prediction increases: The transformation $\tilde{\mathbf{d}} = \mathbf{A}\mathbf{d}$ has to be applied to any new data samples and cannot be computed in advance. Fortunately, the effort for MLE decreases in comparison to PSD-correction, since the distance matrix has to be transformed only once (before MLE) and not for every single likelihood calculation.

The NSD-correction of a distance matrix is somewhat excessive because NSDness is not strictly required. A transformation to the broader class of CNSD matrices would be

sufficient. This can be implemented as described by Glunt et al. [102]. They describe an alternating projection algorithm that computes the nearest Euclidean distance matrix (see also Section 8.1.4). For our purpose, we only need a single projection from that algorithm. This projection maps a distance matrix to the nearest CNSD matrix. This highlights a core advantage of CNSD- against NSD-correction: the CNSD-corrected matrix may be closer to the original matrix than the result of the NSD-correction. Following Glunt et al. [102], we compute $\mathbf{Q} = \mathbf{I} - (2\mathbf{e}\mathbf{e}^{\mathrm{T}})/(\mathbf{e}^{\mathrm{T}}\mathbf{e})$ with the identity matrix $\mathbf{I}$ and $\mathbf{e} = \begin{bmatrix}1 & 1 & \dots & 1 & \sqrt{n}\end{bmatrix}^{\mathrm{T}}$. This is used to calculate $\widehat{\mathbf{D}} = \mathbf{Q}(-\mathbf{D})\mathbf{Q}$. Then, the last row and column of $\widehat{\mathbf{D}}$ are removed, which gives $\widehat{\mathbf{D}}_{(-n,-n)}$. Based on this, we can compute the matrix $\check{\mathbf{D}} = \mathrm{SPEC}_{\mathrm{PSD}}(\widehat{\mathbf{D}}_{-n,-n})$. By appending the last row and column of $\widehat{\mathbf{D}}$ to $\check{\mathbf{D}}$, we produce a matrix of the original dimension $n \times n$. Finally, the transformed CNSD distance matrix is $\tilde{\mathbf{D}} = -\mathbf{Q}\check{\mathbf{D}}\mathbf{Q}$.
Due to these more complex computations, $\tilde{\mathbf{d}} = \mathbf{A}\mathbf{d}$ is not applicable to the CNSD case. To handle new samples consistently, a different approach is required. We can use the straightforward approach suggested by Wu et al. [262]. Let the augmented distance matrix $\mathbf{D}_{aug}$ contain the distances between all training and new, unseen samples. Then, CNSD-correction can be applied to the augmented matrix, and the vectors for new samples can be extracted from the result, i.e.,

$$\mathbf{D}_{aug} = \begin{bmatrix}\mathbf{D} & \mathbf{d}\\ \mathbf{d}^{\mathrm{T}} & 0\end{bmatrix} \quad \text{and (after transformation)} \quad \tilde{\mathbf{D}}_{aug} = \begin{bmatrix}\tilde{\mathbf{D}} & \tilde{\mathbf{d}}\\ \tilde{\mathbf{d}}^{\mathrm{T}} & \tilde{d}\end{bmatrix}. \tag{8.7}$$

We denote this approach as re-transformation. The resulting distance vector $\tilde{\mathbf{d}}$ can be used in the Kriging predictor. In the following, spectrum transformations of the distance matrix are denoted with *NSD-* or *CNSD-correction*.

8.1.3 Spectrum Transformations: Condition Repair

The spectrum transformations are clearly useful to produce definite matrices. Unfortunately, the resulting matrices are not guaranteed to fulfill the other conditions required for distance and correlation matrices as described in Chapter 4. That is, the distances or kernel matrices are no longer guaranteed to be non-negative. The diagonal of the distance matrix may become non-zero. The kernel values (correlations) are also no longer guaranteed to be smaller than one.
Consequently, uncertainty estimates for observed samples (training data) can become non-zero. This is undesirable because it may stall the optimization progress since the expected improvement may become non-zero for already evaluated candidate solutions. A related issue with the nugget effect is described by Forrester et al. [93]. We refer to methods that mend this defect as *condition repair* methods.
A PSD correlation matrix can be repaired with [207]

$$\tilde{\mathbf{K}}^{*}_{ij} = \tilde{\mathbf{K}}_{ij}/\mathrm{sqrt}(\tilde{\mathbf{K}}_{ii}\tilde{\mathbf{K}}_{jj}). \tag{8.8}$$

A CNSD distance matrix $\tilde{\mathbf{D}}$ can be repaired with

$$\tilde{\mathbf{D}}^{*}_{ij} = 2\tilde{\mathbf{D}}_{ij} - \tilde{\mathbf{D}}_{ii} - \tilde{\mathbf{D}}_{jj}. \tag{8.9}$$

The result is non-negative, has a zero diagonal, and is CNSD [53].
Since the nugget effect (spectrum shift) only changes the diagonal of $\mathbf{K}$, Eqs. (8.8) and (8.9) are not required in that case. Instead, the uncertainty estimate can be remedied by the re-interpolation approach [93].
Equations (8.8) and (8.9) repair the matrices associated with the training data. Any new data needs to be treated consistently. That means, Eqs. (8.8) and (8.9) need to be applied to new data. That implies, for any new sample $x^{(l=n+1)}$, the transformed self-similarities ($\tilde{\mathbf{K}}_{ll} = \tilde{\mathrm{k}}(x^{(l)}, x^{(l)})$) or self-distances ($\tilde{\mathbf{D}}_{ll} = \tilde{\mathrm{d}}(x^{(l)}, x^{(l)})$) are required. These quantities are not known, because the “transformed” functions $\tilde{\mathrm{k}}(x, x)$ and $\tilde{\mathrm{d}}(x, x)$ are not known explicitly. We suggest three different methods to determine these values.

Repair by Re-transformation Following the idea of Eq. (8.7), the correlation or distance matrices can be computed for the augmented data set (training data and new samples). Afterwards, a spectrum transformation can be computed for matrices that describe the whole data set. Naturally, this yields the necessary $\tilde{\mathbf{K}}_{ll}$ or $\tilde{\mathbf{D}}_{ll}$ for all samples. Inconsistencies of the transformation are a potential problem: The transformation used for the augmented data may differ from the one used during model training.

Repair by Nearest-Neighbor To approximate $\tilde{\mathbf{K}}_{ll}$, we can determine the nearest neighbors of the new sample $x^{(l)}$ in the training data set. That is, we determine those samples $x^{(i)} \in X$ that have the minimal distance $\mathrm{d}(x^{(i)}, x^{(l)})$. The mean of these neighbors’ self-similarity can be used as an approximation of the new samples self-similarity. Distances can be computed accordingly. A drawback of this approach is that the approximated value may jump if a slight change of a candidate results into a different nearest neighbor.

Repair by Linear Combination A smoother approximation can be generated with a linear combination of all self-similarities in the training set. That is

$$\tilde{\mathbf{K}}_{ll} = \sum_{i}^{n} \beta_i \tilde{\mathbf{K}}_{ii}. \tag{8.10}$$

Here, the weights β_i are based on the reciprocal of the distance to the new sample, $q_i = 1/\mathrm{d}(x^{(i)}, x^{(l)})$, with $\beta_i = q_i/(\sum_i^n q_i)$. This ensures that the weights sum to one. If $\mathrm{d}(x^{(i)}, x^{(l)}) = 0$, then the two samples are identical, and $\tilde{\mathbf{K}}_{ll} = \tilde{\mathbf{K}}_{ii}$. Values for $\tilde{\mathbf{D}}_{ll}$ can be computed in the same way.

One drawback of the latter two approximations is their potential inaccuracy. One result of this is, e.g., that the approximated $\tilde{\mathbf{D}}_{ll}$ can occasionally become too large, yielding a negative result in Eq. (8.9). In such cases, the approximated values have to be set to a respective lower or upper bound.

8.1.4 Nearest Matrix Approach

An alternative approach to spectrum transformation is to compute the nearest correlation matrix [120] or nearest Euclidean distance matrix [102]. One approach to compute these nearest matrices is based on alternating projections. Here, two projections are applied sequentially to the indefinite matrix, until a convergence criterion is fulfilled. The result is the nearest matrix (e.g., under F-norm) that fulfills the required conditions. For correlation matrices or Euclidean distance matrices, the alternating projection approach reveals a strong link to the spectrum transformations. In both cases, the first projection is based on the spectrum clip, and the second projection forces all values on the diagonal to one (correlation) or zero (distance). A process that applies both projections alternatingly converges to a correlation (distance) matrix that is PSD (CNSD). The result is the nearest matrix with the desired properties. Further condition repair is not required.

Unfortunately, these methods lack an efficient and consistent way of handling new data samples. Similarly to the condition repair procedures, a re-transformation as described in Eq. (8.7) can be used to derive $\tilde{\mathbf{k}}$ (or $\tilde{\mathbf{d}}$ analogously). However, the nearest matrix computation already involves an optimization process, which requires significantly more computation time than the single-step spectrum transformations. Hence, recomputing the nearest correlation or distance matrix for every prediction would often require an intractably large computation time.

8.1.5 Feature Embedding

In feature embedding [166], indefinite kernel matrices can be interpreted as real-valued data features. Consequently, these features can be used as an input for a standard CNSD distance function or a PSD kernel function. In the CNSD case, this very intuitive, simple approach works as follows.

$$\tilde{\mathbf{D}}_{ij} = \mathrm{d}_{\text{def}}(\mathbf{D}_{i\cdot}, \mathbf{D}_{j\cdot}),$$

where $\mathbf{D}_{i\cdot}$ and $\mathbf{D}_{j\cdot}$ are the i-th and j-th rows of a potentially indefinite matrix $\mathbf{D}$, and $\mathrm{d}_{\text{def}}(x, x')$ is a CNSD distance function. We use the Euclidean distance, but any CNSD distance for real-valued data is appropriate. Then, the Kriging model is trained with $\tilde{\mathbf{D}}$ and $\tilde{\mathbf{K}}_{ij} = \exp(-\theta\tilde{\mathbf{D}}_{ij})$. To be consistent, new data has to be handled with the same feature embedding. That means, distances between training samples and new samples have to be subject to $\tilde{d}_i = \mathrm{d}_{\text{def}}(\mathbf{d}, \mathbf{D}_{i\cdot})$.

8.2 Indefiniteness and Non-Stationarity

A Link to Non-stationarity One issue of the spectrum transformations is that their meaning is not perfectly clear in the context of Gaussian processes. How can the transformations and repair methods be interpreted?

One hint is given by the necessity of the repair procedure from Eq. (8.8). It is identical to the computation that is needed to determine a correlation matrix based on a covariance matrix of a *non-stationary* process. Non-stationary covariances depend on the spatial location. That is, covariances directly depend on the absolute positions x and x', rather than only their relative position via the distance $d(x, x')$. In the stationary case, $\mathbf{C} = \sigma^2\mathbf{K}$ (cf. Eq. (2.2)). For the non-stationary case, this is not true because the variances (diagonal of $\mathbf{C}$) are no longer constant. In other words, we are able to interpret the spectrum transformations and repair methods as a mapping from an indefinite kernel to a definite but non-stationary kernel.

Therefore, we could now employ covariance matrices instead of correlation matrices. This has two repercussions. Firstly, we would need to adapt the likelihood function from Eq. (2.3) slightly so that it is expressed in terms of the covariance matrix. Secondly, PSDness is a sufficient criterion for the validity of a covariance matrix. No upper or lower bounds on the matrix elements have to be satisfied.

The latter suggests that we no longer need a condition repair procedure. Unfortunately, this would not resolve the core issue of the repair procedures: the need to extend the spectrum transformations to the cross-similarities and self-similarities of new data points (cf. Section 8.1.3). This becomes clear when considering the uncertainty estimate. If we express it with respect to the (corrected) covariance matrix, it becomes (cf. Eq. (2.5))

$$\hat{s}^2(x) = \tilde{c}(x, x) - \tilde{\mathbf{c}}^{\mathrm{T}}\tilde{\mathbf{C}}^{-1}\tilde{\mathbf{c}}. \tag{8.11}$$

Here, $\tilde{c}(x, x)$ is an unknown covariance function that includes the spectrum transformation. The covariance vectors and matrices may be computed, e.g., via $\tilde{\mathbf{c}} = \mathbf{A}\mathbf{c}$ and $\tilde{\mathbf{C}} = \mathbf{A}\mathbf{C}$. However, $\tilde{c}(x, x)$ can only be computed as discussed in Section 8.1.3.

A Note on the Nyström Approximation Non-stationarity motivates an alternative approach to compute $\tilde{c}$ (x,x), inspired by spectral tempering as discussed by Pintore and Holmes [201]. They change the eigenvalue spectrum of PSD covariance matrices, with the goal to introduce non-stationarity into the Gaussian process. We have a slightly different point of view: An indefinite covariance function implicitly produces a non-stationary, PSD kernel by spectrum transformation.

Despite this difference, Pintore and Holmes end up with a similar problem: They need to extend the spectrum transformation to new samples for the purpose of prediction. To that end, they make use of approximations like the Nyström method [201]. The Nyström approximation is frequently used in kernel-based machine learning, to provide low-rank approximations of kernel matrices [206, 260]. For any PSD kernel matrix, the Nyström approximation can be written as [260]

$$\mathbf{C}_{n_o,n_o} = \begin{bmatrix} \mathbf{C}_{n,n} & \mathbf{C}_{n,n_o-n} \\ \mathbf{C}_{n_o-n,n} & \mathbf{C}_{n_o-n,n_o-n} \end{bmatrix} \approx \mathbf{C}_{n_o,n}\mathbf{C}_{n,n}^{-1}\mathbf{C}_{n,n_o}.$$

Here, $\mathbf{C}_{n_o,n_o}$ is the full matrix to be approximated with n_o overall samples (new samples and training data), n is the number of training observations (which are the basis of the approximation), $\mathbf{C}_{n,n_o-n} = \mathbf{C}^{\mathrm{T}}_{n_o-n,n}$ are the similarities between training data and new samples, $\mathbf{C}_{n,n_o} = \left[\mathbf{C}_{n,n}\ \mathbf{C}_{n,n_o-n}\right]$, and $\mathbf{C}_{n_o-n,n_o-n}$ are the similarities between the new samples themselves.
In our case, $\mathbf{C}_{n,n}$ and $\mathbf{C}_{n_o-n,n}$ are given via the spectrum transformations. We only need $\mathbf{C}_{n_o-n,n_o-n}$ for the uncertainty estimate in Eq. (8.11),

$$\mathbf{C}_{n_o-n,n_o-n} = \mathbf{C}_{n_o-n,n}\mathbf{C}_{n,n}^{-1}\mathbf{C}_{n,n_o-n}.$$

For a new sample x in our case, this can be written as

$$\tilde{\mathrm{c}}(x,x) = \tilde{\mathbf{c}}\tilde{\mathbf{C}}^{-1}\tilde{\mathbf{c}}, \tag{8.12}$$

where $\tilde{\mathbf{C}}$ is the transformed covariance matrix of the training data, and $\tilde{\mathbf{c}}$ is the vector of transformed cross-covariances. Unfortunately, by substituting Eq. (8.12) into Eq. (8.11), the uncertainty estimate of our Kriging model becomes exactly zero. Hence, the Nyström approximation seems to be an infeasible solution in our context.

8.3 A Simple Example

To provide an example, we used a one-dimensional, continuous test function:

$$f(x) = x^4 - 2x^2 + x$$

with $x \in [-1.5, 1.5]$. The same function has been used, e.g., in Example 2.3.1. While discrete problems are our main motivation, the implications of models with indefinite kernels are easier to understand if they are visualized for a continuous example.
For our example, fifteen uniform random samples were taken from the interval $[-1, 1]$. They were evaluated with the test function. These samples and observations were the training data for the model.
To model the corresponding data, we used the distance

$$\mathrm{d}_{\text{Non-stat}}(x,x') = |x-x'|(|x|^3 + |x'|^3)/2,$$

The distance $\mathrm{d}_{\text{Non-stat}}(x,x')$ is non-CNSD (see Appendix B), and non-stationary. It depends on $|x|$ and $|x'|$, not just on $|x-x'|$. We chose this distance for two reasons. Firstly, this distance measure is indefinite even in a one-dimensional search space. Secondly, it provides a connection to the discussion of non-stationarity in Section 8.2.

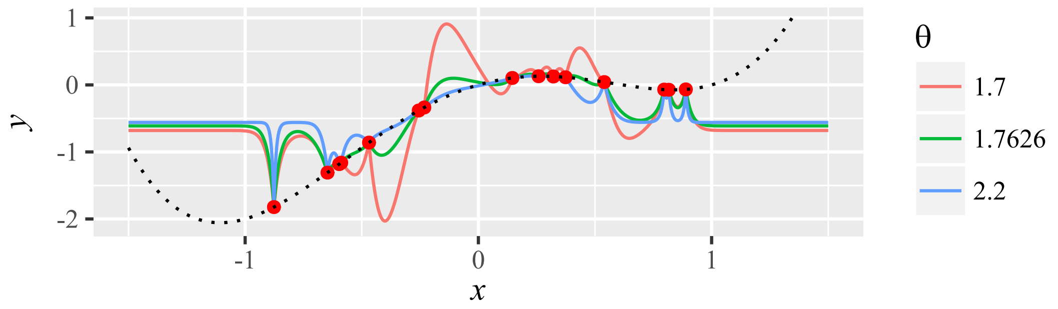


Figure 8.1: *The underlying true function (dotted line), training data (red dots), and three uncorrected Kriging models, each with a different kernel parameter θ, based on the non-CNSD, non-stationary distance described in Section 8.3. The green line ($\theta = 1.7626$) represents the MLE result.*

We used an exponential kernel with $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}_{\text{Non-stat}}(x, x'))$. By MLE, we found that the optimal kernel parameter for the training data was $\theta \approx 1.7626$. Clearly, a non-stationary distance was not a good choice for our stationary test problem. This was confirmed by a visualization, as depicted in Fig. 8.1. The trained model did not represent the true function very well. The poor result was not caused by a failure of the MLE procedure. Other θ values did not improve the model.

One solution to this problem would have been to choose a better-suited distance measure. Under the assumption that we had no other distance measure available, we tested whether the definiteness correction methods were able to improve the model. First, we tested the nugget effect, as shown at the top of Fig. 8.2. The resulting model was better, but left room for further improvement.

Feature embedding in combination with the nugget effect produced a better result, as shown in the center of Fig. 8.2. CNSD correction with spectrum flip and re-transformation yielded a similarly good result (bottom of Fig. 8.2). All other combinations of correction and repair methods are shown in Appendix D, Figs. D.6 to D.9. In all depicted cases, the nugget effect was used. Many produced similarly good results as feature embedding and CNSD correction. Mostly, the spectrum transformations were able to counterbalance the non-stationarity of the underlying distance measure. Still, some configurations performed worse than the uncorrected model.

Based on these preliminary results, we exclude a number of configurations from further investigation: We only investigate models with the nugget effect since it can help to avoid numerical issues. The spectrum square is excluded as it often yielded very poor results, and caused model failures. The reason for these failures is that the spectrum square decreases the magnitude of eigenvalues if $|\lambda| < 1$, while it increases larger eigenvalues. This may frequently produce ill-conditioned matrices.

The nearest-neighbor repair method is also excluded due to inconsistent, discrete jumps in the prediction. Similarly, the nearest Euclidean matrix approach is excluded due to its inconsistent predictions. These inconsistencies are likely to be caused by the fact that training and test samples are not subject to the same transformations. This could

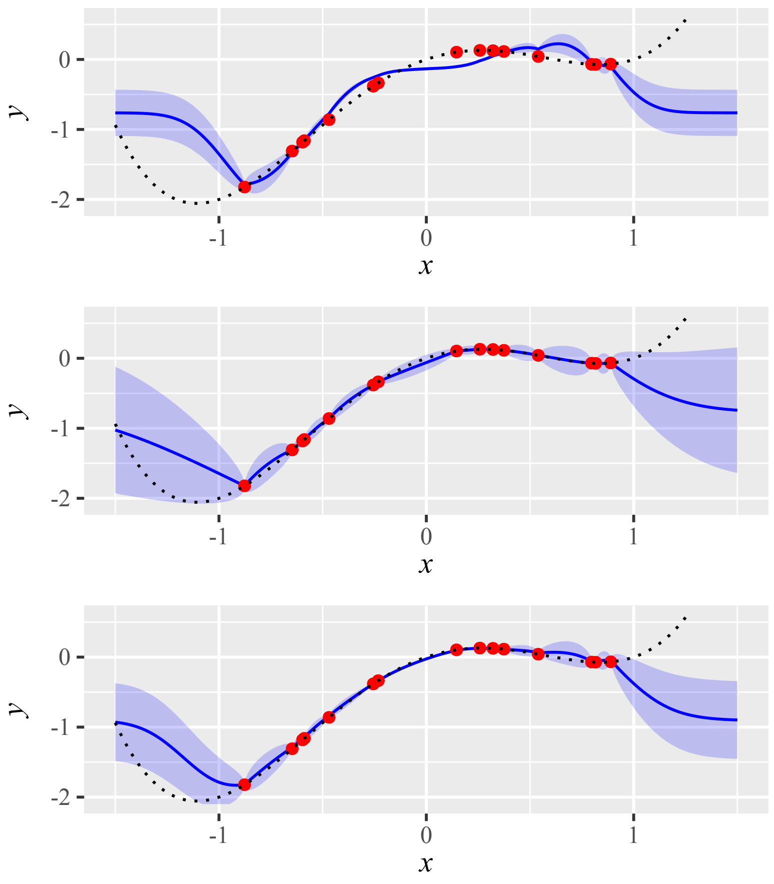


Figure 8.2: *The test function (dotted line), training data (red dots), and a Kriging prediction $\hat{y}(x)$ (blue line). The model uses the non-stationary distance described in Section 8.3. The shaded region indicates the uncertainty estimate, that is, $\hat{y}(x) \pm \hat{s}(x)$. Top: with nugget effect. Center: with feature embedding and nugget effect. Bottom: CNSD-correction, spectrum flip, condition repair with re-transformation and nugget effect.*

be remedied by using a re-transformation approach, but this would lead to a considerably increased computational cost. While the nearest correlation matrix performed somewhat better, we still exclude it, not least because of its computational cost. Finally, PSD-correction in combination with the linear repair method seemed to lead to numerically unstable uncertainty estimates. Therefore, this combination is also excluded.

8.4 Experimental Validation

In the following, a number of more detailed experiments are described. The goal of these experiments was a more extensive comparison in the context of modeling and optimization. Hence, we configured two sets of experiments. Firstly, we tested the model quality, measuring the accuracy of the predictor and the uncertainty estimate. Secondly, the optimization performance was investigated by employing the corresponding models in CEGO.

Test-problems We used two search domains to test our approaches, real vectors and permutations. In both cases, the experiments were performed with simple artificial test functions of the form

$$f(x) = \min_i \mathrm{d}(x, x_{\text{ref}}^{(i)}),$$

where x is a candidate solution (permutation or real vector). The objective function value $f(x)$ was the minimum distance to randomly chosen reference solutions $x_{\text{ref}}^{(i)} \in \mathcal{X}$, with $i = 1, ..., w$. The number of reference solutions w controls the multimodality of the function. In case of $w = 1$, $f(x)$ is unimodal and identical to the UNI problem instances already discussed in Sections 5.3 and 6.3 (cf. [181]). Here, tests were made with $w = 1, 3$ and 5. In addition, different dimensionalities of the search spaces were tested (permutation: $m = 10, 20, 30$, real vector: $m = 3, 5, 8$). The motivations for these test problems were two-fold. Firstly, they have a transparent structure, and their ruggedness can be controlled via w. Secondly, using the same indefinite distance in the model and the test function ensures that observed performances can be attributed to the lack of definiteness and the correction methods. This latter reasoning follows the ideas already outlined in Section 7.4.3.

For the real-valued test problems, we used four different distance measures: the Chebyshev distance, the L0.5 distance, a distance based on the test statistic of the t-test, and the non-stationary distance already used in Section 8.3. For the permutation problems, we selected the Insert distance, the Chebyshev distance, and the L0.5 distance measure. The Chebyshev and L0.5 distances are not very natural measures of distance for permutations. They were chosen to produce test cases that are more pathological. All these distances are non-CNSD. More details on the distances can be found in Appendix B.

Performance measures The accuracy of the models was assessed with the RMSE. To determine the quality of the uncertainty estimate, we computed standardized residuals $(y - \hat{y})/\hat{s}$, cf. [140, 256]. These were used to calculate the Cramèr-von Mises (CVM) test statistic [10], comparing against a normal distribution with zero mean and unit variance. The intent was to use the CVM statistic to measure the quality of the uncertainty estimate. For the modeling experiments, a data set with 100 uniform random samples was generated. The RMSE and CVM values were estimated via 10-fold cross validation.

For the optimization performance, the best values found after 100 objective function evaluations were recorded, except for permutation experiments with $n = 10$, where the best value after 50 evaluations was recorded, due to the smaller search space of these problem instances.

Model and Optimizer settings Most settings of the model and optimizer were taken from Section 5.3.1, with the following exceptions.

- The budget of the optimizer of the surrogate model (a model-free EA) was set to $10,000$. This is the number of EI evaluations in each iteration of CEGO.
- The Cholesky decomposition was used in the likelihood evaluation of the Kriging model. This decomposition may fail to produce results for uncorrected indefinite matrices. In that case, the computation routine for the logarithm of the likelihood was configured to return a penalty of $-1e4 + \lambda_1$. By including the smallest eigenvalue λ_1 into the penalty, the MLE search may be driven into regions with positive λ_1. If no parameterization with positive λ_1 is found, the model training fails.
- The models always used the same distance functions that were employed in the respective test function. This was supposed to mirror the case where an adequate distance is chosen by prior knowledge.
- For the correction of definiteness, we tested PSD-, NSD-, and CNSD-correction. For each of these, we tested transformations with spectrum clip and flip in combination with condition repair methods (none, re-transform, linear). In addition, tests with feature embedding were performed. As a baseline, experiments are performed without any form of correction (except for the likelihood penalty). All correction methods were tested with the nugget effect, except for the baseline (uncorrected), which was tested with and without the nugget effect. Other methods for correction or repair were excluded, as noted at the end of Section 8.3.
- Where the nugget effect was used, η was always added to the diagonal of $\tilde{\mathbf{K}}$, i.e., *after* applying the other correction methods.

8.5 Observations and Discussion

In the following, we summarize the results and analysis of the performed experiments.

8.5.1 Preliminary Observations: Modeling Failures

In some cases, we observed model failures. Here, a failure implies that no numerically tractable models were produced.
For the optimization performance tests, runs were stopped prematurely when a failure occurred. When this happened, the recorded performances were based on the evaluations made before the failure. This penalized model failures because failed runs did not exploit the complete function evaluation budget.
Overall, there were only very few isolated failures. This is mostly because the majority of problematic configurations were excluded based on the results in Section 8.3. For

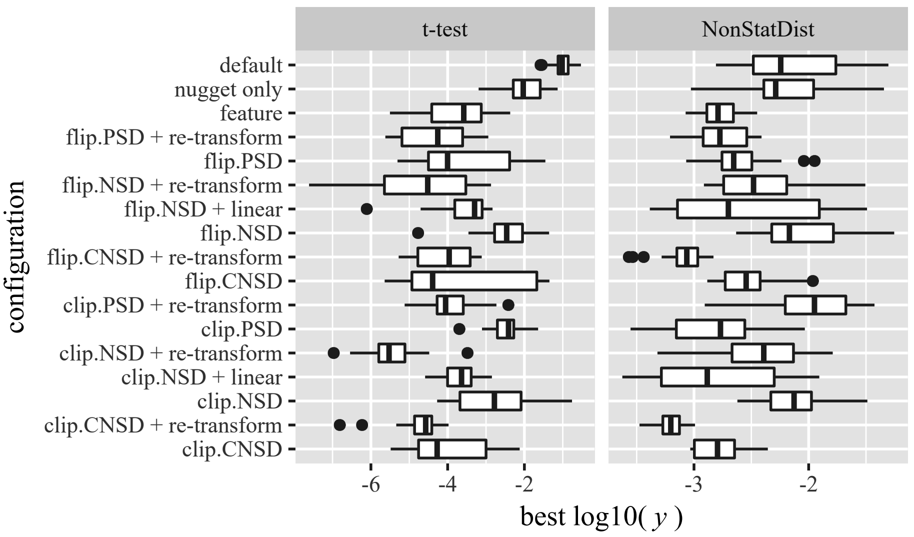


Figure 8.3: *Logarithm of the optimization performance for real-valued problem instances. Lower values are better. The y-axis represents the different definiteness correction methods, where default is a model without any correction. The depicted instances use $m = 5$, and $w = 1$.*

instance, models based on the spectrum square transformation but without the nugget effect would have much larger failure rates.

For the real-valued optimization experiments, 196 of $12,240$ algorithm runs failed. 170 of those were runs without any correction (default model, baseline). The remaining 26 used NSD-correction without condition repair. None of the model quality experiments failed. Neither model quality experiments nor optimization experiments failed for the permutation problems.

8.5.2 Results: Real-valued Problems

Following the procedures from Appendix A, we started with a simple visual analysis. Optimization performance results from two problem instances are shown in Fig. 8.3. Especially for the t-test distance, the correction methods performed quite well. All configurations outperformed the default model without correction. Otherwise, the best performing methods used repair with re-transformation. For both instances, a variant with spectrum clip performed best. The model quality results for the same instances are presented in Fig. 8.4. Two variants with spectrum flip and NSD-correction produced the lowest error. The same configurations performed less well for the optimization experiments. This might be explained by the corresponding CVM values, which showed that the corresponding uncertainty estimates were extremely poor. In fact, there seemed to be a clear conflict between the RMSE and CVM measures. This is fairly obvious, e.g., for the observed results of “flip.NSD”. While it ranks among the best in terms of RMSE, it also ranks worst in terms of CVM. Yet, the RMSE and CVM

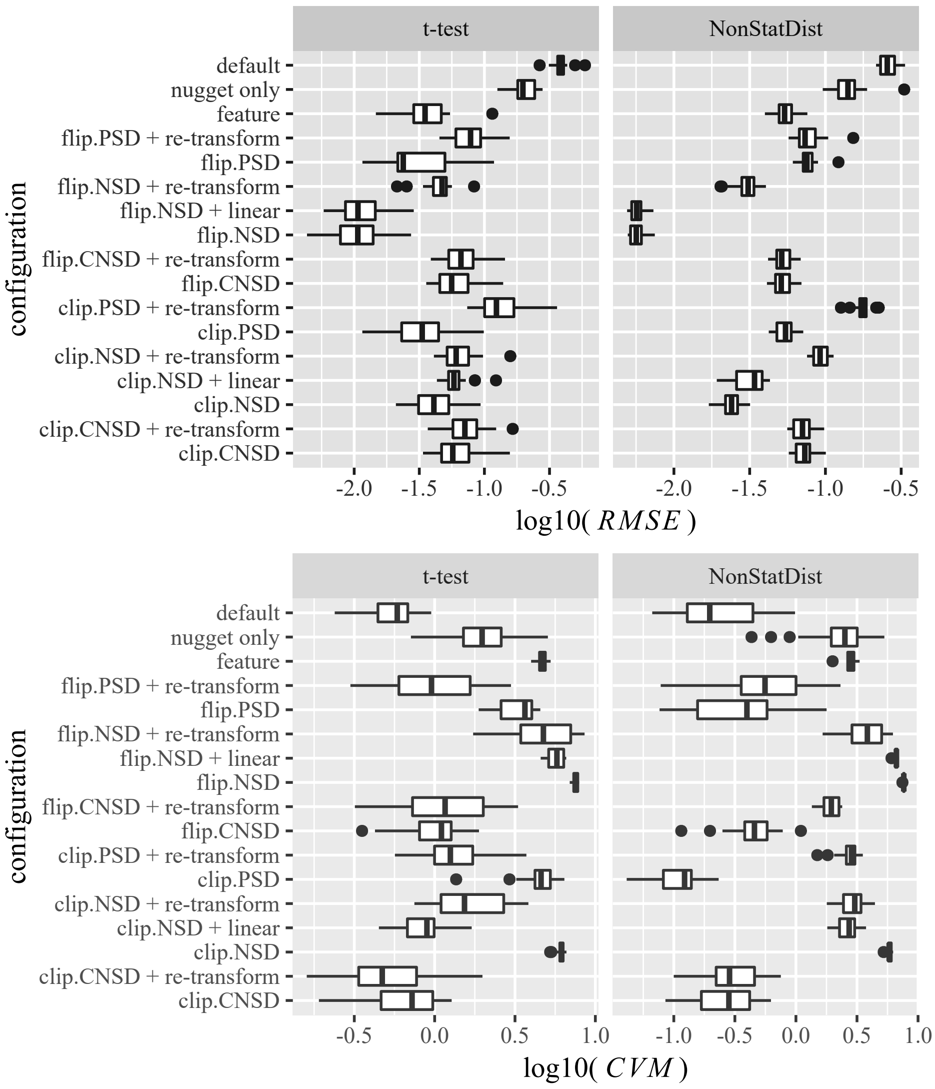


Figure 8.4: *Logarithmic RMSE and CVM values for real-valued problem instances. Lower values are better. The y-axis represents the different definiteness correction methods, where default is a model without any correction. The depicted instances use $m = 5$ and $w = 1$.*

results cannot fully explain the optimization performance results. For instance, the best performing method for the t-test instance with $m = 5$ and $w = 1$ was "clip.NSD+re-transform". This same method was outperformed in terms of RMSE and CVM, e.g., by "clip.CNSD + re-transform". This becomes even more obvious by visualizing the medians of RMSE and CVM together, as shown in Fig. 8.5. The non-dominated configurations on the Pareto front are not necessarily optimal with respect to optimization performance.

The optimization results indicate that the variances of the data differ strongly. Hence, a statistical analysis with parametric models was deemed problematic. A preliminary analysis showed that transformations and a weighted least squares approach could en-

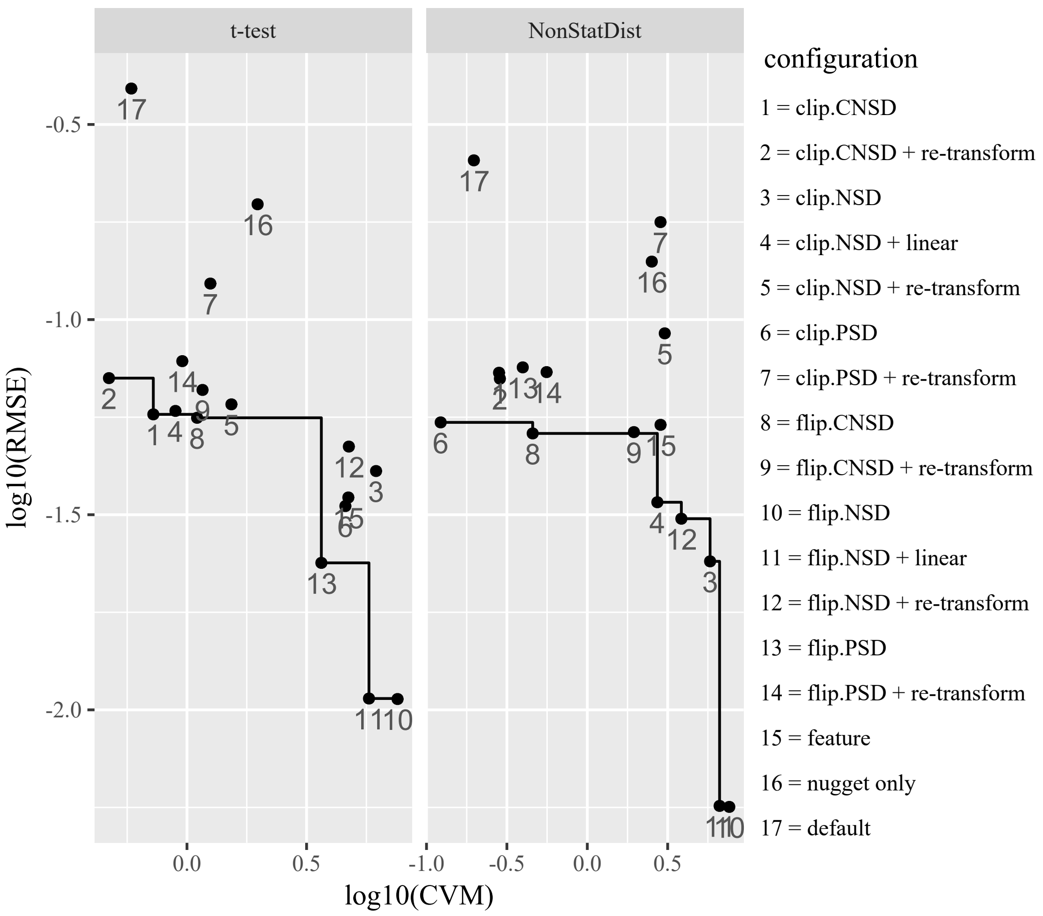


Figure 8.5: *Median of CVM and RMSE values (logarithmic). The solid line is the approximated Pareto front. The problem instances are real-valued with* $m = 5$ *and* $w = 1$.

able a parametric analysis of this data set (cf. Appendix A). Unfortunately, the same did not apply to the permutation experiments. For the sake of consistency, we decided to analyze both cases with non-parametric methods. The same applies to the model quality analysis.

The initial Kruskal-Wallis tests yielded very small p-values ($< 10^{-10}$). Hence, there seemed to be differences between the compared algorithm configurations and a post hoc test was warranted. The ranks resulting from the non-parametric multiple-comparison tests are presented in Table 8.1.

Table 8.1: *Real-valued problems: Ranks based on pairwise comparisons via statistical testing (cf. Appendix A). Green and bold indicates best rank. Each row presents the results of a different test, where "All" indicates a test combining all problem instances. RMSE and CVM are the model quality measures, y indicates the best-observed value from optimization runs. The rows for the individual problem instances only concern optimization performance. Problem instances are denoted as "$m.w$.distance".*

Problem	clip.CNSD	clip.CNSD + re-transform	clip.NSD	clip.NSD + linear	clip.NSD + re-transform	clip.PSD	clip.PSD + re-transform	flip.CNSD	flip.CNSD + re-transform	flip.NSD	flip.NSD + linear	flip.NSD + re-transform	flip.PSD	flip.PSD + re-transform	feature	nugget only	default
All-y	5	2	8	3	2	5	7	4	**1**	9	5	**1**	7	4	6	8	10
All-RMSE	3	4	**1**	2	3	3	6	2	3	2	3	**1**	5	5	2	6	6
All-CVM	3	**1**	7	3	2	5	6	5	2	8	6	3	5	4	4	4	**1**
3.1.L0.5	4	3	3	**1**	3	3	3	**1**	2	4	**1**	**1**	5	2	**1**	5	5
5.1.L0.5	3	3	5	2	3	3	3	2	2	5	2	**1**	4	2	2	4	4
8.1.L0.5	3	3	4	2	3	3	3	2	**1**	5	2	**1**	4	**1**	4	3	5
3.3.L0.5	3	2	4	2	2	3	4	3	**1**	5	3	**1**	5	3	**1**	4	5
5.3.L0.5	2	2	5	2	**1**	4	4	**1**	**1**	5	3	**1**	4	2	3	3	5
8.3.L0.5	2	**1**	3	3	**1**	2	4	**1**	**1**	4	3	**1**	4	3	5	2	5
3.5.L0.5	3	2	4	**1**	2	3	4	2	**1**	5	3	**1**	5	3	**1**	4	5
5.5.L0.5	**1**	**1**	4	2	**1**	3	3	**1**	**1**	4	2	**1**	3	2	3	3	4
8.5.L0.5	2	**1**	3	3	2	3	4	**1**	**1**	4	4	**1**	3	3	5	2	5
3.1.Chebyshev	3	3	3	**1**	3	2	3	2	2	3	**1**	2	3	2	**1**	4	4
5.1.Chebyshev	2	2	4	**1**	2	2	2	3	2	3	**1**	2	3	2	3	4	4
8.1.Chebyshev	2	3	**1**	**1**	2	2	3	3	2	**1**	2	**1**	3	2	2	4	3
3.3.Chebyshev	3	3	4	**1**	2	2	4	2	**1**	4	2	**1**	4	2	**1**	4	4
5.3.Chebyshev	2	**1**	3	2	**1**	**1**	2	**1**	**1**	2	2	**1**	3	2	2	3	3
8.3.Chebyshev	2	2	**1**	**1**	2	2	3	2	**1**	2	2	**1**	2	2	3	2	2
3.5.Chebyshev	3	2	3	**1**	2	2	3	2	**1**	4	2	2	3	2	**1**	3	3
5.5.Chebyshev	2	**1**	2	**1**	**1**	**1**	2	**1**	**1**	2	2	**1**	2	**1**	3	2	2
8.5.Chebyshev	2	2	**1**	**1**	2	2	2	2	**1**	2	2	**1**	3	2	3	2	2
3.1.t-test	2	**1**	3	2	**1**	3	3	2	**1**	3	2	2	3	**1**	2	3	4
5.1.t-test	2	**1**	3	2	**1**	3	2	2	2	3	2	**1**	2	2	2	3	4
8.1.t-test	2	**1**	2	2	**1**	2	3	2	**1**	3	3	**1**	3	2	2	4	4
3.3.t-test	2	**1**	3	2	**1**	3	2	**1**	2	3	2	**1**	2	2	2	3	4
5.3.t-test	2	**1**	3	2	**1**	3	2	2	2	3	3	**1**	2	2	2	4	4
8.3.t-test	**1**	2	2	2	**1**	2	2	**1**	2	2	2	**1**	2	2	2	3	3
3.5.t-test	2	**1**	3	2	**1**	3	3	**1**	2	3	2	2	3	2	2	3	4
5.5.t-test	**1**	**1**	3	2	**1**	3	2	**1**	2	3	2	**1**	**1**	**1**	2	3	4
8.5.t-test	**1**	**1**	2	2	**1**	2	2	**1**	**1**	3	3	**1**	2	2	2	3	4
3.1.NonStatDist	3	**1**	3	**1**	2	3	4	4	**1**	3	**1**	2	3	**1**	2	3	3
5.1.NonStatDist	2	**1**	3	2	3	2	3	2	**1**	3	2	2	2	2	2	3	3
8.1.NonStatDist	**1**	**1**	4	**1**	2	**1**	2	2	**1**	4	**1**	3	**1**	**1**	3	3	2
3.3.NonStatDist	2	**1**	3	**1**	**1**	2	3	3	**1**	3	**1**	**1**	2	**1**	2	3	3
5.3.NonStatDist	2	**1**	4	2	**1**	2	3	3	**1**	4	2	**1**	3	2	3	3	4
8.3.NonStatDist	**1**	**1**	3	**1**	**1**	**1**	2	2	**1**	3	**1**	**1**	2	**1**	3	2	2
3.5.NonStatDist	3	**1**	3	**1**	**1**	2	4	3	**1**	3	**1**	**1**	3	2	2	3	3
5.5.NonStatDist	2	**1**	4	3	**1**	2	4	3	**1**	4	3	**1**	3	2	3	3	4
8.5.NonStatDist	2	**1**	4	3	**1**	2	3	2	**1**	4	3	**1**	2	2	4	3	3

Interestingly, the ranks for model accuracy (RMSE, CVM) and optimization performance (y) often disagreed, especially if RMSE and CVM were considered individually. For a better match with the optimization performance, RMSE and CVM needed to be considered in conjunction. For example, the baseline received a rather poor rank for RMSE, yet a very good rank for the CVM values. This observation can be explained: The baseline did not correct for definiteness (poor RMSE rank), but the uncertainty estimate was not deteriorated by the correction methods (good CVM rank). For the best performing methods, good (but not necessarily best) ranks for CVM and RMSE had to occur simultaneously. This confirmed earlier observations from the visual analysis with Figs. 8.3 and 8.4. It was also confirmed that the default method performed worst. The simple feature embedding approach performed robustly, but it depended on the dimensionality of the search space. It was often among the best performers when the dimensionality of the problem was lowest ($m = 3$). For larger m, this effect disappeared, and feature embedding received worse ranks than the baseline model with nugget effect.
Spectrum transformations often performed very well. Despite this general success, their main drawback was the difficulty of choosing their configuration for: a) condition repair methods, b) different types of spectrum transformation, and c) whether NSD-, CNSD- or PSD-correction should be used:

a) The best performing (y, CVM) configurations used re-transformation. Where applicable, the linear approximation took the middle ground between re-transformation and no repair. The ranking was sometimes reversed for the RMSE results (e.g., for the clip.NSD variants).

b) Spectrum flip performed slightly better than spectrum clip. The good results of spectrum flip are quite promising because spectrum flip is theoretically well-founded in the case of SVMs [166].

c) NSD- and CNSD-correction seemed to outperform PSD-correction. Intuitively, this makes sense: NSD- and CNSD-correction correct the distance matrix, which was the source of the lack of definiteness. If the encompassing kernel function were the source, only PSD-correction would be possible. Despite their similar performance, NSD-correction could be preferred to CNSD-correction due to higher computational effort of the latter. On the other hand, CNSD-correction has promising theoretical properties: The corrected matrix is closer to the original matrix.

8.5.3 Results: Permutation Problems

For the permutation problems, two examples of the results are visualized with box plots in Figs. 8.6 and 8.7. Notably, the model quality was very similar for most methods, especially in case of the problem instance with the Insert distance. The overall best performing methods in terms of optimization performance seemed to be those that only used the nugget effect, and several variants with spectrum clip.

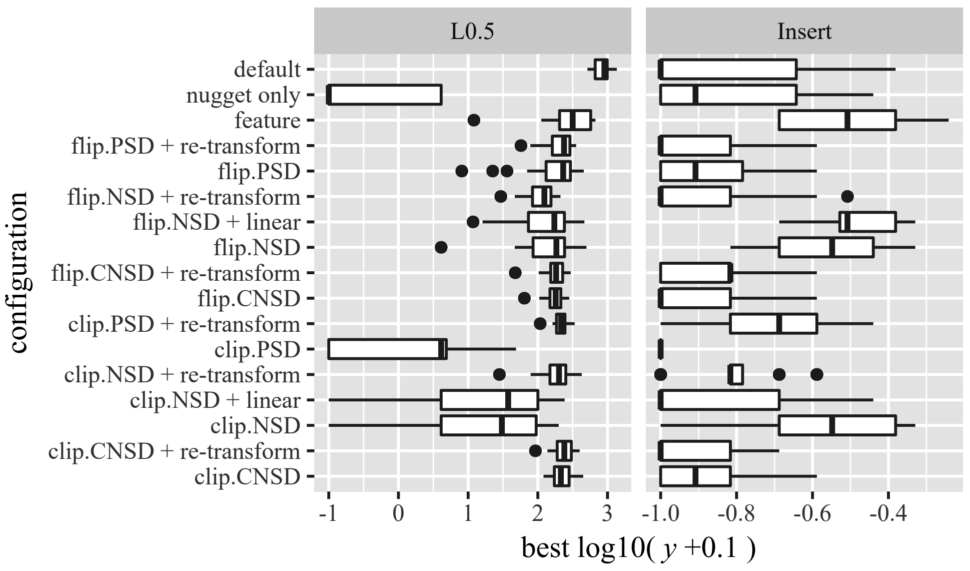


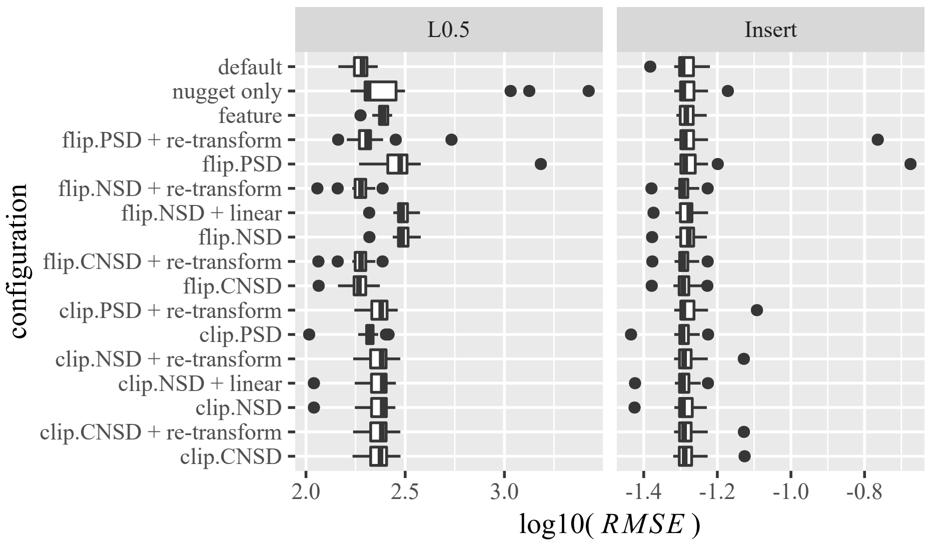
Figure 8.6: *Logarithm of the optimization performance for permutation problem instances. Lower values are better. The y-axis represents the different definiteness correction methods, where default is a model without any correction. The depicted instances use $m = 20$ and $w = 1$.*

As before, we continued with a statistical evaluation. A preliminary analysis indicated that non-parametric tests were warranted. Among other reasons, this was because many ties were present in the data set (see also Appendix A).

The initial tests for the presence of differences indicated p-values of 0.0019 or smaller. Hence, a post hoc multiple comparison test was performed. The resulting ranks shown in Table 8.2 confirmed the visual analysis. Correction with the nugget effect seemed to be sufficient for good performance. Here, the more sophisticated correction methods did not have that much merit. It did not become perfectly clear why the spectrum transformations provided more benefits for the real-valued test problems. It is possible that the optimization problems were too simple to detect any meaningful differences. Creating difficult, but pathological test cases is a challenge. The applications described in Chapter 9 may provide better test cases.

To investigate this further, we made additional experiments with $m = 5$. Here, no differences in optimization performance could be observed at all. These problems were often solved to optimality within as few as 2 iterations of the CEGO algorithm. Essentially, the problem with $m = 5$ was too easy to detect any relevant differences.

The discussed results differ in some cases from previous findings [264]. This may be explained by the differences in the experimental setup. Most results presented here considered higher-dimensional problems. In addition, the experiments we described were conducted with a self-adaptive EA with a five times larger budget of model evaluations. The resulting boost in performance may explain why no (or less) differences are observed in contrast to the previous study.



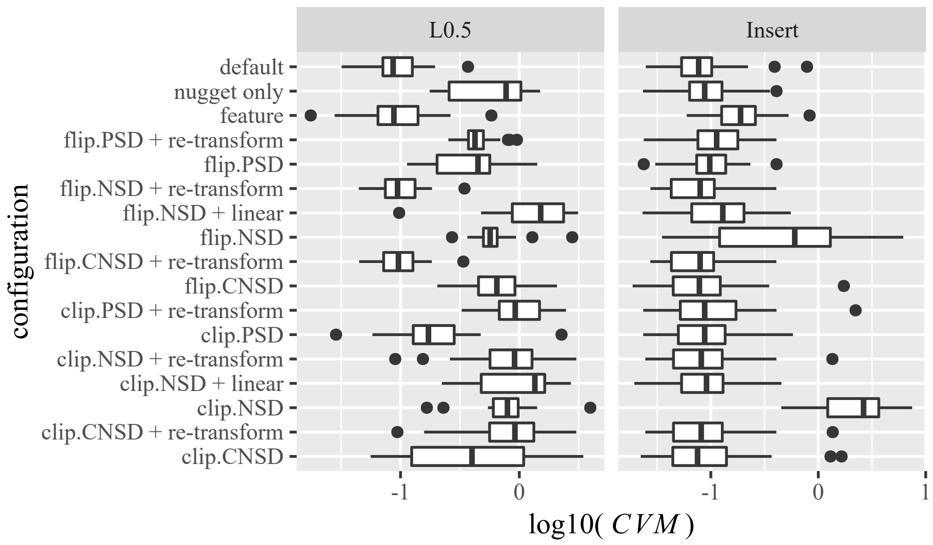


Figure 8.7: *Logarithmic RMSE and CVM values for permutation problem instances. Lower values are better. The y-axis represents the different definiteness correction methods, where default is a model without any correction. The depicted instances use $m = 20$, and $w = 1$.*

Table 8.2: *Permutation problems: Ranks based on pairwise comparisons via statistical testing (cf. Appendix A). Green and bold indicates best rank. Each row presents the results of a different test, where "All" indicates a test combining all problem instances. RMSE and CVM are the model quality measures, y indicates the best-observed value from optimization runs. The rows for the individual problem instances only concern optimization performance. Problem instances are denoted as "$m.w$.distance".*

Problem	clip.CNSD	clip.CNSD + re-transform	clip.NSD	clip.NSD + linear	clip.NSD + re-transform	clip.PSD	clip.PSD + re-transform	flip.CNSD	flip.CNSD + re-transform	flip.NSD	flip.NSD + linear	flip.NSD + re-transform	flip.PSD	flip.PSD + re-transform	feature	nugget only	default
All-y	2	2	5	**1**	2	**1**	3	3	4	6	4	3	4	4	6	**1**	4
All-RMSE	**1**	**1**	**1**	**1**	**1**	**1**	2	**1**	**1**	3	3	**1**	3	2	**1**	2	2
All-CVM	5	3	7	3	3	4	5	6	**1**	5	4	**1**	4	5	2	4	**1**
10.1.L0.5	**1**	**1**	**1**	**1**	**1**	**1**	2	**1**	**1**	**1**	**1**	**1**	2	**1**	2	**1**	3
20.1.L0.5	3	3	**1**	**1**	3	**1**	3	2	2	2	2	2	3	3	3	**1**	4
30.1.L0.5	2	2	2	2	2	**1**	2	3	2	2	2	2	2	2	3	**1**	4
10.3.L0.5	2	2	2	**1**	2	2	**1**	2	2	2	**1**	2	2	2	**1**	**1**	3
20.3.L0.5	**1**	**1**	3	2	**1**	**1**	**1**	2	2	3	3	2	3	2	4	**1**	4
30.3.L0.5	**1**	**1**	3	**1**	**1**	**1**	2	2	3	3	3	2	3	3	4	**1**	4
10.5.L0.5	3	2	2	**1**	2	2	**1**	3	3	2	**1**	3	2	2	**1**	**1**	2
20.5.L0.5	**1**	**1**	3	**1**	**1**	**1**	2	2	2	3	3	2	3	2	3	**1**	4
30.5.L0.5	**1**	**1**	4	2	**1**	**1**	2	3	3	4	3	2	3	3	4	**1**	5
10.1.Insert	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
20.1.Insert	**1**	**1**	3	**1**	2	**1**	2	**1**	2	3	3	**1**	**1**	**1**	3	2	2
30.1.Insert	**1**	**1**	2	2	**1**	**1**	2	**1**	**1**	3	2	**1**	2	2	3	**1**	**1**
10.3.Insert	3	3	3	**1**	3	**1**	**1**	3	3	3	**1**	3	**1**	**1**	2	**1**	**1**
20.3.Insert	**1**	**1**	3	**1**	2	**1**	**1**	**1**	**1**	2	2	2	**1**	**1**	2	**1**	**1**
30.3.Insert	**1**	**1**	2	**1**	**1**	**1**	**1**	**1**	**1**	2	2	**1**	**1**	**1**	2	**1**	**1**
10.5.Insert	2	**1**	2	**1**	2	**1**	**1**	2	2	2	**1**	2	**1**	**1**	**1**	**1**	**1**
20.5.Insert	**1**	**1**	3	**1**	2	**1**	**1**	**1**	**1**	2	2	**1**	**1**	**1**	2	**1**	**1**
30.5.Insert	**1**	**1**	2	**1**	**1**	**1**	**1**	**1**	**1**	2	2	**1**	**1**	**1**	2	**1**	**1**
10.1.Chebyshev	**1**	**1**	2	**1**	**1**	**1**	**1**	2	2	2	2	2	2	2	2	**1**	**1**
20.1.Chebyshev	**1**	**1**	**1**	**1**	2	**1**	2	2	2	3	3	2	3	3	3	2	2
30.1.Chebyshev	**1**	**1**	**1**	**1**	**1**	**1**	2	**1**	2	2	2	**1**	2	2	2	**1**	**1**
10.3.Chebyshev	**1**	**1**	2	**1**	**1**	**1**	**1**	2	2	2	**1**	2	**1**	**1**	2	**1**	**1**
20.3.Chebyshev	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	2	**1**	**1**	2	2	2	**1**	**1**
30.3.Chebyshev	**1**	**1**	2	**1**	**1**	**1**	2	**1**	2	2	2	**1**	2	2	2	**1**	**1**
10.5.Chebyshev	2	**1**	2	**1**	**1**	**1**	**1**	2	2	2	**1**	2	**1**	**1**	2	**1**	**1**
20.5.Chebyshev	**1**	**1**	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	2	2	2	**1**	**1**
30.5.Chebyshev	**1**	**1**	**1**	**1**	**1**	2	2	**1**	**1**	**1**	**1**	**1**	2	2	**1**	**1**	**1**

8.6 Conclusions

Finally, we come back to our initial research question:

Answer 8.1. Indefinite Kriging. How can Kriging deal with indefinite kernels?
We showed that spectrum transformations are a promising approach towards Kriging with indefinite kernels. Overall, it seemed that the exact choice of method does largely depend on the problem characteristics. In some cases, a simple correction via the nugget effect may be sufficient.

While the methods we employed were quite successful for a set of real-valued test problems, they were not that beneficial for permutation problems. Still, methods like spectrum clip or flip remain a promising choice. Spectrum square was clearly undesirable as it frequently produced numerical issues and model failures. Usually, an additional condition repair method for the transformed data was beneficial.
Due to these results, it may be recommendable to make some preliminary benchmarks in practice, before any experiments with the expensive, real-world problem. This may help to identify methods that are adequate for a specific problem. If no preliminary tests are possible, CNSD-correction with spectrum flip and repair via re-transformation may be recommendable. This combination performed rather well in several experiments, and has some desirable theoretical properties (e.g., with respect to nearness of the resulting matrix to the original, indefinite matrix).

Chapter 9

Applications and Extensions

Up to this point, the experiments and examples in this thesis mostly considered real, binary, and permutation spaces. While these provide transparent test cases, it is of interest to examine the behavior of our methods for more complex problems. To that end, we consider the application of our kernel-based modeling and optimization methods to two additional, challenging types of search spaces: trees for symbolic regression (Section 9.1) and conditional variables (Section 9.2). In addition, we suggest a test function generator based on a simulation approach (Section 9.3). This allows future algorithm developments to be tested with benchmarks based on real-world data.

9.1 A Surrogate for Symbolic Regression

One interesting type of solution representation are trees. They pose a particular challenge to modeling and optimization algorithms due to the complex relationship between representation and fitness. Trees are often used in the context of Genetic Programming (GP), which automatically evolves computer programs that aim to solve a certain task [152]. In GP, computer programs are subject to an evolutionary search process, which improves them based on their fitness, i.e., their ability to solve a given problem. Examples for GP tasks are symbolic regression (SR), production scheduling [190], and machine learning in general [90].
For the complex tree shaped structures that occur in GP, it is challenging to select a suitable distance measure and find a feasible modeling approach. We attempt to answer the following research questions.

Section 9.1 is based on the article *"Linear Combination of Distance Measures for Surrogate Models in Genetic Programming"* by Zaefferer et al. [272]. Especially in the problem description, the description of the distances, and the experimental setup, major parts were taken verbatim from the original article. Otherwise, the text was revised and the description of the distances and the analysis were supplemented with a discussion of definiteness. This includes additional experimental results with definiteness correction methods. The analysis was extended, especially with visualizations.

Question 9.1. Tree distances. What distances are available for modeling in tree-spaces, and what properties do they have?

Question 9.2. Performance. What impact do these distances have on the performance of an SMBO algorithm?

To answer these questions, we investigate bi-level optimization problems based on different SR tasks. While these test functions are not that expensive to evaluate (and hence are not a natural use-case for surrogate models), they present a challenging benchmark for the proposed models. This provides some important stepping-stones for the future design of SMBO algorithms with tree-based data and similarly complex search domains. For example, the proposed phenotypic distance may apply (with suitable adaption) to graph optimization problems, which, e.g., occur in the context of neuroevolution [96, 239].

9.1.1 Related Work

We have already mentioned some relevant developments in Chapter 3. Here, we would like to outline how these works relate to our own.

Kattan and Ong [145] describe an approach with two distinct RBFN models (semantic and fitness). The conjunction of both models is used to evolve a subset of the population. They report that their approach outperforms standard GP on three different tasks, including SR.

Hildebrandt and Branke [121] optimize job dispatching rules with support from a surrogate model. Their surrogate model is a nearest neighbor regression model based on a phenotypic distance. This distance evaluates the similarity of rules based on their outcome for a few reference scenarios. They demonstrate that their model allows for a faster evolution of good solutions. This approach is also discussed and extended by Nguyen et al. [191].

Piltát and Neruda [200] propose to identify useful genotypic features of the tree structures. They aim to predict the quality of an individual solely based on these genotypic features, and test their approach with different SR tasks. They conclude that features such as the size of the tree structure, the types of labels, and the numbers of labels in a tree are important.

All these approaches use rather large budgets and mostly employ surrogate models as filters or decision support in an evolutionary algorithm. To the best of our knowledge, only Moraglio and Kattan [180] describe a *sequential* SMBO approach to GP where a very limited number of function evaluations is allowed. They use an RBFN with appropriate distance measures, such as the structural Hamming distance. Their results did not indicate a significant improvement over the use of a model-free optimization approach.

In contrast to these works, we aim to learn Kriging models (following the idea of EGO [140]) and employ them in a sequential SMBO framework with a severely limited number of fitness function evaluations. Our models are based on a linear combination of three diverse distances. Like several of the above-described studies, we use SR as a test case. We want to show that the relation between complex structures and their associated fitness can be learned and exploited for optimization purposes, even with extremely limited evaluation budgets. Although SR is not particularly expensive, we argue that it presents a difficult and challenging test case to investigate whether our proposed models are able to learn the complex interactions of fitness and genotype.

9.1.2 Bi-level Problem Definition

The goal of SR is to find symbolic expressions (formulas) that solve a regression task [152]. In essence, SR searches for a formula that best represents a given data set. GP attempts to solve this problem by using evolutionary techniques.
One frequently chosen representation for symbolic expressions is the tree representation. Each tree consists of internal nodes and external nodes (leaves). The nodes and leaves are labeled. Labels on the internal nodes indicate mathematical operators (e.g., $+, -, *, /$). The labels on the leaves represent input variables and constant coefficients. In the following, $s_i, c_i \in \mathbb{R}$, where c_i is a constant coefficient of the symbolic expression, and s_i is the i-th input variable of the regression problem to be solved. As an example, Fig. 9.1 shows the tree structures of several symbolic expressions.
Our goal is to develop models that learn the relation between discrete tree structures and their fitness. Here, we are not interested in the influence of the constant coefficients. Hence, we suggest a bi-level problem definition and integrate the optimization of the coefficients into the objective function.
In our bi-level definition, the upper level represents the optimization of the discrete tree structure. For each fitness evaluation of the upper level, the lower level optimization problem has to be solved, which comprehends the optimization of the constants. Therefore, the upper level problem is defined by

$$\min_{x} F(x, \mathbf{c}^*) \qquad \text{subject to} \quad \mathbf{c}^* \in \arg\min_{\mathbf{c}} f(x, \mathbf{c}),$$

where x is the tree structure representation, $\mathbf{c} \in \mathbb{R}^{d_\mathbf{c}}$ are the $d_\mathbf{c}$ constant coefficients, and $f(x, \mathbf{c})$ is the lower level objective function. The determined solution of the lower level problem is denoted with $\mathbf{c}^*$. The number of coefficients $d_\mathbf{c}$ depends on x. In extreme cases, the tree x may not contain any constants ($d_\mathbf{c} = 0$), which eliminates the lower level problem. The objective function values are determined with

$$\begin{aligned} F(x, \mathbf{c}^*) &= f(x, \mathbf{c}^*), \\ f(x, \mathbf{c}) &= 1 - |\text{corr}(\hat{y}(x, \mathbf{c}), \mathbf{y})|, \end{aligned} \tag{9.1}$$

where $\hat{y}(x, \mathbf{c})$ denotes the output of the symbolic expression for the problem data set, $\mathbf{y}$ is the corresponding vector of true observations, and corr$(\cdot, \cdot)$ is the Pearson correlation coefficient. If $\hat{y}(x, \mathbf{c})$ becomes infeasible (e.g., due to a negative square root or division by zero), we assign a penalty value of one (the upper bound of our fitness function). In the following, when we discuss fitness evaluations, we will usually refer to evaluations of the upper level objective function.

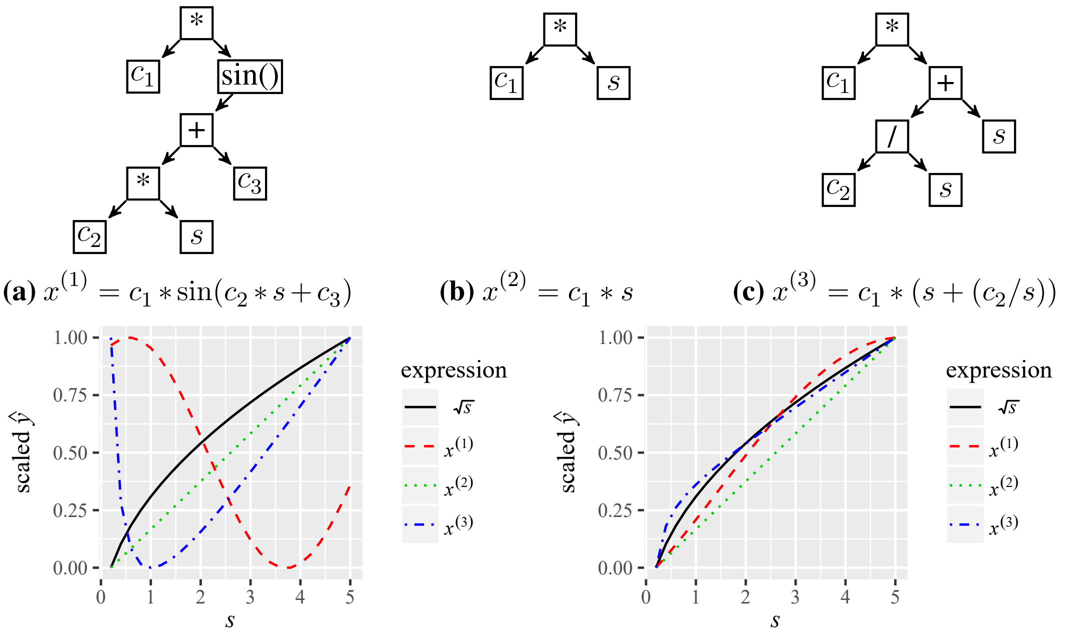
Example 9.1.1. To provide an example, let us assume we want to find a symbolic expression that approximates $y = \sqrt{s}$. An upper level optimization process (e.g., an EA, or CEGO), suggests three symbolic expressions. These expressions are

$$\begin{aligned} x^{(1)} &= c_1 * \sin(c_2 * s + c_3) \\ x^{(2)} &= c_1 * s \\ x^{(3)} &= c_1 * (s + (c_2/s)) \end{aligned} .$$

The tree representations of these expressions are shown at the top of Fig. 9.1. Their approximate solutions to the lower level problem (derived by global optimization with DIRECT [139]) are

$$\begin{aligned} x^{(1)} &= -0.67 * \sin(-0.43 * s + 0.63) \\ x^{(2)} &= 5.68 * s \\ x^{(3)} &= 6.58 * (s + (-0.41/s)) \end{aligned} .$$

The output of the three expressions, with default coefficients ($c_i = 1$) and with optimized coefficients, are shown at the bottom of Fig. 9.1. Here, $x^{(3)}$ receives the best upper level objective function value because its lower-level solution most strongly correlates with $y = \sqrt{x}$.



(a) $x^{(1)} = c_1 * \sin(c_2 * s + c_3)$ **(b)** $x^{(2)} = c_1 * s$ **(c)** $x^{(3)} = c_1 * (s + (c_2/s))$

(d) *default coefficients,* $c_i = 1$ **(e)** *coefficients* c_i *are optimized*

Figure 9.1: *Top: Three symbolic expressions in tree representation. The numbers before the colon are node IDs. Bottom: Outputs of the expressions and the function they intend to approximate, before and after optimization of the coefficients. Since we are interested in the correlation between the expressions, the depicted y-values of each curve are scaled to* $[0, 1]$.

9.1.3 Kernels for Bi-level Symbolic Regression

To learn Kriging models for the SR problem, we use the exponential kernel $\mathrm{k}(x, x') = \exp\left(-\theta \mathrm{d}(x, x')\right)$. In the following, we introduce several choices for $\mathrm{d}(x, x')$.

Phenotypic Distance The Phenotypic Distance (PhD) estimates the dissimilarity of two individuals (trees) based on their program output / phenotype, instead of using their code / genotype. This idea has been suggested by Hildebrandt and Branke for evolving dispatching rules via GP [121]. They defined a phenotypic dissimilarity by comparing the outcome of decision rules based on a small set of test situations.
Our SR tasks require a different definition of the phenotypic distance. We propose to measure the correlation between the outcomes of two symbolic expressions, with all numeric constants set to one. Hence, we save the effort of the optimization of the constants and compare the outputs of the expressions with

$$\mathrm{d}_{\text{PhD}}(x, x') = 1 - |\text{corr}(\hat{y}(x, \mathbf{1}), \hat{y}(x', \mathbf{1}))|. \tag{9.2}$$

If either x or x' are infeasible (e.g., due to division by zero), the distance is set to its upper bound of one. Essentially, our PhD variant sees the similarity of the curves shown in Fig. 9.1d and a model based on the PhD attempts to match these similarities to the curves in Fig. 9.1e.
Setting all coefficients to one is of course arbitrary. A random sample would also be possible but potentially problematic. A difference in phenotype could be perceived due to a different assignment of the constants on the leaves, rather than an actually different behavior of the symbolic expressions.
Instead of this definition (or in addition to it) we could also limit the number of training data samples used to evaluate the symbolic expressions. This would be closer to the approach of Hildebrandt and Branke [121]. Since our use cases involve rather small data sets, this would lead to a negligible reduction in computational cost. Nevertheless, it may be a promising idea for data sets that are more complex.
Note that the PhD is not necessarily a distance on trees. In the form it is used here, it is a distance on the output of symbolic expressions, which may be represented by trees.

Proposition 1. *The phenotypic distance $d_{PhD}(x, x')$ is non-CNSD.*

Proof. To prove this by example, we test the following four symbolic expressions with the values $s \in \{0, 0.01, \ldots, 0.99, 1\}$, compute the corresponding outputs y and receive the phenotypic distance matrix via their correlations:

$$\begin{aligned} x^{(1)} &= s^3 - s \\ x^{(2)} &= s^2 \\ x^{(3)} &= s \\ x^{(4)} &= s - s^2 \end{aligned} \quad \text{with } \mathbf{D}_{\text{PhD}} \approx \begin{bmatrix} 0.00 & 0.99 & 0.76 & 0.04 \\ 0.99 & 0.00 & 0.03 & 0.75 \\ 0.76 & 0.03 & 0.00 & 1.00 \\ 0.04 & 0.75 & 1.00 & 0.00 \end{bmatrix}$$

This matrix is not CNSD, because the critical eigenvalue of $\hat{\mathbf{D}}_{\text{PhD}}$ is $\hat{\lambda} \approx 0.12 > 0$ (cf. Section 4.4.1). □

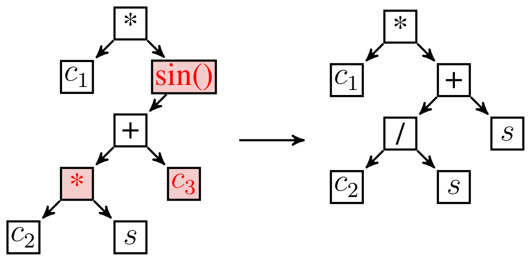


Figure 9.2: *Example of how two trees can be transformed into each other via three edit operations (indicated by red color). From left to right: delete node "sin()", relabel "*" to "/", and relabel "c_3" to "s"*

Tree Edit Distance As an alternative to the PhD, we also employ genotypic distances, i.e., distances between the trees that represent the symbolic expressions. One possible definition of distance between trees is the minimal number of edit operations required to transform one tree into another. This approach is denoted as the Tree Edit Distance (TED). We use the TED implementation that was introduced by Pawlik and Augsten [196]. It is available in the APTED library version 0.1.1 [195]. The APTED implementation counts the following edit operations: node deletion, node insertion, and node relabeling. As demonstrated in Section 7.4.4, the TED is not CNSD.
To provide an example, the TED between the trees $x^{(1)}$ and $x^{(3)}$ from Example 9.1.1 is three, because three edit operations are required to transform either of them into the other. This is illustrated in Fig. 9.2.

Structural Hamming Distance The Structural Hamming Distance (SHD) [182] has been used to express genotypic dissimilarity for model-based GP in several studies [180, 145, 121]. Roughly speaking, it compares two trees by recursively checking each node that the two trees have in common. To compare nodes, it uses the Hamming distance (denoted HD$(x[0], x'[0])$), which is one if the two node labels are different and zero otherwise. The original SHD (SHD1) is defined as

$$\mathrm{d}_{\mathrm{SHD1}}(x, x') = \begin{cases} 1, & \text{if } \mathrm{arity}(x[0]) \neq \mathrm{arity}(x'[0]) \\ \mathrm{HD}(x[0], x'[0]), & \text{if } \mathrm{arity}(x[0]) = \mathrm{arity}(x'[0]) = 0 \\ \Delta(x, x'), & \text{if } \mathrm{arity}(x[0]) = \mathrm{arity}(x'[0]) = n_{\mathrm{b}}, \end{cases}$$

with

$$\Delta(x, x') = \frac{1}{n_{\mathrm{b}} + 1} \left(\mathrm{HD}(x[0], x'[0]) + \sum_{i=1}^{n_{\mathrm{b}}} \mathrm{d}_{\mathrm{SHD1}}(x[i], x'[i]) \right). \qquad (9.3)$$

Here, x and x' are trees, $x[0]$ indicates a root node of x, $x[i]$ with $i \geq 1$ is the i-th subtree of the root node, and arity$(x[0])$ implies the number of subtrees linked to the corresponding node. The SHD is illustrated in Fig. 9.3, using $x^{(1)}$ and $x^{(3)}$ from Example 9.1.1.
We use a slight variation, which we refer to as SHD2. For the sake of simplicity, we give a definition for trees with a maximum arity of two, that is, $n_{\mathrm{b}} \leq 2$. SHD1 and

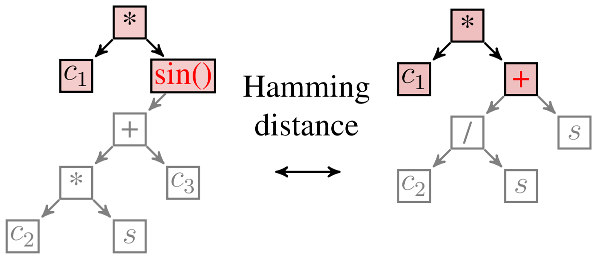


Figure 9.3: *Rough illustration of the structural Hamming distance between trees: The Hamming distance on the nodes of the matching structure. Non-matching structure is grayed out. Red font indicates a difference in the relevant nodes.*

SHD2 are identical, except for the case arity$(x[0])$ = arity$(x'[0]) = n_b > 1$. Then, Eq. (9.3) becomes

$$\Delta(x,x') = \frac{1}{n_b+1}\Big(\mathrm{HD}(x[0],x'[0]) +$$
$$\min\big(\mathrm{d}_{\mathrm{SHD2}}(x[1],x'[1]) + \mathrm{d}_{\mathrm{SHD2}}(x[2],x'[2]), \mathrm{d}_{\mathrm{SHD2}}(x[1],x'[2]) + \mathrm{d}_{\mathrm{SHD2}}(x[2],x'[1])\big)\Big).$$

That means, when two subtrees $x[1]$, $x[2]$ are compared with their counterparts $x'[1]$, $x'[2]$, we use the pairing or alignment between x and x' that yields the smaller distance. Potentially, this is more accurate, since it does not depend on the (arbitrary) initial alignment of the two trees. For instance, consider the two expressions $x = s + 1$ and $x' = 1 + s$, where the left part of each expression is correspondingly placed on the left branch in the tree representation. Under SHD1, both expressions would have a non-zero distance, whereas SHD2 yields exactly zero. The latter is clearly more appropriate because the actual expressions are identical. Unfortunately, SHD2 requires additional computational effort, even more so for larger arities.

Proposition 2. *The second variant of the structural Hamming distance $d_{SHD2}(x,x')$ is non-CNSD.*

Proof. To prove this by example, we consider the following samples, where the brackets imply the structure of the underlying tree, and the operator in the middle represents the root node. The expressions are

$$\begin{array}{rcccc} x^{(1)} &=& (s/s) &+& (s+s) \\ x^{(2)} &=& (s/s) &+& (1+s) \\ x^{(3)} &=& (s+s) &*& (s*s) \\ x^{(4)} &=& (1+s) &*& (s*s) \\ x^{(5)} &=& (1/s) &*& (s-s) \end{array} \quad \text{with } \mathbf{D}_{\mathrm{SHD2}} = \frac{1}{9}\begin{bmatrix} 0 & 1 & 4 & 5 & 5 \\ 1 & 0 & 5 & 4 & 5 \\ 4 & 5 & 0 & 1 & 3 \\ 5 & 4 & 1 & 0 & 2 \\ 5 & 5 & 3 & 2 & 0 \end{bmatrix}.$$

This matrix is not CNSD as the critical eigenvalue of $\hat{\mathbf{D}}_{\mathrm{SHD2}}$ is $\hat{\lambda} \approx 0.0062 > 0$ (cf. Section 4.4.1). □

Linear Combination The introduced distances for SR trees are fairly diverse. Especially the genotypic and phenotypic measures contribute quite different information. Hence, we decided to combine them. We propose a linear combination of PhD, TED, and SHD2, using MLE as discussed in Section 6.2. We decided to focus on one of the SHD variants due to their similarity and chose the SHD2 variant due to its potentially increased accuracy. The increased computational cost of SHD2 is irrelevant compared to the significantly larger costs of the TED. Hence, the linear combination is

$$\mathrm{k}(x, x') = \exp\left(-\beta_1 \mathrm{d}_{\mathrm{SHD2}}(x, x') - \beta_2 \mathrm{d}_{\mathrm{PhD}}(x, x') - \beta_3 \mathrm{d}_{\mathrm{TED}}(x, x')\right). \tag{9.4}$$

Each distance receives a weight $\beta_i \in \mathbb{R}^+$ that is determined by MLE.

Comparison of the Distances To get a rough idea of the differences in behavior, we made some preliminary tests. We first calculated the distance matrices for 100 randomly generated trees (symbolic expressions). The trees are generated as described in Section 9.1.4 for the Kotanchek2D problem. We computed the Pearson correlation between the different distance matrices. For this example, the SHD variants yielded a strong correlation of 0.99. For the rest, the correlation were 0.51 (PhD, SHD2), 0.29 (PhD, TED), and 0.37 (TED, SHD2). The two SHD variants seemed to behave very similar. Of the rest, PhD and SHD2 had the highest correlation. The weak correlation of TED and SHD may seem surprising because they both consider the genotype. At least in parts, this is due to their strongly different scaling. In addition, SHD may give more weight to structural differences close to the root node.
This point becomes more obvious from Fig. 9.4, which visualizes the corresponding distance matrices. Here, the trees are sorted based on their complexity (tree depth and number of nodes). It shows that the SHD does not distinguish between trees of different complexity very well. Several large blocks of the SHD matrices have a value of one, indicating that the respective trees are at maximum distance. This lack of perceiving a more fine-grained difference is problematic. It implies that any model based on SHD is potentially inaccurate for trees of unobserved complexity. The TED and PhD tend to see larger distances for more complex trees. This is obvious for TED: Complex trees require more operations to be transformed into each other. The weaker effect in case of the PhD can also be explained: Complex trees may produce phenotypic behavior that is more diverse, but the probability to produce redundancies also rises.
With regard to the computational effort, we note that TED is by far the most expensive measure. It is followed by the PhD, and the cheapest measure is SHD1. While the specifics strongly depend on the implementation, we note that the TED required at least an order of magnitude more computation time than the other distance measures.

9.1.4 Case Study

Experimental Setup *Symbolic Regression Test Problems:* As test cases, we selected six SR problems. We chose the Newton, sine-cosine, Kotanchek2D, and Salustowicz1D problems as used by Flasch [90] and the sqr and sqr+log problem as used by

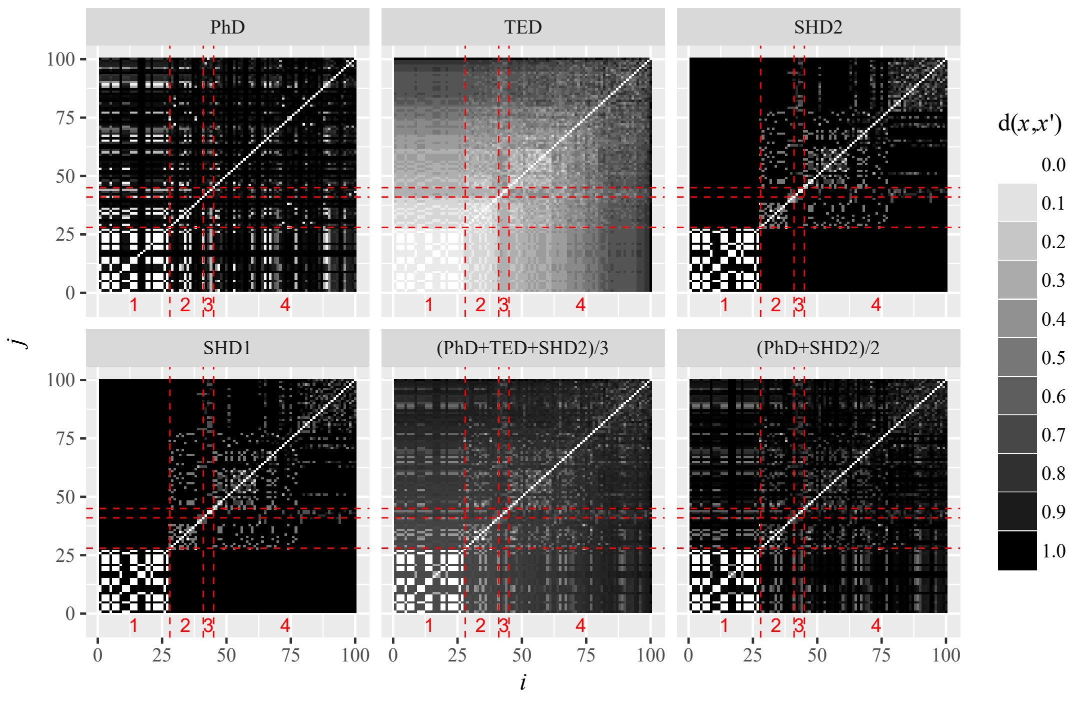


Figure 9.4: *Image plot of the different tree distance measures. Each image cell is the cell of a distance matrix. The trees are sorted by their complexity (tree depth and number of nodes). Trees in the lower left corner are less complex than those in the upper right. The plots in the lower right and lower center depict equally-weighted combinations of the corresponding distances. The tree depth is annotated in red at the bottom of each plot. Extended from a figure in [272].*

Kattan and Ong [145]. All problem configurations remained unchanged, i.e., operator set, data set size, and bounds for variables were chosen as defined in the cited works. We did not evaluate the derived symbolic expressions on an additional test set. This was due to the goal of this case study. We wanted to determine how well we could learn and then exploit the relation between trees, symbolic expressions, and the respective fitness. The generalizability of each individual fitness evaluation was not important for that purpose.

Lower level optimization of the constants: To optimize the lower level objective function, we selected the locally biased version of the DIRECT algorithm [94, 139] for a global search. DIRECT used $1000 \times d_\mathrm{c}$ evaluations of the objective function. The results of DIRECT were further refined with a Nelder-Mead local search [189] (also $1000 \times d_\mathrm{c}$ evaluations).

Upper level optimization of the structure: All algorithms received a budget of 100 upper-level objective function evaluations to emulate an expensive optimization problem. We used Random Search (RS) and a model-free EA as baselines (cf. Section 2.1). The corresponding operators were taken from the `rgp` package [91].

The RS and the EA created new individuals via `randfuncRampedHalfAndHalf`,

parameterized with a maximum tree depth of 4 and a probability to generate constants of 0.2. This procedure creates expressions with maximum tree depth half of the time. Else, the tree is randomly grown, which implies that each new node has a probability of 0.5 to become a terminal node (leaf).
Furthermore, the EA employed `crossoverexprFast` for recombination, which randomly exchanges subtrees. For mutation, `mutateSubtreeFast` was used. The parameters of the mutation operator were specified as follows: 0.1 (probability to insert a subtree), 0.1 (probability to delete a subtree), 0.1 (probability of creating a subtree instead of a leaf), 0.2 (constant generation probability), and 4 (maximum tree depth).
Since constant coefficients were not considered at the upper level, the respective bounds in the operator were both set to one. That is, a “1” is used as a placeholder for the constant coefficients. This is convenient because a value of one is also used in the computation of the PhD.
We employed a standard EA (based on `optimEA` in the `CEGO` package [263]) that used the above-described operators. The EA used truncation selection. A fixed number of children was created in each generation. The population size and the number of children were tuned (see Section 9.1.4).
In addition to the model-free baselines, the upper level problem was also solved by CEGO (see Algorithm 4). Four different variants were tested: One with each of the single distance measures (TED, PhD, SHD2) and one variant with the combined kernel given in Eq. (9.4). The model was trained within $1,000$ likelihood evaluations (via DIRECT). An EA was used to search the surrogate model with $10,000$ evaluations of the expected improvement criterion in each iteration. CEGO was initialized with 20 random trees that are created with `randfuncRampedHalfAndHalf`.
For the analysis, we recorded the best individual for each run. In addition, we recorded the weights used for the linear combination of the distances in each iteration, to evaluate the contribution of each distance function over the runtime. Each algorithm run was repeated 20 times.

Algorithm Tuning We decided to tune some potentially sensitive parameters to allow for a more fair comparison between the model-based and model-free algorithm. The model-free algorithm's population size n_{pop} and number of children n_{off} produced in each iteration were tuned. All combinations of $n_{\text{pop}} \in \{5, 10, 15, 20\}$ and $n_{\text{off}} \in \{1, 2, 3, 4, 5\}$ were tested. The optimization performance was expected to be sensitive to these parameters, due to the extremely small fitness evaluation budget.
For CEGO, we did not tune n_{pop} and n_{off}. Due to CEGO's larger complexity, we decided to set the parameters based on experience only, without a detailed tuning. In fact, due to the larger number of evaluations (of the surrogate model), the algorithm should be less sensitive to n_{pop} and related parameters (cf. the results in Section 5.3). Since $10,000$ evaluations of the surrogate model were allowed, a relatively large $n_{\text{pop}} = 200$ was given to the respective EA, and $n_{\text{off}} = 10$. Instead of these parameters, the comparison of the different distances and kernels may be viewed as the tuning effort invested for CEGO.

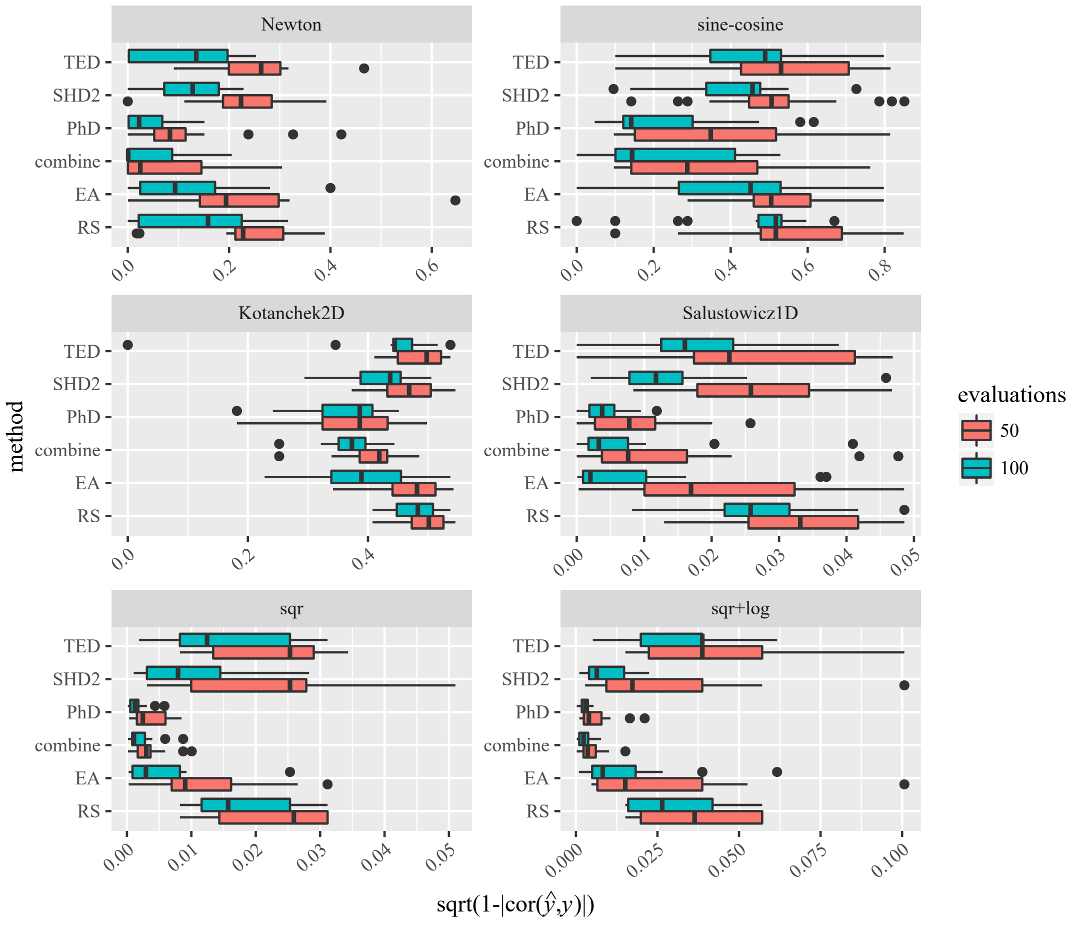


Figure 9.5: *Box plot of best-observed values after 50 and 100 evaluations. The performances of CEGO with single distances are denoted by the respective abbreviations. The linear combination is denoted by "combine". Potentially indefinite distances are not explicitly corrected. The depicted data has been subject to a square-root transformation for the sake of readability.*

We also performed preliminary experiments with the Mean Square Error (MSE) instead of the correlation-based measurements in Eqs. (9.1) and (9.2). The MSE-based experiments yielded rather poor results with CEGO. This may partly be explained by the penalty for infeasible candidates. The penalty value is very difficult to set for the MSE case. A poor choice may severely impair the ability to train a good Kriging model because of strong jumps or plateaus in the fitness landscape. Overall, the behavior of the correlation-based measures seemed to be more benign. While our preliminary experiments were not very detailed, they can be counted as additional tuning effort. They informed the choice of the correlation measure used in the phenotypic distance.

Analysis and Discussion Box plots of the best-observed fitness after 50 and 100 evaluations are shown in Fig. 9.5. The reported results concern the tuned, model-free EA that achieved the best mean rank on all problems ($n_{\text{pop}} = 15$, $n_{\text{off}} = 1$).

Table 9.1: *Pairwise statistical comparison of the performance on all SR instances, where a 1 indicates that the algorithm of the corresponding row is better than the one in the corresponding column, while -1 indicates that the row-algorithm is worse. A zero indicates no significant differences. Here, the non-CNSD distances were not corrected. The last column provides a summarizing rank for the corresponding algorithms in each row.*

Algorithm	RS	EA	combine	PhD	SHD2	TED	rank
RS	0	-1	-1	-1	-1	0	4
EA	1	0	-1	-1	1	1	2
combine	1	1	0	0	1	1	1
PhD	1	1	0	0	1	1	1
SHD2	1	-1	-1	-1	0	1	3
TED	0	-1	-1	-1	-1	0	4

The minimal n_{off} makes sense because it allows for many iterations despite the small budget.
It seemed that the CEGO variants based on the PhD and the linear combination outperformed their competitors. TED and SHD on the other hand did not perform well. They were sometimes even worse than the model-free EA. Mostly, this behavior was consistently observed for results after 50 and 100 function evaluations.
Following the descriptions in Appendix A, we supplemented the visual analysis with the non-parametric Friedman test and the Conover post hoc test. We tested for differences in performance after 100 objective function evaluations, using each SR problem instance as a block. The initial Friedman test produced a p-value close to zero ($< 10^{-16}$). The result of a pairwise comparison via the Conover post hoc test with a significance level of $\alpha = 0.05$ is shown in Table 9.1. This confirmed the conclusions from the visual analysis: The CEGO variants with PhD or with a linear combination of the distances outperformed all other configurations. This is consistent with previous results by Hildebrandt and Branke [121], who report that a similar phenotypic distance achieved better results than SHD1.
Given these results, it could be assumed that the PhD contributed most strongly to the favorable result of the linear combination. This assumption seemed to be true in the beginning of each run, but Fig. 9.6 shows a very strong drift towards the two genotypic distances later on. What is the reason for this behavior? One hint is given in Fig. 9.7. That figure shows LOESS fits of the algorithm performances[1]. Importantly, the model with the PhD started to give deteriorating suggestions after roughly 35 function evaluations. The same deterioration was not observed for the other approaches. The linear combination seemed to avoid this issue by shifting to the genotypic distances.
One potential reason for this behavior is the definiteness of the distances. All three distances are non-CNSD. As discussed in Chapter 7, this may not necessarily have the same effect for every distance. Hence, we presumed that the PhD produced poor results because it was more strongly affected by its lack of definiteness.

[1]LOESS models are locally weighted polynomial regression models [64].

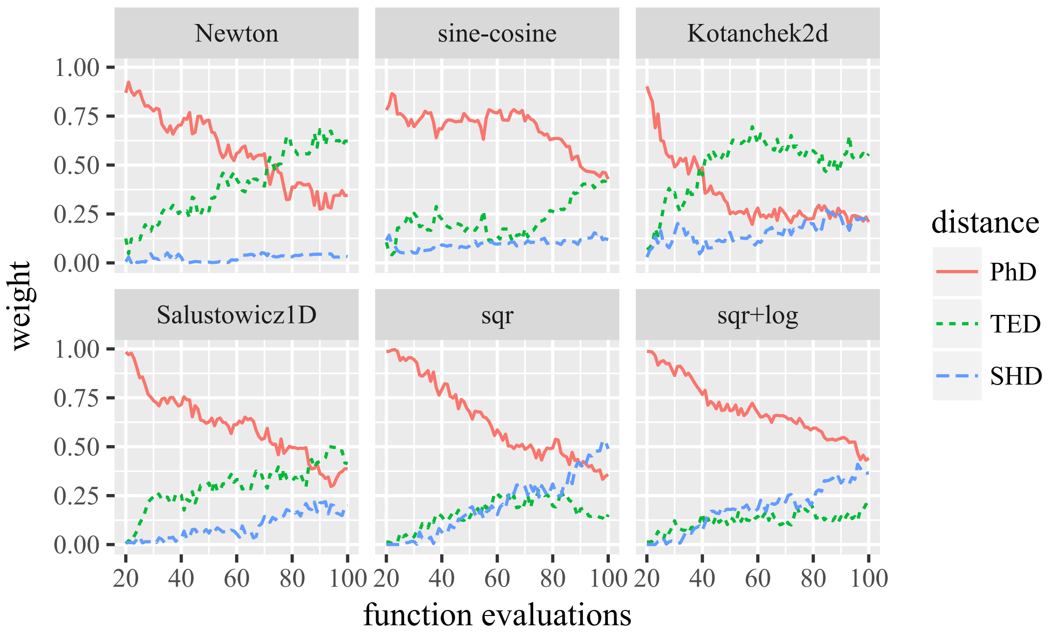


Figure 9.6: *Average normalized weights for the different kernels/distances, determined by the linear combination via MLE. Without explicit correction of indefinite matrices.*

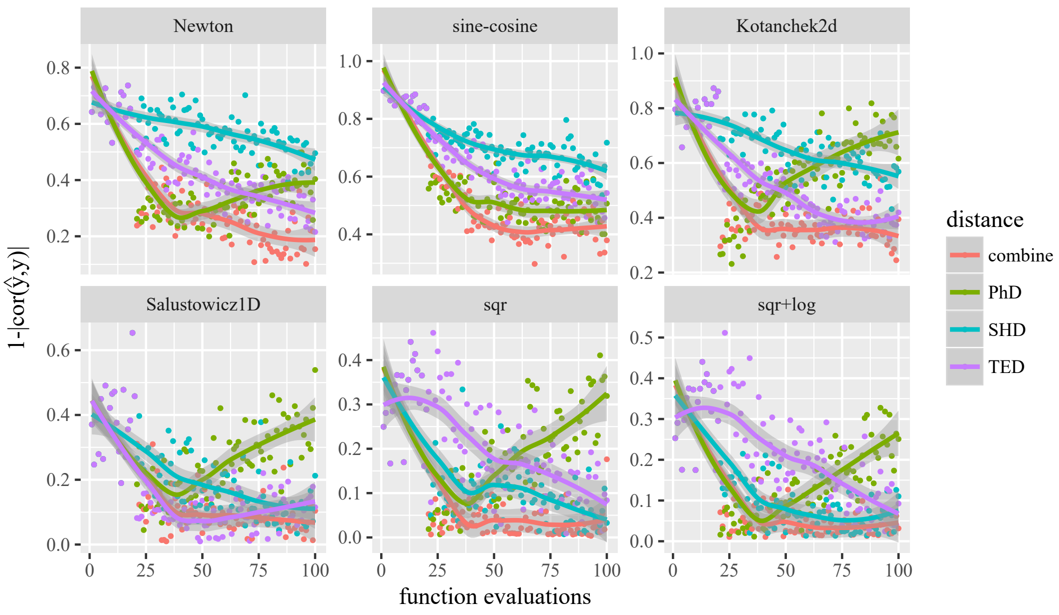


Figure 9.7: *Each function evaluation by different CEGO variants without CNSD-correction (dots), with a LOESS fit (lines), and its 95% confidence interval (shaded).*

Correcting Definiteness As a consequence, we repeated the experiments. This time, we corrected the indefinite kernels explicitly. We used CNSD-correction with spectrum flip and repaired matrix conditions via re-transformation (cf. Chapter 8).

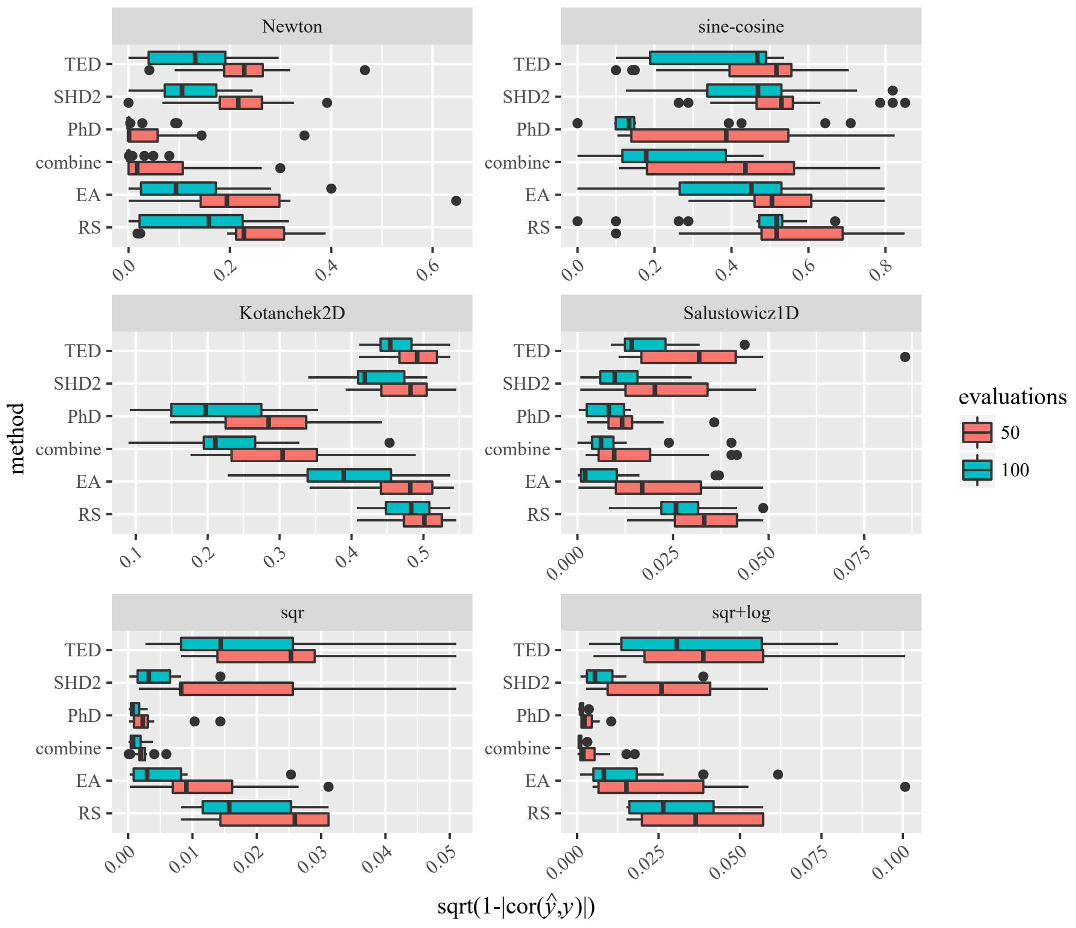


Figure 9.8: *Box plot of best-observed values after 50 and 100 evaluations. The performances of CEGO with single distances are denoted by the respective abbreviations. The linear combination is denoted by "combine". Potentially indefinite distances are corrected via spectrum transformation. The depicted data has been subject to a square-root transformation for the sake of readability.*

As Fig. 9.8 indicates, the performance of some CEGO variants improved. The strongest improvement was observed for Kotanchek2D. A statistical test procedure yielded similar results as for the uncorrected case, with two exceptions. CEGO+TED was now evaluated to be better than RS, and no significant differences between CEGO+SHD2 and the EA were found. The ranking changed accordingly: The EA and CEGO+SHD2 shared the second rank, CEGO+TED ranked third, and RS ranked last. In summary, the correction of definiteness improved the CEGO variants in some cases.

In a much more drastic way, the correction affected the weights determined by the linear combination, as depicted in Fig. 9.9. Except for the sine-cosine instance, the PhD was favored nearly exclusively. No shift to the genotypic distances was observed. This suggests that the previously observed decline of the PhD's weights was largely due to indefiniteness. SHD2 and TED were mostly included to counterbalance that

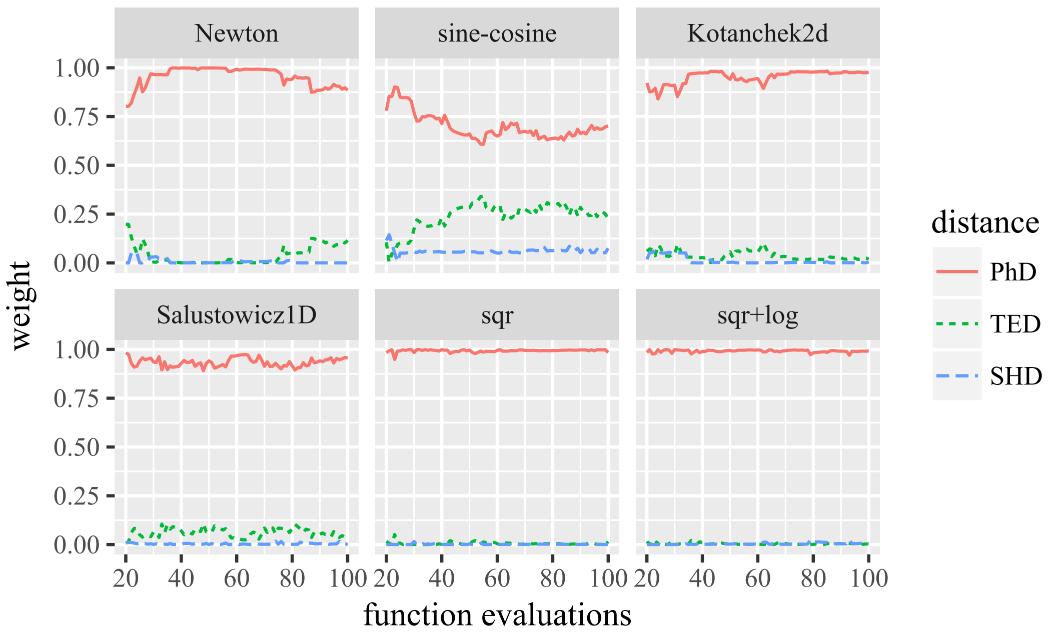


Figure 9.9: *Average normalized weights for the different kernels/distances, determined by the linear combination via MLE. With CNSD-correction.*

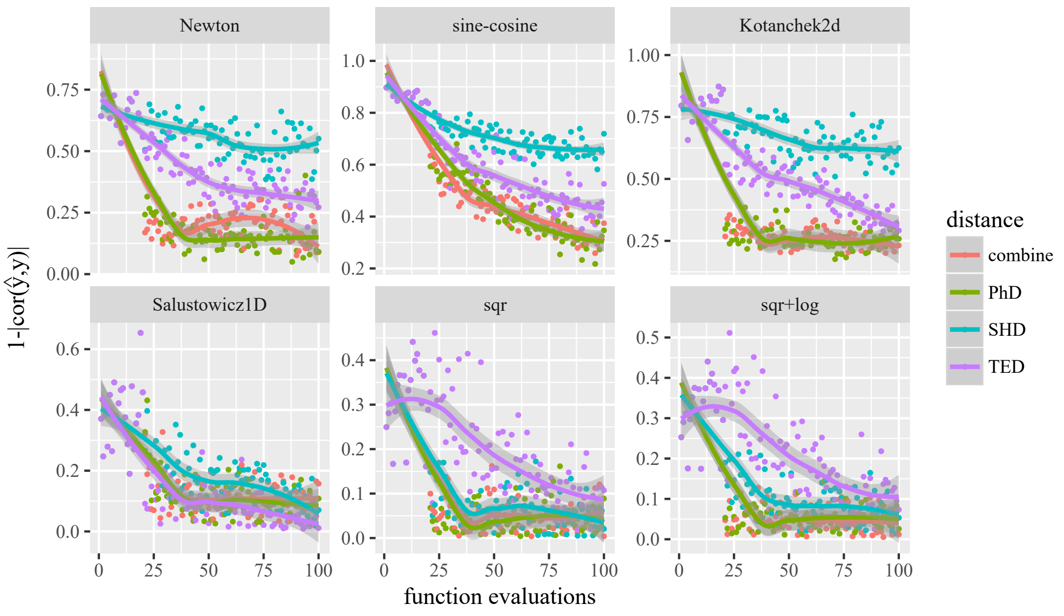


Figure 9.10: *Each function evaluation by different CEGO variants with CNSD-correction (dots), with a LOESS fit (lines), and its 95% confidence interval (shaded).*

effect. This was further confirmed by the LOESS fits shown in Fig. 9.10. The strong deterioration observed in Fig. 9.7 is no longer present.

9.1.5 Summary

We applied CEGO to symbolic regression, using a tree representation. This was connected to two core research questions, to be answered as follows.

Answer 9.1. Tree distances. What distances are available, and what properties do they have?

> We identified two types of distances. Genotypic distances operate on the tree structures themselves, whereas phenotypic distances attempt to compare the behavior or outcome of these structures. For our symbolic regression test problem, we devised a phenotypic distance based on the correlation of symbolic expressions. In addition, two variants of genotypic distances were investigated, SHD and TED.

Answer 9.2. Performance. What impact do these distances have on the performance of an SMBO algorithm?

> The overall best results were produced by the PhD. This was to be expected because the phenotypic behavior is much closer to the actual fitness. The genotypic distances may be problematic, e.g., when changes in the genotype do not lead to according changes in the fitness. Most importantly, all investigated distances are non-CNSD. Especially, the performance of a PhD-based model seemed to be affected by this lack of definiteness. A linear combination was revealed to mitigate this by shifting to the less problematic genotypic distances. Even better, a correction via spectrum flip was able to avoid this issue more comprehensively.

Overall, the idea of phenotypic distances seems to be very promising. Not only did the PhD yield good result for our use case, it could also be extended to other search domains, such as neuroevolution [96]. First tests towards that goal are described by Stork et al. [239].

Still, our results indicate that genotypic distances should not be completely disregarded. A diverse mixture of distances may help to mitigate any problems occurring with individual distances. With that said, we would often prefer more simple genotypic distances (e.g., the SHD) instead of edit distances (e.g., the TED). The costs involved in computing some edit distances may render a surrogate model infeasibly expensive. This is especially problematic for modeling in graph spaces: The graph edit distance is NP-hard [274].

Further research should investigate how well our results apply to other problem classes. Ideally, these should include actual real-world problems. While SR is a challenging task, it is unclear whether the specific challenges posed are actually representative for many real-world problems, e.g., from engineering. Frequently, engineering problems concern the optimization of geometric shapes (e.g., airfoil shapes or the shape of ship hulls). In some cases, tree representations may be of interest. Trees can describe complex geometries based on simple shapes using constructive solid geometry, as shown in Fig. 9.11. This kind of free-form shape optimization may be an interesting application for our tree-kernel-based models. Good phenotypic distances would of course depend on a specific application. Measures like the volume of an intersection of two objects could be an interesting starting point.

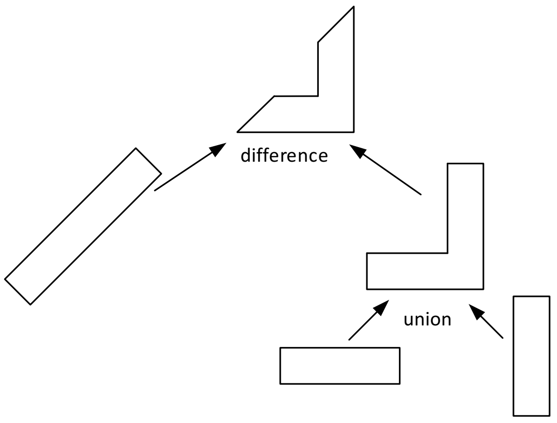


Figure 9.11: *Two-dimensional example for a more complex shape made from simple elements via a constructive solid geometry tree.*

9.2 Modeling Hierarchical Variables

A particular challenge for SMBO are hierarchical search spaces, which are spanned by conditional variables. A conditional variable has an effect on an objective function $f(x)$, if some condition is fulfilled. We refer to this as the conditional variable being active. Respectively, if the condition is not fulfilled, the variable is inactive. Hierarchical search spaces can be depicted as directed acyclic graphs. Nodes are the variables and edges denote the conditions. The edges point from the trigger variables to the conditional variables [129].

Algorithm tuning is one field where conditional variables are a frequent occurrence. Consider, e.g., the tuning of an EA. The step-size of a specific mutation operator only matters if that operator is actually used. Here, the step-size is the conditional variable. The condition is based on a binary trigger variable that decides on the inclusion of the mutation operator. Variable activity is also an issue in the tuning of neural networks [241]. Other potential applications can be found in engineering. Frequently, some process variable may only be active or relevant, if other variables or even environmental factors satisfy a condition.

Mostly, SMBO approaches for hierarchical search spaces can be categorized in three of the six strategies discussed in Chapter 3: the naive approach, mapping, and kernel-based models.

- **Naive:** There is no strict necessity to incorporate any information about activity

Section 9.2 is based on the article *"A First Analysis of Kernels for Kriging-based Optimization in Hierarchical Search Spaces"* by Zaefferer and Horn [270]. It was prepared in a coequal cooperation with Daniel Horn, who especially contributed to the statistical and visual analysis of the experimental investigation. Since this article was written in equal parts by both of its authors, it is not discussed entirely. We discuss the kernels contributed by the author of this thesis in more detail (with few verbatim adoptions) and briefly summarize the experimental results. An additional kernel (Wedge-kernel) is proposed.

in the SMBO framework. This is technically feasible, yet ignores potentially helpful information about the problem landscape.

- **Mapping:** One standard approach is to map to a non-hierarchical search space by imputing some constant value whenever a variable is inactive [246, 125, 198]. We denote this as the *imputation approach.*

- **Kernel-based:** The activity information can be integrated into the model's kernel. One example is the Arc-kernel [129, 241]. Otherwise, the hierarchical dependencies can be employed to devise tree-based, hierarchical Gaussian processes [34, 33, 133].

Following the main theme of this thesis, we are interested in using a kernel-based approach. The Arc-kernel [129, 241] is already a quite promising solution. Yet, we argue that it may not be an optimal solution in all cases. Hence, we want to investigate the following question.

Question 9.3. Hierarchical kernels. What kernels should be used to model hierarchical search spaces?

We discuss the existing Arc-kernel and propose alternatives.

9.2.1 Kernels for Hierarchical Search Spaces

In the following, our notation is based on the article by Hutter and Osborne [129]. A function $\delta_i(x)$ is used to denote the activity condition. That is, if $\delta_i(x) = $ false, the i-th variable x_i of the candidate solution x is inactive. Correspondingly, $\delta_i(x) = $ true implies that the same variable is active. For all kernels we discuss, the function $\delta_i(.)$ could as well be dependent on environmental or extraneous variables, which are not present in x. This may be especially of interest in potential engineering applications. For the sake of simplicity, we stick to the $\delta_i(x)$ notation.
As throughout most of this thesis, we use an exponential kernel. Here, it has the form $\mathrm{k}(x, x') = \exp(-\sum_{i=1}^{m} \mathrm{d}_i(x, x'))$. We discuss and propose different choices for the distances $\mathrm{d}_i(x, x')$.

The Arc-kernel The Arc-kernel has been proposed by Hutter and Osborne [129] to enable modeling of conditional variables. When comparing a variable from two data samples x_i and x'_i, the Arc kernel assumes:

- If both are inactive ($\delta_i(x) = \delta_i(x') = $ false), then the distance is $\mathrm{d}_i(x_i, x'_i) = 0$. This is obvious: If the compared variables are both inactive, they both have the same impact on the observed variable y (i.e., no impact).

- If both are active ($\delta_i(x) = \delta_i(x') = $ true) the distance depends on the values of x_i and x'_i.

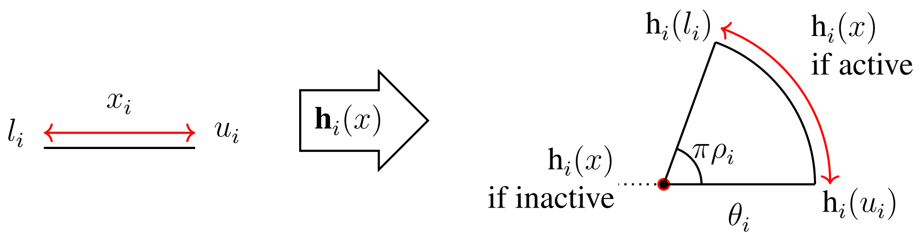


Figure 9.12: *The mapping function $h_i(x)$ of the Arc-kernel, which embeds conditional variables in a two-dimensional space by placing active samples on an arc and inactive samples on the center of the corresponding circle. Left: the original conditional variable. Right: the embedding.*

- Finally, if only one variable is active ($\delta_i(x) \neq \delta_i(x')$), then Hutter and Osborne assume that the samples are incomparable. Therefore, they propose that the distance in this dimension should be a constant.

We will later show a more direct implementation of these assumptions (Ico-kernel). However, it turns out that this direct implementation is non-CNSD. Hutter and Osborne avoid this by finding a suitable embedding function $\mathrm{h}_i(x)$. This function maps the hierarchical data into a Euclidean space, so that a distance $\mathrm{d}_i(x, x')$ can be written as $\mathrm{d}_i(x, x') = \mathrm{d}_{\text{Euc}}(\mathrm{h}_i(x), \mathrm{h}_i(x'))$. By making sure that the embedding respects the assumptions listed above, Hutter and Osborne derive a CNSD distance that also satisfies these assumptions [129]. For continuous x, the mapping function is:

$$\mathrm{h}_i(x) = \begin{cases} \begin{bmatrix} 0 & 0 \end{bmatrix}^{\mathrm{T}}, & \text{if } \delta_i(x) = \text{false} \\ \theta_i \begin{bmatrix} \sin(\pi\rho_i \frac{x_i}{u_i - l_i}) & \cos(\pi\rho_i \frac{x_i}{u_i - l_i}) \end{bmatrix}^{\mathrm{T}}, & \text{otherwise.} \end{cases} \tag{9.5}$$

Here, u_i, l_i are the upper and lower bounds of the i-th variable. The mapping function $\mathrm{h}_i(x)$ embeds the conditional variable in a two-dimensional space, as depicted in Fig. 9.12. The depicted arc shape of the embedding lends this kernel its name [241]. The resulting CNSD distance function is:

$$\mathrm{d}_{\text{Arc}\,i}(x, x') = \begin{cases} 0, & \text{if } \delta_i(x) = \delta_i(x') = \text{false} \\ \theta_i, & \text{if } \delta_i(x) \neq \delta_i(x') \\ \theta_i \sqrt{2 - 2\cos(\pi\rho_i \frac{x_i - x_i'}{u_i - l_i})}, & \text{if } \delta_i(x) = \delta_i(x') = \text{true.} \end{cases} \tag{9.6}$$

The kernel variables $\theta_i \in \mathbb{R}^+$ and $\rho_i \in [0, 1]$ can be determined by MLE. A very similar distance for categorical variables is also described by Hutter and Osborne [129]. In the experiments, we used the square root of the Euclidean distance in the embedded space, to be consistent with the other kernels.

Indefinite Conditional Kernel As an alternative, we propose the Indefinite Conditional kernel (Ico-kernel), which implements the assumptions of the Arc-kernel in a more direct way:

$$\mathrm{d}_{\mathrm{Ico}i}(x,x') = \begin{cases} 0, & \text{if } \delta_i(x) = \delta_i(x') = \text{false} \\ \rho_i, & \text{if } \delta_i(x) \neq \delta_i(x') \\ \theta_i \mathrm{d}_i(x_i, x_i'), & \text{if } \delta_i(x) = \delta_i(x') = \text{true.} \end{cases}$$

Just like the Arc-kernel, $\mathrm{d}_{\mathrm{Ico}i}(x,x')$ is zero if both samples are inactive, it is a constant (ρ_i) if activity differs, and it directly depends on the samples if both are active. Here, $\mathrm{d}_i(x_i, x_i')$ is an appropriate distance for the respective variable. For instance, if $x_i \in \mathbb{R}$ then $\mathrm{d}_i(x_i, x_i') = (x_i - x_i')^2$. Or, if x_i is a categorical variable, then $\mathrm{d}_i(x_i, x_i')$ could be the Hamming distance. The parameter $\rho_i \in \mathbb{R}^+$ can be determined by MLE. Unfortunately, this kernel is indefinite, as implied by the name we chose.

Proposition 3. *$d_{Ico i}(x,x')$ is not CNSD.*

Proof. Remember that CNSD functions satisfy $\mathbf{c}^{\mathrm{T}}\mathbf{D}\mathbf{c} \leq 0$, for all distance matrices $\mathbf{D}_{\mathrm{Ico}}$ and all $\mathbf{c} \in \mathbb{R}^n$ subject to $\sum_{i=1}^n c_i = 0$ (cf. Section 4.4). We prove that this does not hold with an example. For an arbitrary conditional variable x_i, we choose the kernel parameters $\theta_i = 10$ and $\rho_i = 1$. Three samples are compared: $x_i^{(1)} = 1$, $x_i^{(2)} = 2$ and $x_i^{(3)} = 1$. The first two are active, the last is inactive (due to some condition on a different variable x_j). For this example, we use the square deviation when comparing active variables, $\mathrm{d}_i(x_i, x_i') = (x_i - x_i')^2$. This yields the distance matrix

$$\mathbf{D}_{\mathrm{Ico}_i} = \begin{bmatrix} 0 & 10 & 1 \\ 10 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

With $\mathbf{c} = \begin{bmatrix} 0.5 & 0.5 & -1 \end{bmatrix}^{\mathrm{T}}$, we receive $\mathbf{c}^{\mathrm{T}}\mathbf{D}_{\mathrm{Ico}_i}\mathbf{c} = 3 > 0$. Hence, the distance is not CNSD. □

To account for the lack of definiteness in the experiments from the original article [270], we also tested one of the correction methods from Chapter 8. We chose a CNSD-correction via spectrum flip, with repair by re-transformation. We denote this corrected variant as the IcoCor-kernel.

Imputation Kernel So far, all kernels that we discussed are based on the main ideas of the Arc-kernel. Importantly, the Arc-kernel assumes that an active and an inactive sample cannot be compared, resulting into a constant distance (in the conditional dimension). This is a rather careful assumption. It prefers to ignore any knowledge encoded in the active variable, rather than employing it when it may be misleading or irrelevant.

However, there may be some cases where this is not true. In fact, since one sample is active, small changes to that sample lead to a change in the observed variable. The Arc-kernel does not respect this. More formally, consider the case where

$$\delta_i(x) = \text{false and } \delta_i(x') = \delta_i(x'') = \text{true,}$$

where $x_i'+h = x_i''$ and h is some small perturbation. Then, the corresponding distances are $\mathrm{d}_{\mathrm{Arc}i}(x,x') = \mathrm{d}_{\mathrm{Arc}i}(x,x'')$, which to some extent contradicts that $f(x') \neq f(x'')$. This observation also applies to the Ico-kernel variant.
Hence, we propose a different assumption about the distance of conditional variables: If $\delta_i(x) \neq \delta_i(x')$, then the distance in dimension i should depend on the active variable, but not on the inactive one.
We propose to implement this assumption into a distance by comparing the active variable against a parameter ρ_i, i.e.,

$$\mathrm{d}_{\mathrm{Imp}_i}(x,x') = \begin{cases} 0, & \text{if } \delta_i(x) = \delta_i(x') = \text{false} \\ \theta_i \mathrm{d}_i(x_i', \rho_i), & \text{if } \delta_i(x) = \text{false} \neq \delta_i(x') \\ \theta_i \mathrm{d}_i(x_i, \rho_i), & \text{if } \delta_i(x) = \text{true} \neq \delta_i(x') \\ \theta_i \mathrm{d}_i(x_i, x_i'), & \text{if } \delta_i(x) = \delta_i(x') = \text{true}, \end{cases}$$

where d_i is an appropriate distance for the i-th variable. The kernel parameter ρ_i has the same data type as x_i. That is, if $x_i \in \mathbb{R}$ then $\rho_i \in \mathbb{R}$, and we could use the Euclidean distance. Accordingly, if x_i is categorical then ρ_i is also categorical, and we could use the Hamming distance. While the data type is identical, the bounds u_i and l_i need not be identical for x_i and ρ_i. In fact, it may be recommendable to use a wider range of values for ρ_i, to account for cases where good candidates for imputation lie outside the original bounds. Similarly, categorical ρ_i should have an additional category (or level), to account for cases where none of the categories of x_i are good imputation candidates.

Proposition 4. *$d_{Imp_i}(x,x')$ is CNSD if the underlying distance $d_i(x_i,x_i')$ is CNSD.*

Proof. To prove this, we follow the same approach used for the Arc-kernel by Hutter and Osborne [129]. We need to show that $\mathrm{d}_{\mathrm{Imp}_i}(x,x')$ is equivalent to a CNSD distance in some embedding space defined by the mapping function $\mathrm{h}_i(x)$. That means, we should be able to write $\mathrm{d}_{\mathrm{Imp}_i}(x,x') = \mathrm{d}_i(\mathrm{h}_i(x),\mathrm{h}_i(x'))$, where $\mathrm{d}_i(.,.)$ is a CNSD distance function.
For $\mathrm{d}_{\mathrm{Imp}_i}(x,x')$, the mapping function is

$$\mathrm{h}_i(x) = \begin{cases} x_i, & \text{if } \delta_i(x) = \text{true} \\ \rho_i, & \text{otherwise.} \end{cases}$$

Hence, the resulting distance $\mathrm{d}_{\mathrm{Imp}_i}(x,x')$ is CNSD, if $\mathrm{d}_i(x_i,x_i')$ is CNSD. □

The mapping function $\mathrm{h}_i(x)$ embeds the conditional variable in a one-dimensional space, as depicted in Fig. 9.13.
This kernel is closely related to the imputation approach mentioned in the introduction of this chapter. It imputes a value ρ_i for every inactive variable. Therefore, we call it the imputation kernel or Imp-kernel. The main difference to the imputation approach is that the Imp-kernel does not choose the imputed value a-priori. Rather, it is determined by an appropriate training method (i.e., MLE in case of Kriging, or cross-validation in case of SVM).
In comparison to the Arc- and Ico-kernels, the Imp-kernel's main drawback is the nature of the parameter ρ_i. Firstly, the bounds on ρ_i may be difficult to set. Secondly, ρ_i may be a categorical parameter. Both issues complicate the training procedure.

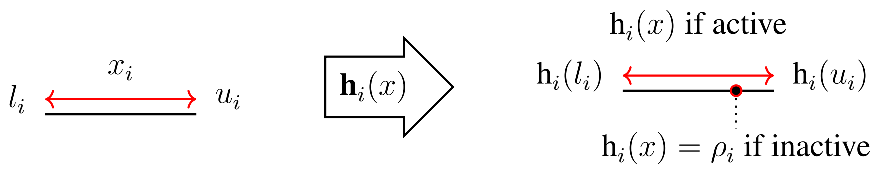


Figure 9.13: *The mapping function $h_i(x)$ of the Imp-kernel, which embeds conditional variables in a one-dimensional space by placing active samples on a line and inactive samples at a single point. Left: the original conditional variable. Right: the embedding.*

The Imputation-Arc Kernel In practice, it may be a challenge to determine whether the Imp- or Arc-kernel should be preferred, especially when the problem is a true black-box. For such a case, we suggest employing the methods from Chapter 6, to combine or select the correct kernel. Specifically, we suggest the ImpArc-kernel

$$\mathrm{k}_{\mathrm{ImpArc}}(x, x') = \exp\Big(-\sum_{i=1}^{m} \beta_{1,i}\mathrm{d}_{\mathrm{Arc}\,i}(x, x') + \beta_{2,i}\mathrm{d}_{\mathrm{Imp}_i}(x, x')\Big),$$

where $\beta_{1,i}, \beta_{2,i} \in \mathbb{R}^+$ are determined by MLE. Clearly, other kernels may also be included, but potentially redundant kernels should be avoided (e.g., Ico-, IcoCor-, and Arc-kernel).

Wedge Kernel Finally, we propose a more general kernel that is based on the same assumptions as the Imp-kernel: the Wedge-kernel. We will show that the Imp-kernel is a special case of the Wedge-kernel. Furthermore, the Wedge-kernel has similarities to the Arc-kernel, since it also maps to a two-dimensional space.
The Wedge-kernel for $x_i \in \mathbb{R}$ uses the embedding, or mapping function

$$\mathrm{h}_i(x) = \begin{cases} \begin{bmatrix} 0 & 0 \end{bmatrix}^{\mathrm{T}}, & \text{if } \delta_i(x) = \text{false} \\ \begin{bmatrix} \theta_{1,i} + v\left(\theta_{2,i}\cos(\rho_i) - \theta_{1,i}\right) & v\theta_{2,i}\sin(\rho_i) \end{bmatrix}^{\mathrm{T}}, & \text{otherwise,} \end{cases} \tag{9.7}$$

with the scaled variable value $v = (x_i - l_i)/(u_i - l_i)$. Here, the kernel parameters are $\rho_i \in [0, \pi]$, $\theta_{1,i} \in \mathbb{R}^+$, and $\theta_{2,i} \in \mathbb{R}^+$, specifying the angle and two adjacent side lengths of a triangle. Instead of mapping active samples onto an arc, they are mapped onto one side of a triangle, or wedge[2]. The inactive samples are mapped to the opposing corner of the wedge, similarly to the Arc-kernel. The mapping is visualized in Fig. 9.14.
Just as with the Imp-kernel, the distance between an active and an inactive sample depends only on the active one. In contrast to the other kernels, we need an additional third parameter to specify the shape of the wedge. This may be a drawback.
When i) $\theta_{1,i} = 0$, ii) $\theta_{2,i} = 0$, iii) $\rho_i = 0$, or iv) $\rho_i = \pi$, the wedge collapses to a line segment and a point that both lie on the same line. This implies a collapse into a

[2]We use the term "wedge" to avoid confusion with the triangle kernel $\mathrm{k}_{\mathrm{tri}}(x_i, x_i') = 1 - |x_i - x_i'|$.

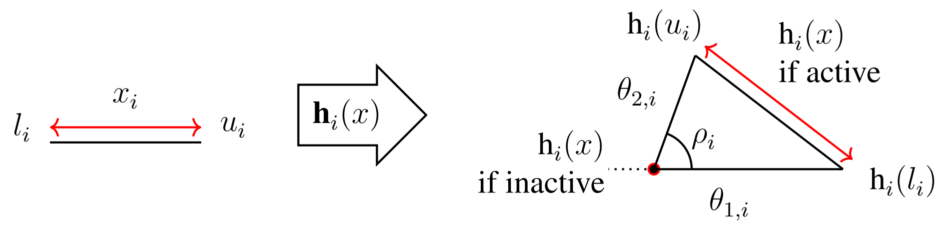


Figure 9.14: *The mapping function $h_i(x)$ of the Wedge-kernel, which embeds conditional variables in a two-dimensional space by placing active samples on a side of a triangle (wedge) and inactive samples at the opposing corner. Left: the original conditional variable. Right: the embedding.*

one-dimensional space, identical to the embedding of the Imp-kernel. Hence, the Imp-kernel is a special case of the Wedge-kernel. Here, the four cases i)-iv) define where the point lies in relation to the line segment (on the upper bound, on the lower bound, outside of the bounds, and inside of the bounds).
Based on the squared Euclidean distance, the Wedge-kernel can be written as

$$
\mathrm{d}_{\mathrm{Wedge}_i}(x,x') = \|\mathrm{h}_i(x)-\mathrm{h}_i(x')\|^2 =
\begin{cases}
0, & \text{if } \delta_i(x)=\delta_i(x')=\text{false}\\
\Big(\theta_{1,i}+\hat{v}\left(\theta_{2,i}\cos(\rho_i)-\theta_{1,i}\right)\Big)^2+\left(\hat{v}\theta_{2,i}\sin(\rho_i)\right)^2, & \text{if } \delta_i(x)=\text{false}\neq\delta_i(x')\\
\Big(\theta_{1,i}+v\left(\theta_{2,i}\cos(\rho_i)-\theta_{1,i}\right)\Big)^2+\left(v\theta_{2,i}\sin(\rho_i)\right)^2, & \text{if } \delta_i(x)=\text{true}\neq\delta_i(x')\\
\left(\frac{x_i-x_i'}{u_i-l_i}\right)^2\left(\theta_{1,i}^2+\theta_{2,i}^2-2\theta_{1,i}\theta_{2,i}\cos(\rho_i)\right), & \text{if } \delta_i(x)=\delta_i(x')=\text{true,}
\end{cases}
$$

with $\hat{v}=(x_i'-l_i)/(u_i-l_i)$.

Proposition 5. *$d_{\mathit{Wedge}_i}(x,x')$ for $x_i\in\mathbb{R}$ is CNSD.*

Proof. Again, we can employ the same approach as used for the Arc-kernel by Hutter and Osborne [129]. We need to show that $\mathrm{d}_{\mathrm{Wedge}_i}(x,x')$ is equivalent to a CNSD distance (here: squared Euclidean) in some embedding space defined by the mapping function $\mathrm{h}_i(x_i)$. This function is given in Eq. (9.7). Hence, $\mathrm{d}_{\mathrm{Wedge}_i}(x,x')$ is CNSD. □

Similarly to the Imp-kernel, a variant for categorical variables can be derived for the Wedge-kernel. One simple idea would be the following mapping, which maps a single categorical variable to a space spanned by a categorical variable and a real variable:

$$
\mathrm{h}_i(x) = \begin{cases} \begin{bmatrix}\rho_i & 0\end{bmatrix}^{\mathrm{T}}, & \text{if } \delta_i(x)=\text{false}\\ \begin{bmatrix}x_i & \theta_{2,i}\end{bmatrix}^{\mathrm{T}}, & \text{otherwise.}\end{cases}
$$

where ρ_i is categorical and $\theta_{2,i}\in\mathbb{R}^+$. We can use a mixed distance in the embedding space, with the scaled Hamming distance ($\theta_{1,i}\mathrm{HD}(x_i,x_i')$, with $\theta_{1,i}\in\mathbb{R}^+$) for the categorical variable and Euclidean distance for the real variable. The corresponding CNSD distance function for categorical variables is

$$\mathrm{d}_{\text{WedgeCat}_i}(x, x') = \begin{cases} 0, & \text{if } \delta_i(x) = \delta_i(x') = \text{false} \\ \theta_{1,i}\text{HD}(x'_i, \rho_i) + \theta_{2,i} & \text{if } \delta_i(x) = \text{false} \neq \delta_i(x') \\ \theta_{1,i}\text{HD}(x_i, \rho_i) + \theta_{2,i} & \text{if } \delta_i(x) = \text{true} \neq \delta_i(x') \\ \theta_{1,i}\text{HD}(x_i, x'_i) & \text{if } \delta_i(x) = \delta_i(x') = \text{true}, \end{cases}$$

This distance function is closely connected to the real-valued variant. It can collapse to a categorical variant of the Imp-kernel if $\theta_{2,i} = 0$. In addition, the distance between an active and inactive sample depends on the active sample. Also, the scale of distances between active samples, and between samples of mixed activity may differ, depending on the parameterization.

The Wedge-kernel was not included in the original publication. Hence, it was also not included in the experiments summarized in the following.

9.2.2 Previous Experiments

Experiments are required to determine how the proposed kernels behave in practice. As a first step, we proposed a simple, artificial test function [270]:

$$f(x) = (x_1 - c_1)^2 + \begin{cases} 0 & \text{if } x_1 \leq c_3 \\ (x_2 - 0.5)^2 + c_2 & \text{else} \end{cases}, \tag{9.8}$$

Here, the coefficients $c_i \in \mathbb{R}$ control the behavior. The coefficient c_3 defines when the conditional variable (x_2) becomes active. The coefficient c_1 affects the location of the optimum, which may lie in either the inactive or the active region. Importantly, c_2 controls whether the structure of the Imp-kernel fits to the objective function. For instance, if $c_2 = 0$, the active term $(x_2 - 0.5)^2$ becomes zero for $x_2 = \rho = 0.5$. Hence, 0.5 is an ideal value to be imputed. In other words, the imputation kernel is a good fit if there exists a value $x_i = \rho_i$ for which the function behaves as if x_i is inactive, while it is actually active. We refer to the original article [270] for more details on the experimental setup.

Four main conclusions were drawn from the experiments [270]. Firstly, incorporating the hierarchical structure into the kernel was crucial for the optimization performance. Experiments with a standard kernel performed worst. Secondly, the experiments confirmed that the Imp-kernel worked well when the objective function fit well to the Imp-kernel's structure (here: $c_2 = 0$). Else, the Arc-kernel performed better. Thirdly, the Arc-kernel, the Ico-kernel, and the IcoCor-kernel all produced similar results overall. Finally, a weighted sum of Arc- and Imp-kernel often achieved the middle ground between their individual performances.

Hence, we answer the core question of Section 9.2:

Answer 9.3. Hierarchical kernels. What kernels should be used to model hierarchical search spaces?

If any expert knowledge suggests that the Imp-kernel is a good fit, it is recommendable to select that kernel. If expert knowledge suggests otherwise, the Arc-kernel (or a variant of it) is probably more suited. When no expert knowledge is available, it may be attempted to derive this knowledge from the data, i.e., by learning a weighted sum of the two kernels.

The Wedge-kernel was not tested in our experiments. Hence, we remark on it below.

9.2.3 Future Experiments

The artificial test function from Eq. (9.8) was a good first step, since it provided a very transparent test case. It allowed understanding *why* the tested algorithms behaved as they did.
Nevertheless, we do not know how relevant this simple test function is for real-world problems from algorithm tuning or engineering. Therefore, tests with more relevant problems are an obvious suggestion for future work. Especially, algorithm tuning is a promising use case because it is more easily accessible than most applications from the engineering domain.
The Wedge-kernel is likely to avoid shortcomings of both the Imp-kernel and the Arc-kernel. The Imp-kernel may force a model to be smooth when transitioning from the active to the inactive region, since distances may have to become zero close to the boundary. This is relaxed by the Wedge kernel, which allows some minimal distance to remain. At the same time, the Wedge kernel avoids the Arc-kernel's assumption of constant distance between inactive and active samples. Hence, the Wedge-kernel seems very promising. Experiments should verify whether these promising features manifest in terms of good performance, or whether the need to determine an additional kernel parameter outweighs the benefits.
Finally, it may be of interest for future work to define an encompassing kernel that includes the Arc- and Wedge-kernel as special cases.

9.3 Simulation for Test Function Generation

Throughout this thesis, algorithms and models were developed to provide good solvers for expensive, discrete optimization problems. We mostly used artificial test functions to evaluate, compare, and understand these developments. Test functions are a weak

Section 9.3 is mostly based on *"Simulation Based Test Functions for Optimization Algorithms"* by Zaefferer et al. [268]. The text is in parts taken verbatim from that publication. However, it was largely rewritten and extended to 1) give a clearer motivation supported by visualizations, 2) add some additional remarks on related approaches, 3) discuss the advantages and disadvantages in a more structured way, 4) provide more illustrative examples to explain how the test function generator works, and 5) show results from an additional experiment with a random forest model.

point of any experimental procedure: It is not always clear whether they are a good basis to judge algorithm performance.
This issue is not limited to this thesis. In general, numerical tests are a crucial element of the development, improvement, and understanding of optimization algorithms. This leads to our main research question for this section:

Question 9.4. Generating test functions. How can we generate test functions that fulfill requirements such as difficulty and real-world relevance?

To provide a potential solution, this section introduces a test function generator based on Kriging simulation. This framework may be used to generate diverse and relevant benchmarks for discrete SMBO algorithms, but also applies to other problem domains.

9.3.1 Motivation

It is particularly difficult to provide test functions for expensive optimization problems, which are in the focus of this thesis. Expensive problems often require that the experimenters have access to complex simulation codes. The simulation codes are also often confidential if they are developed by an industry partner. Even worse, if real-world experiments are performed, access to laboratory equipment or an industrial plant may be needed. This access may be available for a single research group, but not the whole community. Even if researchers have access to a simulation or experiment, the costs for evaluations are large. This often prohibits comparing a larger number of algorithms or configurations.
One approach to this issue is to employ sets of artificially created test functions (cf. [183, 259]). This procedure is well-established in continuous optimization. In continuous optimization, test functions can be composed by combining step functions, linear functions, convex functions and sinusoidal functions. Such compositions enable to create varied test functions with different properties. Yet, it is not guaranteed that the resulting functions are actually representative for real-world problems.
This is an essential issue for the methods discussed in this thesis. Test functions for expensive, combinatorial problems are even more problematic than those for continuous problems. We are not aware of any real-world, expensive, combinatorial optimization problems that have been made openly available. Thus, most of the benchmark problems that were used in this thesis are well known problems that are cheap to compute, e.g., the TSP. Just as in the continuous case, it is questionable whether performances estimated for these test problems are actually representative for real-world problems.
Thus, we would like to propose a generator that produces test functions with desirable features. Numerous important test function features are listed in the literature, e.g., modality, dimensionality, continuity, linearity, separability, and scalability [73, 259, 113]. We focus on the following criteria. Importantly, some of these criteria apply to a *test function generator* rather than individual (or sets of) test functions.

(C.1) Difficulty: This feature is related to the statement by Whitley et al. [259]. They state that test functions should be "*resistant to hill-climbing*". That is, prob-

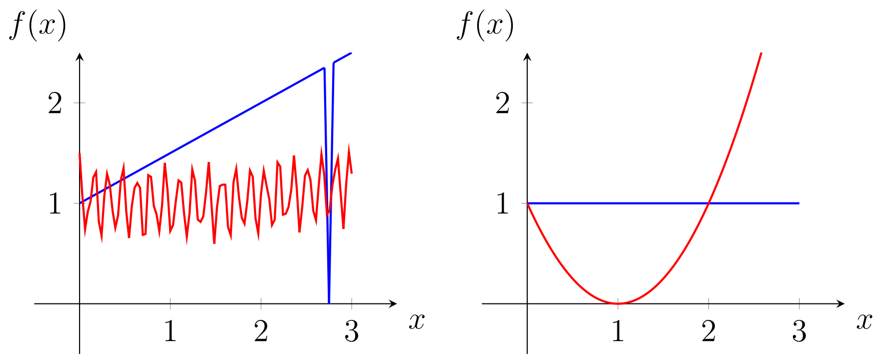


Figure 9.15: *Examples of continuous minimization problems that may be too difficult (left) or too easy (right) to be used as meaningful test functions for black-box optimization algorithms.*

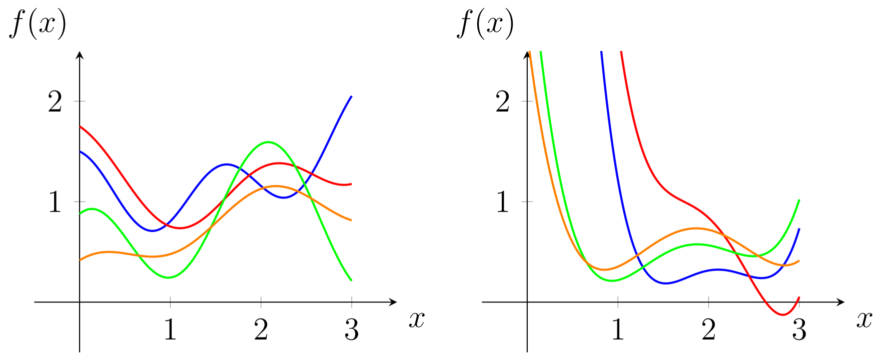


Figure 9.16: *Examples for two diverse sets of test functions.*

lems should not be too easy to solve. If a problem has low difficulty, even a poor algorithm may perform well. Extending this notion, exceedingly difficult problems should also be avoided. If a problem is too difficult (e.g., needle in the haystack), all algorithms are equally bad. Figure 9.15 illustrates potential examples for these cases.

(C.2) Diversity: Diversity implies that individual instances in a set of test functions should not all be the same (or very similar). Rather, test functions should be varied and randomized. This feature may apply to sets of test functions as well as test function generators. Diverse, randomized test instances enable to select separate instances for the tuning of algorithms and for the comparison of algorithms. Ideally, test functions for a comparison are not known *a priori*, that is, when planning the experiment. This reflects a standard in machine learning, where data sets are partitioned into a training, a validation, and a test set [114]. On the other hand, too much diversity may also be detrimental. If different test functions have nothing in common, it is unlikely that any general conclusions can be drawn from the results. Figure 9.16 shows some examples of test functions that may be sufficiently diverse.

(C.3) Flexibility: In addition to the diversity of instances, test functions should neither be restricted to one rigid problem class. Rather, a test function generator should be able to reflect a varied set of problem classes. Flexibility is also used in machine learning and can be linked to the number of parameters that are necessary to specify a model [132]. The term “generalizability” is used in a similar way by Bartz-Beielstein [21]. For example, a flexible function generator would be able to produce functions on both sides of Fig. 9.16.

(C.4) Relevance: The test problems should reflect real-world problem behavior, so that performance estimates are more meaningful to practitioners. If an algorithm is developed for a certain application, the test functions should reproduce the behavior of the application as much as possible.

(C.5) Cost: Generated test functions should be inexpensive to evaluate, allowing for numerous tests and comparisons.

While all criteria are important, it is likely that they have to be traded off against each other in practice. For instance, some problems may become more costly (C-5) the more relevant to a real-world problem they are (C-4).
Test function generators that satisfy the criteria (C.1)-(C.5) to some extent have been suggested in previous research [218, 204, 21, 90, 88, 72]. These approaches attempt to learn data-driven regression models of the objective function. The resulting models are used as test functions. Unfortunately, nearly all data-driven regression or interpolation models result into some form of smoothed function landscape. Hence, the resulting test functions may be easier to solve than the underlying real-world problems.
Therefore, test function generators based on data-driven models should respect the following criterion:

(C.6) Ruggedness: test functions derived from data-driven models should not smoothen the problem landscape.

This also agrees with Preuss et al. [204], who state that “*accuracy is much less important than topology*”. In other words, we are less interested to receive data-driven test functions that predict accurate function values. Rather, we want test functions that behave as the real objective function (e.g., in terms of ruggedness or number of optima).
In contrast to previous approaches, we attempt to avoid problems with smoothing (C.6) by using simulation instead of estimation of Kriging models. In addition, simulation is a principled way to generate diverse test functions (C.2). We introduce these ideas with simple, continuous examples. The benefits and practical limits of our approach are demonstrated in a case study based on a real-world data set.

9.3.2 Related Approaches

A very basic and inexpensive way to generate test functions is to use simple mathematical expressions. One classical example is the sphere function $f(x) = \sum(x_i)^2$, which reflects the behavior of many continuous optimization problems in the vicinity of the optimum [183]. Sets of such functions are often collected to form a larger set of test functions, e.g., combining functions like the sphere, Branin, or Rosenbrock function.

The advantage of fixed test function sets is that studies become comparable. Hence, runs of "old" algorithms that have already been tested need not be repeated to compare them to "new" algorithms. A further benefit is that the properties and behavior of these functions are known. This allows drawing conclusions about an algorithm's ability to handle certain types of challenges (e.g., non-separable problems). The disadvantage of fixed test function sets is that they are known *a priori*. This allows designing or tuning algorithms only for these specific functions. Hence, the algorithms may be over-adapted to the structure of the function set. This issue is analogous to the problem of overfitting in statistics.

We denote methods that create classical test function sets [73, 183, 228] as *synthetical*. By this, we imply that the functions are designed from scratch, combining individual function elements to produce certain behavior (e.g., multimodality, non-separability).

The well known Comparing Continuous Optimizers (COCO) platform or *Black-Box Optimization Benchmark* (BBOB) [112] relies on a fixed, synthetical set of artificial test functions [113]. Furthermore, BBOB attempts to generate randomized, varied problem instances by, e.g., translating and rotating the underlying functions [112].

Gallagher and Yuan [97] propose the *Gaussian Landscape Generator* (GLG). The GLG is a synthetical approach that produces random combinations of Gaussian curves with different centers, widths, and rotation. At each point in the search space, the fitness is the maximum over all Gaussian curves. The number of local optima can be controlled via changing the number of Gaussian curves. Since the orientation and placing is randomly determined, the GLG can generate numerous, diverse test functions. While this satisfies C.1 and C.2, it is unclear whether the resulting functions are representative for real-world problems. Similar test function generators are described by Barrera and Coello [19].

Another synthetical test function generator is the *Krigifier* that derives its name from the Kriging model. The Krigifier generates random Gaussian processes [248]. The user can specify the trend and covariance structure, and the Krigifier generates a realization with these properties. Since the number of potential model realizations is infinite, this allows for the generation of diverse functions sets. The difficulty can be tuned with the parameters of the covariance function. As for most synthetical test function generators, the real-world relevance is not clear.

In contrast to these synthetical approaches, we denote data-driven test function sets as *observational*, because they are derived from observations of real functions. As an early example, Rudolph et al. [218] propose the tuning of an optimization algorithm's performance via a Kriging surrogate model. A surrogate model of the expensive system is used as a test function for the optimization algorithm. Their approach has also been used to tune the CMA-ES for a set of artificial benchmarks [204]. Preuss et al. [204] conclude, "*that the concept of using a surrogate model approach for tuning optimization algorithms possesses a fruitful and as yet unexplored potential for*

optimization under scarce resources".
An observational approach is also discussed by Bartz-Beielstein [21]. The proposed benchmarking framework includes randomized test function generation based on data-driven models, variation of model parameters to generate diverse instances, and appropriate statistical tools for the analysis of results based on mixed models [62].
Similarly to these approaches, Flasch [90] suggest to use Kriging models as test functions for symbolic regression with genetic programming. To derive a more varied set of functions from the Kriging model, its parameters are changed in a controlled but randomized way. For instance, the kernel parameters of the Kriging model may be slightly enlarged or decreased. The predictor of the altered model is the desired test function. We denote this as *parameter-variation*.
Fischbach et al. [88] extend the parameter-variation approach to deal with two problems. Firstly, data driven model may not be sufficiently sensitive to changes of the model parameters. This yields test functions that are too similar, and violate our requirement for diversity (C.2). Secondly, the converse may be true. That is, small changes of the parameters yield large changes in the behavior of the model. The resulting test functions may not have any similarity with the original real-world problem. Fischbach et al. [88] suggest computing some measure of similarity between the test functions and the original problem. If the computed values violate a lower or upper bound specified by the user, the test functions are disregarded.
A recent observational approach by Dang et al. [72] also concerns algorithm tuning. They propose to use surrogate benchmarks based on random forest models. To verify that their models respect the actual problem behavior, they consider the accuracy, homogeneity, and variable importance of the resulting models.
In this thesis, we propose an observational approach, based on Kriging. Unlike earlier works, our generator uses simulation rather than estimation of Kriging models. This has two advantages. Firstly, simulation may produce diverse test functions in a principled way (C.2). Secondly, it may avoid the pitfall of smoothing (C.6).

9.3.3 Simulation-based Test Function Generator

Kriging Simulation A detailed description of Kriging was already given in Section 2.3. The corresponding predictor given in Section 2.3.2 is an *estimator* of a function value of a new sample x. In this context, estimation and simulation have quite different goals. Estimation aims to produce *a value* that is as *close to the true value* as possible. In contrast, simulation should produce *a set of values* whose moments are as *close to the moments* of the true data as possible [142].
Hence, the simulation of a Kriging model produces realizations of a Gaussian process. The mean and covariances of these realizations depend on the model configuration and the training data. Surprisingly, the predictor is derived from the same model and data but is not guaranteed to have the corresponding mean and covariances [69]. Since the goal is to make a good, isolated guess, estimation produces a smoothed landscape. While high frequency changes could be ignored by an estimator, the simulation is able

to reflect them [69].
Approaches for simulation of Kriging models are, e.g., decomposition approaches, the spectral method, or the turning bands method [69]. Since it is simple to compute and straightforward to implement, we use an approach that is based on the computation of the square root of the covariance matrix [69]. To produce the simulation, we first need to specify a set of n_{sim} simulation samples $X_{\text{s}} \subseteq \mathcal{X}$. Then, the correlation matrix $\mathbf{K}_{\text{s}}$ for this data set is computed. The matrix is decomposed with

$$\sigma^2 \mathbf{K}_{\text{s}} = \mathbf{C}_{\text{s}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\text{T}},$$

where $\mathbf{C}_{\text{s}}$ is the covariance matrix with its corresponding eigen-decomposition into eigenvectors $\mathbf{U}$ and diagonal eigenvalue matrix $\mathbf{\Lambda} = \text{diag}(\boldsymbol{\lambda})$. The square root of $\mathbf{C}_{\text{s}}$ is

$$\mathbf{C}_{\text{s}}{}^{1/2} = \mathbf{U}\text{diag}(\lambda_1^{1/2}, ..., \lambda_{n_{\text{sim}}}^{1/2})\mathbf{U}^{\text{T}}.$$

It is used for the simulation with

$$\hat{\mathbf{y}}_{\text{s}} = \mathbf{1}\hat{\mu} + \mathbf{C}_{\text{s}}{}^{1/2}\boldsymbol{\epsilon}. \tag{9.9}$$

Here, $\hat{\mathbf{y}}_{\text{s}}$ are the unconditional simulated values for the samples X_{s}, and $\boldsymbol{\epsilon}$ is a vector of independent random samples with $\epsilon_i \sim N(0, 1)$ and $i = 1, ..., n_{\text{sim}}$.

Kriging Conditional Simulation While the unconditional simulation reproduces the moments of the training data, it does not reproduce the training data itself. To mend this, the simulation can be conditioned with the training data. This ensures that the simulation reproduces the training data exactly, i.e., $\hat{\mathbf{y}}_{\text{s}} = \hat{\mathbf{y}}$ if $X_{\text{s}} = X$. The resulting conditional simulation is closer to the estimation, but still has twice its error [142, 69].
Lantuejoul [156] gives a plausible example to demonstrate that the conditional simulation is useful despite its larger error. Figure 9.17 depicts depth measurements for an undersea cable. A Kriging model is built with the depicted data points and used to estimate the true depth. Furthermore, a conditional simulation is performed.
Whether either of them should be used depends on the purpose of the analysis: If the goal is to have a good idea of the true depth at a specific location, e.g., because a diver has to check the cable at that location, then the estimation is preferable. But if the length of the cable needs to be estimated before production starts, the estimation would grossly underestimate its length (due to smoothing). The conditional simulation would be a better basis for that purpose. It needs to be stressed that the depicted estimations and simulations are different aspects produced by the same instance of a Kriging model.
While approaches that are more efficient are possible, we decided for a straightforward conditional simulation approach [240]. We need three correlation matrices: The correlations of the training data $\mathbf{K}$, the simulation samples $\mathbf{K}_{\text{s}}$, and the cross-correlations

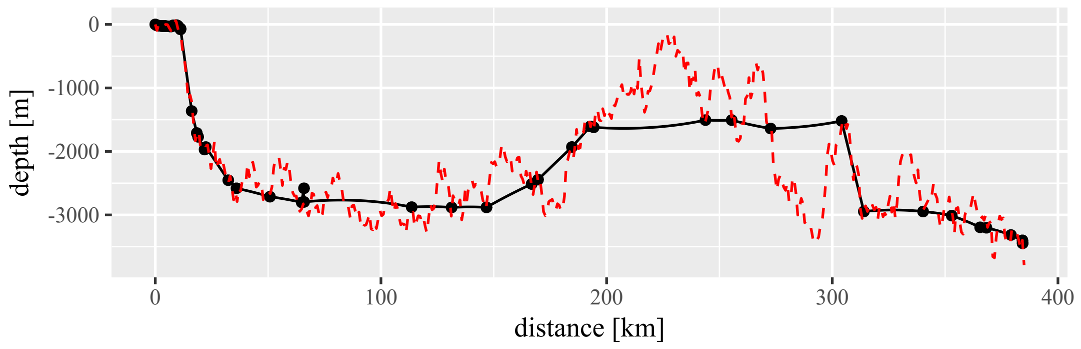


Figure 9.17: *Undersea cable depth estimation (black line), conditional simulation (dashed red line), and the given data (black dots). Based on the example presented in [156]. Slightly modified from [268].*

between simulation samples and training data $\mathbf{K}_\mathrm{x}$. Then, the correlation matrix required for the conditional simulation is [240]

$$\mathbf{K}_\mathrm{cs} = \mathbf{K}_\mathrm{s} - \mathbf{K}_\mathrm{x}\mathbf{K}^{-1}\mathbf{K}_\mathrm{x}{}^\mathrm{T}.$$

Similarly to Eq. (9.9), $\mathbf{C}_\mathrm{cs} = \sigma^2\mathbf{K}_\mathrm{cs}$ is used to produce the conditional simulation

$$\hat{\mathbf{y}}_\mathrm{cs} = \hat{\mathbf{y}} + \mathbf{C}_\mathrm{cs}{}^{1/2}\boldsymbol{\epsilon}. \tag{9.10}$$

The vector $\hat{\mathbf{y}}$ results from the predictor in Eq. (2.4).
It can be easily checked that this simulation is conditioned by the training data. If training and simulation samples are identical, then $\mathbf{K}_\mathrm{s} = \mathbf{K}_\mathrm{x} = \mathbf{K}$. Hence, $\mathbf{K}_\mathrm{cs} = \mathbf{0}$ and Eq. (9.10) results into $\hat{\mathbf{y}}_\mathrm{cs} = \hat{\mathbf{y}}$.

Test Function Generator We describe the proposed test function generator based on non-conditional or conditional simulation of Kriging models in Algorithm 6.
We distinguish two cases. Either, an existing data set is used, or some experimental design procedure is used to generate samples in the search space that are evaluated with the expensive objective function (Line 2 - Line 7). In practice, a mix of both is possible, e.g., when a rather small existing data set is augmented by additional observations. The data is used to train a Kriging model (Line 8). Afterwards, the simulation samples are created (Line 9), which are the samples at which the model is simulated. For each of the n_test test functions that are created, a separate simulation is performed (Line 11), e.g., by specifying different random number generator seeds. If a simulation covers the whole search space, then the resulting simulated values are the test function. Clearly, this may only occur if the space is discrete and sufficiently small. If the search space is not covered completely, then we require an additional step that interpolates between the simulation samples (Line 15 in Algorithm 6). To guarantee that the interpolation step actually reproduces the training data in the conditional simulation case, it

Algorithm 6 Simulation-based test function generation, as proposed in [268].

```
1: Given: number of training samples $n$, simulation samples $n_{\text{sim}}$ (usually $n_{\text{sim}} \gg n$), the number of required test functions $n_{\text{test}}$, and (optionally) the expensive real-world objective function $f(x)$.
2: if $f(x)$ is available then
3:     Create $n$ samples $X = \{x^{(1)}, \ldots, x^{(n)}\}$;                ▷ Training samples
4:     $y_i = f(x^{(i)}) \quad \forall\, i \in \{1, ..., n\}$;                ▷ Expensive training observations
5: else
6:     User provides data set $\{X, \mathbf{y}\}$;
7: end if
8: $M = \texttt{model}\ (X, \mathbf{y})$;                ▷ Train Gaussian process model
9: Create $n_{\text{sim}}$ samples $X_{\text{s}} = \{x^{(1)}, \ldots, x^{(n_{\text{sim}})}\}$;                ▷ Simulation samples
10: for $j = 1$ to $n_{\text{test}}$ do
11:     Create (un)conditional simulations $\hat{\mathbf{y}}_{\text{s}}^{(j)}$ with Eq. (9.9) or Eq. (9.10);
12:     if $X_{\text{s}} = \mathcal{X}$ then                ▷ Simulation covers the whole search space
13:         Simulation $\hat{\mathbf{y}}_{\text{s}}^{(j)}$ is the required $j$-th test function;
14:     else
15:         Provide $j$-th test function as interpolation of simulated samples using Eq. (2.4): $\hat{y}_{\text{s}}^{(j)}(x) = \hat{\mu} + \mathbf{k}_{\text{s}}{}^{\text{T}}\mathbf{K}_{\text{s}}{}^{-1}(\hat{\mathbf{y}}_{\text{s}}^{(j)} - \mathbf{1}\hat{\mu})$;
16:     end if
17: end for
```

is useful to ensure that $X \subset X_{\text{s}}$. The interpolation step has to be used with care. It essentially relies on estimation and may violate the ruggedness criterion (C.6). However, the simulation interpolates n_{sim} samples, which should be more numerous than the n training samples. Therefore, the smoothing problem should usually be less severe for simulation, compared to pure estimation.
With respect to our criteria, we can summarize the following advantages.

(C.1) Difficulty: Since the simulations are based on real-world data, sufficient difficulty of the function should be given. By avoiding smoothing, the simulations avoid the creation of test functions that are too simple.

(C.2) Diversity: The simulations are random realizations of Gaussian processes. An infinite number of realizations can be sampled at random. Sets of such test functions are naturally diverse, yet reproduce the features of the real-world data. Compared to the parameter variation approach (see Section 9.3.2), this is a more principled way of generating diverse functions. We are not relying on user specified bounds on some measure of similarity between problem instances. A simulation-based test function is guaranteed to reflect the properties of the learned Gaussian process model. In contrast, some arbitrary change to model parameters would not yield that guarantee

(C.3) Flexibility: The desired flexibility is inherent to Gaussian process models. By adapting the kernel functions or other aspects of the model, quite different data sets can be represented.

(C.4) Relevance: Due to being based on real-world data, and due the prevention of smoothing, the resulting test functions should reflect the behavior of the real-world optimization problem. In fact, both the behavior (non-conditional) and data itself (conditional) may be reproduced by the simulation test functions. This is in contrast to other Kriging-based test function generators such as the Krigifier, which was not intended to use real-world data.

(C.5) Cost: Since the underlying model can be trained with a static data set, further evaluations of a potentially expensive objective function are not required.

(C.6) Ruggedness: Avoiding some form of smoothing by the model is one core reason to rely on simulation rather than estimation.

Correspondingly, the following potential drawbacks have to be considered.

(C.1) Difficulty: It is hard to interpret the difficulty and behavior of simulation-based test functions. In particular, the number, value, and location of the optima are unknown. If required, such features have to be approximated. This is in contrast to many classical test function generators, such as BBOB, where these features are usually known.

(C.2) Diversity: Conditional simulations may produce test functions that have little diversity if the trained Kriging model fits the data very well. The model estimates low variances, and all realizations of the simulation are nearly identical (Fig. 9.18 shows an example). This could occur if the modeled problem is extremely simple (e.g., linear) or if the training data set is fairly large. To detect such a case, the estimated uncertainties at the simulation sample locations X_s can be compared against a threshold value. In many use-cases, sparsity of training data due to high costs of evaluation would render this issue unlikely. Furthermore, unconditional simulation is not affected by this issue.

(C.3) Flexibility: The flexibility of Kriging has an inherent drawback. Increasing the number of model parameters leads to a more flexible model, but it also increases the complexity of the MLE-based training procedure.

(C.4) Relevance: The training data may introduce bias. If insufficient data is collected, the model may not learn the problem structure. This necessarily affects all observational, data-driven test function generators.

(C.5) Cost: The evaluation cost is in general a strong point of test functions based on surrogate models. Still, if the number of training samples n and/or the number of simulation samples n_sim become very large, the computational effort and/or memory requirement may grow too large.

(C.6) Ruggedness: It is important and difficult to specify a good value for the number of simulation samples n_{sim}. While large values may be costly (see C.5 above), rather small values of n_{sim} may lead to excessive smoothness, due to the additional interpolation step. Fortunately, this smoothness issue is less severe than in the estimation case. The number of training samples n is restricted by the cost of evaluation of the expensive, real-world objective function $f(x)$. The parameter n_{sim} is not restricted by this cost.

In addition, the model selection and configuration may introduce bias. Competing SMBO algorithms that employ different models may not be reasonably compared on simulation-based test functions, especially if only one of them uses Kriging. It is likely that the SMBO algorithm with the Kriging model would receive an unfair advantage.

9.3.4 Illustrative Examples

We demonstrate the implications of our test function generator with a simple and intuitive example. Despite the focus of this thesis, we decided against a discrete test case, mostly because continuous examples can be easily visualized. Hence, we chose the simple, real-valued, one-dimensional function

$$f_{\text{example1}}(x) = \exp(-20x) + \sin(6x^2) + x, \tag{9.11}$$

with $x \in [0, 1]$. Our goal was to show how estimation, simulation, and the underlying ground truth $f_{\text{example1}}(x)$ interact.
Hence, a Kriging model was trained with $n = 6$ uniform random samples evaluated with f_{example1}. A second Kriging model was trained with $n = 11$ equidistant samples, to illustrate the effect of a better model fit. Both were simulated with $n_{\text{sim}} = 100$ simulation samples. In each case, twenty potential test functions were generated, ten each with unconditional and conditional simulation, as shown in Figure 9.18.
Of course, the training data was not reproduced by the unconditional simulations. Only their behavior was similar to the original function. Besides random differences, no clear distinction between the two models ($n = 6$, $n = 11$) was observed for the unconditional simulations.
The conditional simulations had a quite different behavior. As expected, they reproduced the training data. The larger the distance to some training observation became, the more deviation occurred between the simulations and the original function. This behavior is linked to increased uncertainty estimates (cf. Section 2.3.3). For the equidistant samples, the improved model quality led to conditional simulations that were nearly identical. In lights of these observations, it should be clear that if a diverse set of problems from the same class are of interest, unconditional simulation is a better choice. Conditional simulations are better if the behavior of a specific problem instance is of interest.

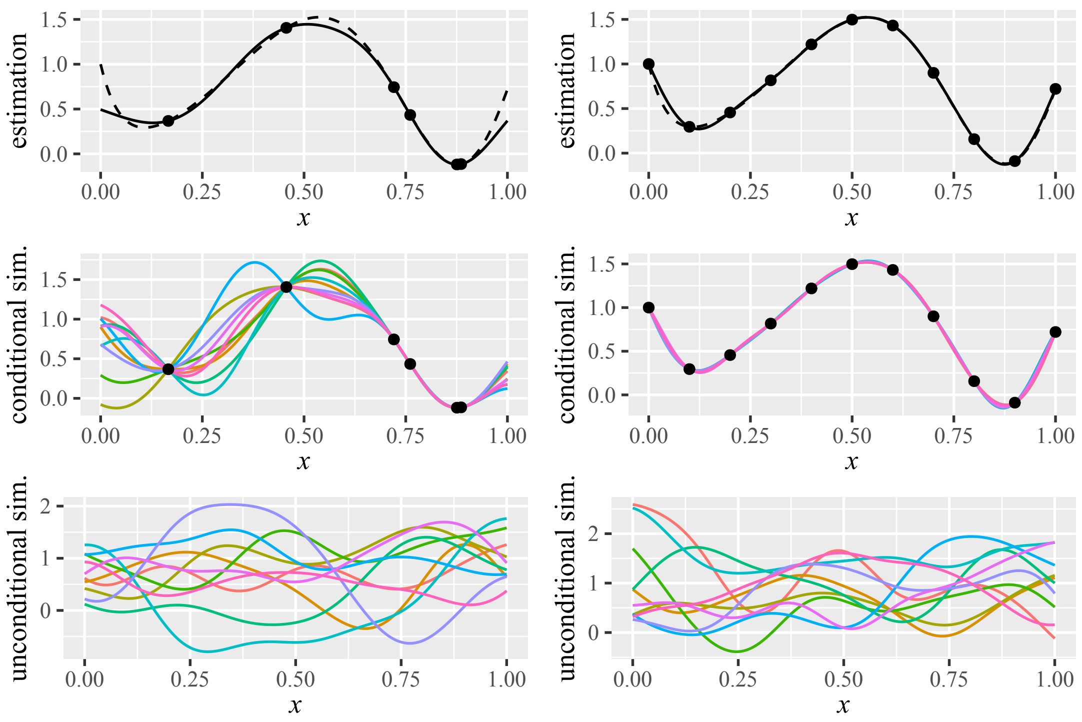


Figure 9.18: *Top: The function $f_{example1}(x)$ (dashed, black) and the Kriging estimation (solid, black) based on training data (dots). Middle: 10 realizations of a conditional simulation. Bottom: 10 realizations of an un-conditional simulation. Left: trained with six uniform random samples. Right: eleven equidistant samples.*

To explain the importance of the number of samples (n and n_{sim}), Fig. 9.19 shows a similar example. Here, we chose the function

$$f_{\text{example2}}(x) = x + \sum_{i=1}^{10} \sin(72(c_{1i} + 0.5)(x - c_{2i})), \tag{9.12}$$

with c_{ji} representing uniform random samples from the interval $[0, 1]$. This function is clearly quite nonlinear and hence more difficult to model than $f_{\text{example1}}(x)$.

An insufficient number of training samples $n = 5$ clearly led to a poor model. Correspondingly, both estimation and simulation instances were much smoother than the objective function. However, with an increasing number of training samples, both improved. Similarly, the simulations strongly depended on n_{sim}. In particular, the simulations with $n_{\text{sim}} = 2$ were excessively smooth. It can be concluded, that both the training data set size n and the number of simulation samples n_{sim} need to be sufficiently large to produce a representative test function based on simulation.

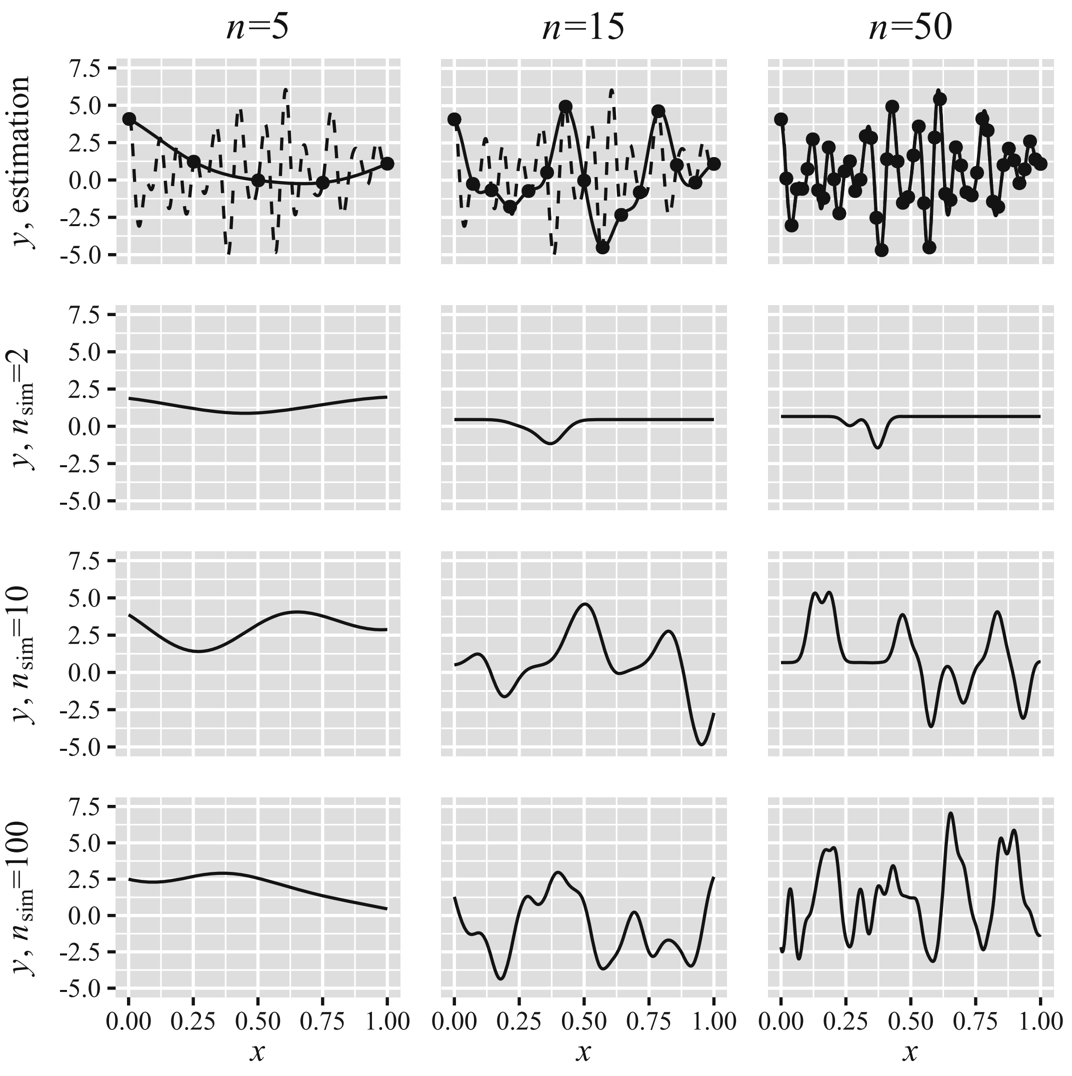


Figure 9.19: *Estimation and unconditional simulation instances of a Kriging model for $f_{example2}(x)$ (solid black lines). Each segment depicts a different number of training samples n and simulation samples n_{sim}. The dashed line is the true function $f_{example2}(x)$. Black dots are training samples.*

9.3.5 Protein Landscape Application

Data and Problem This section presents a real-world application of the simulation-based test function generator. To that end, we chose an openly available data set from the field of computational biology [214, 37]. The data set contains the fitness values of all DNA sequences of length ten. Here, fitness refers to the affinity to a fluorescent target protein: allophycocyanin. The data set has previously been used for the assessment of evolutionary algorithms, using a finite state machine model [215].

Candidate solutions x are DNA sequences with $m = 10$ bases, i.e., strings with ten letters that are either `A`, `C`, `T`, or `G`. The fitness $f_{\text{affinity}}(x)$ is the result of the complex measurements described by Rowe et al. [214], and is part of the data set. It has to be maximized.

Test Function Generation We tested different configurations of the model and the training data.

- We generated two individual training data sets. The first was selected from the complete space (*train-complete*). The second was selected from a subspace of 1024 sequences (*train-subspace*). For the subspace, the last 5 elements of each sequence were arbitrarily fixed to `ACGTA`. In both cases, $n = 100$ sequences x and their fitness $y = f(x)$ were selected (randomly, uniformly) to emulate an expensive objective function.

- Two models were trained with the corresponding data, a Kriging model, and a random forest model. For Kriging, we chose an exponential kernel $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x'))$ based on the Hamming distance. The Hamming distance also proved to yield good results in other studies [271] and has the additional advantage of low computational costs. The Hamming distance was also used in the original study that introduced the underlying data set [214]. For the random forest regression model, we decided to use the `randomForest` package in R with default settings. We denote Kriging with KR and random forest with RF.

- We analyzed the derived test functions in the complete space (*test-complete*) and the subspace (*test-subspace*).

- For the KR model, we investigated three different types of test functions. Firstly, a single instance was created with the predictor (*estimate*). Secondly, $n_{\text{test}} = 10$ test functions were created by unconditional simulation. Since the complete data set is available, there was little motivation for conditional simulation. The idea was to create test functions that show similar behavior as the real protein fitness landscape. For the subspace, all $n_{\text{sim}} = 1024$ sequences were simulated (*simulate-all*). For the complete space, simulating all 4^{10} sequences was computationally infeasible. Hence, $n_{\text{sim}} = 1000$ sequences were simulated, and the test function was generated via the interpolation step (*simulate-interpolate*).

All resulting test instances are summarized in Table 9.2.
Due to the discrete, small search space, brute force was used to estimate the global optimum of each generated test function landscape. The number of local optima was also determined by brute force, i.e., the number of sequences with neighbors of equal or worse fitness. Here, Hamming neighborhood was employed. That means, the neighbors of a sequence were defined as all sequences that differed in exactly one element from the original sequence.

Table 9.2: *Test function instances based on protein-data.*

ID	search space	model	test function generation	# of instances
0			real-world data	1
1	complete	KR	simulate-interpolate	10
2	complete	KR	estimate	1
3	complete	RF	estimate	1
7	subspace	KR	simulate-all	10
8	subspace	KR	estimate	1
9	subspace	RF	estimate	1

Landscape Analysis: Complete Space To understand the behavior of the derived test functions, we first report landscape characteristics of the test functions based on simulations in the complete, unrestricted search space. Rowe et al. [214] report a correlation length of roughly 4.5 for the whole search space. They estimated this value by calculating the auto-correlation of random walks in the fitness landscape. Their result was nicely reproduced by our KR model, which was trained with just 100 samples: The correlation length (the reciprocal of the kernel parameter θ) determined by maximum likelihood estimation during model training was 4.48. Another good match was the reported fitness distance correlation, which was -0.32 for the original data. We estimated a mean of -0.37 and a standard deviation 0.09 for the ten simulations.
Unfortunately, we also observed a strong mismatch between the simulation-based test functions and the original fitness landscape. Rowe et al. [214] report that the data set has 6805 local optima. The simulations only produced between 5 and 50 local optima. Hence, the simulation-based test functions seemed to be a poor representation of the underlying optimization problem. This problem was clearly linked to the last step of the test function generator: interpolation. Essentially, the interpolation step smoothed through local optima that would have been present in a pure simulation of the complete search space. Since the number of simulated samples $n_{\text{sim}} = 1000$ was much smaller than the number of local optima in the real landscape, the resulting test functions were necessarily much easier than the real problem.
As expected, the same problem occurred for the estimation-based test functions. The KR estimation was even worse with just 4 local optima. The estimation of the RF model was slightly better with 32 local optima.

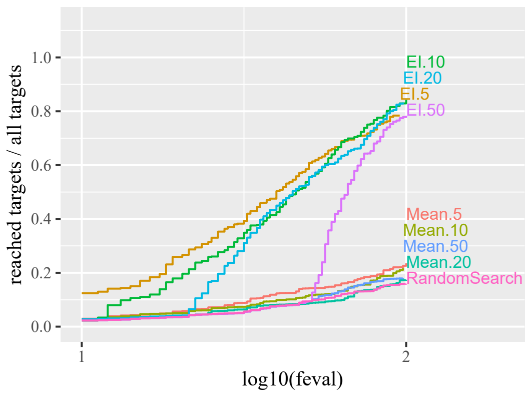
Landscape Analysis: Subspace We avoided the interpolation step by restricting the modeling and the analysis to a subspace of just 1024 DNA sequences. Then, estimation with KR or RF yielded 2 local optima. The KR simulation produced 10 to 19 local optima. The real landscape has exactly 16 local optima in the subspace. Without the interpolation step, the earlier violated criteria (C.1, C.4, C.6) seemed to be satisfied for the simulations in the subspace.
In theory, we could repeat the same simulation experiment (without interpolation) with the complete search space, but that is computationally infeasible: Even just storing

the required 4^{10} x 4^{10} covariance matrix is prohibitive. A more efficient simulation technique would be required.

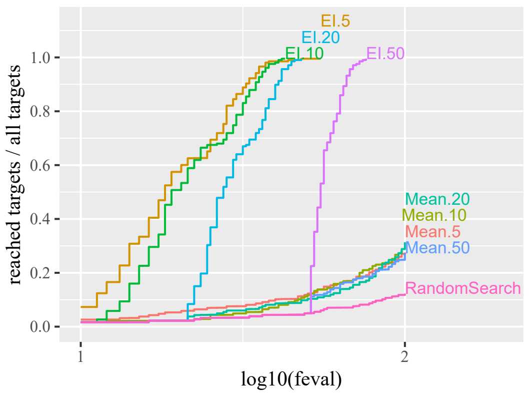
Performance Analysis In additional experiments, we used the simulation-based test functions to evaluate the performance of optimization algorithms. We restricted the optimization performance tests to the five-base subspace to avoid problems with the interpolation step. Ten different simulation-based test functions were created, and each algorithm was run twenty times on each function, resulting into 200 replications.
In addition, we generated a single estimation-based test function with KR and another one with RF. To provide a ground-truth, the tests were also repeated on the actual objective function, i.e., directly using the real-world data. Since these cases (KR estimation, RF estimation, real-world data) only involved a single test function instance in each case, all 200 replications were spent on that single instance.
We tested a variant of CEGO as described in Algorithm 4, with the following settings.

- **initialization**: n_{init} uniform random samples, with $n_{\text{init}} \in \{5, 10, 20, 50\}$.
- **model**: KR with Hamming distance.
- **optimizer**: Due to the small search space, the infill criterion was optimized by brute force.
- **infill criterion**: Either expected improvement (EI) or the predicted mean of the model (Mean).
- **termination:** The algorithm terminated after 100 objective function evaluations.

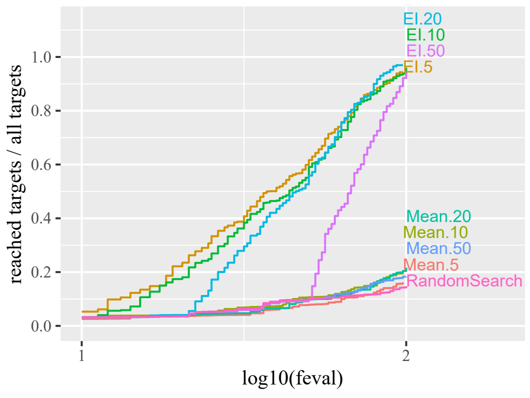
Random search was used as a baseline comparison to the model-based approaches. Hence, we compared 9 algorithms: random search and eight CEGO variants. Each CEGO variant corresponded to different configurations of the infill criterion (EI, or Mean) and $n_{\text{init}} \in \{5, 10, 20, 50\}$.
To estimate and compare performance over time, we used a set of target values and recorded the respective runtime required to reach these targets. Here, runtime implies the number of objective function evaluations. The global optimum $y_{\text{opt}} = f(x_{\text{opt}})$ of each test function was determined by brute force. Based on the determined optimum, the fitness gap was defined as follows fgap$(x) = f(x) - y_{\text{opt}}$. Evenly spaced targets for the fitness gap were specified on a logarithmic scale: tar$_{\text{fgap}} = 10^0, 10^{-0.2}, \ldots, 10^{-6}$. During each algorithm run, we recorded the runtime when a target was attained. We aggregated the resulting data by calculating the fraction of all targets reached at a certain runtime. The resulting curves (fraction of targets reached against runtime) are also called run length distributions [124], empirical cumulative distribution functions (ECDF) [112] or data profiles [184].
The results from our experiments are depicted in Fig. 9.20. The figure indicates that using the EI infill criterion was crucial for good performance of CEGO. The Mean infill criterion produced performance close to the random search baseline. Results at



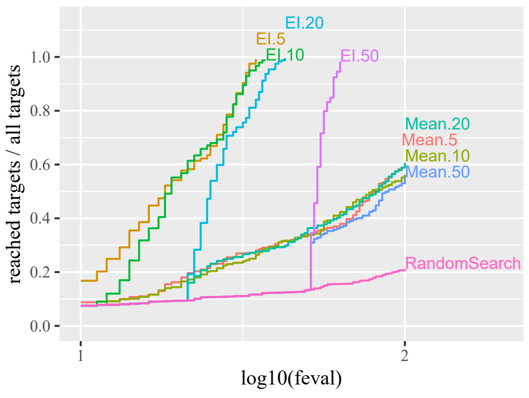
(a) *unconditional Kriging simulation*



(b) *Kriging estimation*



(c) *real objective function*



(d) *random forest estimation*

Figure 9.20: *Logarithmic ECDF plots for comparing optimization algorithms in four test cases. The labels inside the plots indicate the configuration of the employed algorithm, that is, whether EI or the predicted mean were used as an infill criterion and the size of the initial design. The x-axis depicts the logarithm of the number of fitness function evaluations (feval). Lines end when no further improvement is recorded.*

exactly 100 evaluations were mostly insensitive to the initial design size n_{init}. Earlier in the runs, smaller n_{init} yielded superior results.

The strong positive effect of EI could be attributed to the ruggedness of the problem and the large number of local optima. The explorative properties of the EI infill criterion seem to have helped to deal with these difficulties.

Furthermore, the runs with the Mean criterion were more distinct from the random search baseline in case of estimation. This is no surprise, because these functions were clearly smoother than the simulations and the real objective function. Essentially, the performance of all algorithms was overestimated by the estimation-based test functions. In contrast, the simulation-based runs were much closer to the results on the true objective function.

This stresses that simulation approaches are more suited to satisfy the required criteria of test function generators, especially with regard to difficulty (C.1), relevance (C.4),

and ruggedness (C.6). This result was independent of the chosen model. RF and KR estimation both produced similar test results. Both seemed to capture similar behavior from the training data. There were only small differences; Algorithm runs on the RF instances seemed to converge slightly faster.
While the simulation-based test functions were close to reproducing the real-world problem, small differences remained. The goal of the simulation-based test functions was not to produce exact surrogate functions for the true, real-world problem. Rather, it was desired to create test function instances with similar behavior. Still, the selected kernel of the KR model may be further improved, to generate simulations that are more accurate. For instance, the isotropic nature of the kernel may not have been a perfect choice. Rowe et al. [214] report that bases at the start of a sequence have larger impacts than the bases later in the sequence. Hence, it would be promising to use an anisotropic kernel, which allows learning the importance of each base. This additional accuracy could produce better results but would also increase the computational cost of the model.

9.3.6 Summary

The core question of Section 9.3 is answered as follows.

Answer 9.4. Generating test functions. How can we generate test functions that fulfill requirements such as difficulty and real-world relevance?

> We proposed to use Kriging simulation to this end. Kriging simulation can generate test functions that emulate the behavior of real-world optimization problems. The simulation-based approach is able to produce difficult (C.1), diverse (C.2), flexible (C.3), relevant (C.4), and inexpensive (C.5) test problems that may avoid harmful smoothing (C.6).

Simple examples were used to illustrate the idea. A protein sequence data set was employed to demonstrate the implications of a complex real-world problem. This demonstration showed that, while not without hazards, simulation does present a clear advantage over estimation-based approaches. The simulation-based test functions can *reflect the behavior of the real-world problem rather than just the data itself.*
We strongly recommend using a simulation approach if data-driven test functions are desired. Simulation is a principled way of generating diversity and avoiding undesirable smoothness. Still, simulation-based test functions are not supposed to replace classical test function sets. These test function sets do have merits, e.g., their properties and behavior are well understood. If an algorithm were assessed without any specific application in mind, a mix of both would be ideal. If an algorithm were assessed with the desire to determine performance on problems with specific features (e.g., separability, unimodality), classical test functions would probably be preferable. Contrarily, if an algorithm is assessed in the context of a specific real-world application (i.e., C.4 is important), a simulation-based test function generator should be preferred.

In the latter case, if performance on a class of *problems with similar behavior* as the real objective function is of interest, unconditional simulation would be more appropriate. Conditional simulation would be more appropriate if the performance of an algorithm for *a specific problem* is of interest.
Future research should focus on the main problem of the simulation-based function generator: dealing with large rugged search spaces. These remain a challenge because they require many simulation samples, which leads to intractable computational requirements. Especially for discrete search spaces, Gaussian Markov Random Field models [219] may be a promising solution. Here, the Markov property induces sparsity in the inverse of the covariance matrix, which may be exploited to deal with large sample sizes.
Furthermore, a method for determining the required number of simulation samples n_{sim} is desirable. As the examples in Section 9.3.4 showed, a small n_{sim} provides satisfying results if the problem itself is rather smooth. Clearly, the smoothness of the model is specified once the kernel parameters are determined. It should be possible to estimate the required n_{sim} from the covariance structure of the model.
Finally, simulation is not limited to Gaussian processes. It may be beneficial to include different stochastic process model types into the analysis, especially, to reduce the potential bias introduced by a specific model choice.

9.4 Conclusions

The preceding sections show that SMBO methods like CEGO are not limited to vector-valued, discrete search spaces, such as strings or permutations. While it is a challenge to define adequate distances for graph structures, such as trees, the resulting kernels enable to learn the mapping from complex representations to a fitness value. Especially for trees, phenotypic distances seem to be a very promising idea. Once such a phenotypic distance is defined, a modeling algorithm may become completely independent of the underlying solution representation.
The results on hierarchical search spaces point in a slightly different direction. Here, the idea is not to derive some similarity based on the phenotype. Rather, the genotype distance is reinforced with expert knowledge about the modeled variables. While this is a less generalizable approach, the resulting kernels seemed to perform extremely well.
These examples form interesting and challenging test cases for our methods. Still, SMBO algorithms require broader, real-world relevant benchmarks. Our proposed simulation-based test function generator may provide such benchmarks. We suggest that more data sets and examples should be made available to the research community, leading to relevant, easily reproducible test problems.

Part III

Closure

Chapter 10

Final Evaluation and Outlook

A particular challenge in the development of SMBO algorithms is their interdisciplinary nature. They combine methods from different fields, including statistics, machine learning, evolutionary computation, and classical optimization. Hence, it is no surprise that this thesis touched on aspects from all of these areas. This chapter attempts to consolidate all aspects that have been discussed in this thesis, giving a retrospect of the various lines of research.

Just like a good riddle, research is often a reward in itself rather than just a means to an end. Still, we would be glad if this work provided some interesting solutions to practitioners as well as some starting points for future research. To that end, this chapter will contain advice to practitioners as well as an outline of potential future research.

10.1 Summary

The core motivation of this thesis was to extend SMBO to combinatorial and discrete optimization problems. As a foundation, we first outlined the core concepts of EAs, SMBO, Kriging, and EGO in Chapter 2. These were four of the main ingredients involved in this thesis. In providing these foundations, we necessarily took a perspective from the field of continuous optimization. This is where most of these ideas originated.

To extend on these origins, Chapter 3 dealt with the more recent developments in the context of discrete or combinatorial problems. Here, we classified discrete surrogate modeling approaches into six strategies. We identified feature extraction, mapping, and kernel-based models as the most promising strategies, due to their applicability to a wide range of problems. Among them, kernel-based models stand out, because they are the basis of models like Kriging.

Kernels are in fact a widely discussed topic, involved in numerous methods from the field of statistics and machine learning. For the sake of clarity, and to lay some further foundation for later chapters, we provided some terminology and definitions in Chapter 4.

With the foundations given, Chapter 5 investigated the extension of Kriging-based optimization via EGO to combinatorial, discrete, or mixed optimization problems. Here, the intuitive idea was to replace the continuous kernels in the models with more appropriate kernels for discrete spaces. At a first glimpse, this straightforward approach could have settled the problem. However, it turned out that things are not that simple.

Importantly, the phrase *appropriate kernel* is rather vague. It immediately led us to two questions. Firstly, are our kernels missing some crucial properties? Secondly, what do we do if several kernels are available?

With regard to the latter question, this thesis showed repeatedly that we rarely consider only a single kernel. Rather, there is usually a set of potential kernels available. Hence, it is of paramount importance to *select* kernels, or to *combine* them in some way. To that end, this thesis discussed various methods in Chapter 6, based on concepts related to correlation, likelihood, and cross-validation. Links to the literature on multi-kernel methods were provided. We observed in an experimental study that it was indeed beneficial to consider multiple kernels rather than sticking to a single one.

Another key aspect of kernels is their definiteness. Kernels are often desired to be PSD (e.g., correlation functions) or CNSD (distances). It may require some time and effort to determine whether a kernel or distance is definite. In some cases, this is possible, e.g., in case of the kernels proposed in Section 9.2. Unfortunately, not all kernels have such a simple structure. Thus, we suggested an empirical approach in Chapter 7. Essentially, a search procedure (directed or random) can be used to find cases where a kernel yields an indefinite matrix, thus proving that the kernel is indefinite. In this context, several distances for permutations were shown to be non-CNSD.

However, what is gained by this? Firstly, if no indefinite cases are discovered, we can simply assume that the kernel is definite and resume with standard kernel-based modeling approaches. This was the case for the majority of the investigated measures of distance for permutations.

For those kernels that were shown to be indefinite, we proceeded to consider methods that are able to produce a model *despite* the indefiniteness of the underlying kernel. This was discussed in Chapter 8. We borrowed various methods from the field of support vector machines, explained how they are applied to Kriging, proposed additional correction and repair methods, and discussed how this relates to non-stationary models. An experimental investigation showed that approaches based on spectrum transformations might improve Kriging models with indefinite kernels. However, choosing the right transformation and repair method remains difficult.

To summarize the steps so far, this thesis i) showed how kernels on discrete objects enable combinatorial optimization with Kriging models, ii) investigated how different kernels can be selected or combined, iii) proposed methods to determine whether individual kernels are definite, and iv) explained how Kriging models may incorporate indefinite kernels.

To stress the relevance of these deliberations, we considered more complex search spaces in Chapter 9. In particular, we investigated symbolic regression problems and proposed several new kernels for hierarchical search spaces. To enable benchmarks

with more real-world relevance in the future, we also proposed a simulation-based test function generator. In contrast to other data-driven test function generators, a simulation allows for a principled way to generate diverse test instances. In addition, simulations may avoid producing test instances that are too smooth. We illustrated the implications of this generator with a real-world data set.
This concludes the scientific contributions. As a final note, we would like to mention that most methods discussed in the experimental part of this thesis are available in the open source software package CEGO [263]. The CEGO package is developed for the statistical computing language R. It includes models, optimization algorithms, kernels, distances, and several other tools required to implement or use the methods discussed in this thesis.

10.2 Advice for Practitioners

Kernel and Model Selection In Chapter 6, we have dealt with various options for dealing with multiple kernels. Based on the experimental results, one conclusion seems obvious: While it may be hard to select the best multi-kernel method it seems to be important to use *any* multi-kernel method rather than just a single kernel.
However, multi-kernel methods are no magic bullets that automatically always produce the best model. They necessarily have to pay for their greater flexibility with a lowered accuracy on some specific problems. This follows from the fact that they first have to learn which kernel or kernel-combination works for a specific application. Necessarily, their efficiency deteriorates with an increasing number of individual kernels. The probability of favoring a sub-optimal kernel increases.
Hence, a further suggestion is that a-priori knowledge should be taken into account when possible. Any available knowledge about the problem to be solved should be used to reduce the number of kernels to decide on. If a single good kernel is deemed adequate for a certain problem (based on some expert knowledge or problem understanding), it makes little sense to invest additional effort into inventing other kernels, only to justify the use of some multi-kernel method. Similarly, if it is clear that some kernel is a poor choice for a problem, it makes no sense to include it in the model. Multi-kernel methods should be used when there is considerable uncertainty about which kernels work well.

Constructing New Kernels In the contrary to the multi-kernel considerations above, there may be practical problems where no good kernels are readily available. Then, a new kernel may have to be designed. In that case, how can new kernels be devised?
The first, obvious suggestion is to check the literature. While this may not always yield a ready-made solution, finding kernels for similar problems may be helpful: They may provide hints on a kernel's design. We used this approach successfully to develop a new, definite kernel for hierarchical search spaces (cf. Section 9.2).
It may also be helpful to check whether the literature describes evolutionary variation

operators for these cases. Kernels and variation operators encode very similar information; both consider some sort of neighborhood in the corresponding search space. This is, e.g., obvious for edit distances, since the corresponding edit operations are often used as variation operators. The close relationship between variation operators and kernels is also discussed by Lane et al. [155].

Another idea may be to follow the idea of convolution kernels [116]. It may be simpler to design kernels on parts of an actual candidate solution. These individual parts may then be combined in an overall kernel. Similarly, new kernels may also be generated in an automated way by employing genetic programming [95, 150].

If an idea for a kernel does not present itself, it may help to think of the problem in the context of the modeling strategies from Chapter 3. It may be easier to think of a mapping function, rather than a similarity function (kernel). Or else, an application expert may be able to derive meaningful features, and use those as the basis of the kernel or model.

Given the fact that indefinite kernels may be used successfully in various surrogate models (cf. Chapter 8), it may be questioned whether effort be spent on designing *definite* kernels. Alternatively, should a potentially more simple, yet indefinite kernel be preferred? The answer depends on the situation. If there already is a good idea of how to design a definite kernel, this should clearly be used, to avoid the additional complexities of modeling with indefinite kernels. Furthermore, it may be useful to consider the goal of the planned experiments. If a general solution approach is required, designing a definite kernel may be profitable. If only a solution to a specific problem is desired, an indefinite kernel may be perfectly fine.

Dealing with Indefinite Kernels One by-product of our investigations is that indefinite kernels are a very relevant challenge. Chapter 7 showed not only how we could use an empirical approach to investigate the definiteness of a kernel. Moreover, several of the tested distances turned out to be indefinite. In addition, all kernels used in the experiments with symbolic regression trees (Section 9.1) were discovered to be indefinite. In this latter case, the detrimental effect of this lack of definiteness became obvious during the analysis of the experimental results.

However, there are ways to compensate for a lack of definiteness. Chapter 8 discussed spectrum transformation that may be used to correct indefinite kernels. Here, the difficulty is to find the right configuration. In practice, we suggest making some initial tests with artificial test function. Such preliminary experiments may be a reasonable basis to decide on the right correction method, without spending any resources on evaluations with the expensive, real-world objective function.

If the more complex methods based on spectrum transformations are disregarded for some reason, there may be more simple alternatives. While results may not be optimal, a simple correction via the nugget effect at least provides a feasible model. The results from Section 9.1 also suggest that a combination of several kernels may render a model more robust against issues with definiteness.

10.3 Future Work

As demonstrated in this thesis, combinatorial or mixed data structures are not an impassable hurdle to SMBO methods. Yet, some obstacles remain to be eliminated. We limit this final section to issues that are closely connected to this thesis. For a wider view on open issues in this field, we refer to recent surveys [232, 26, 63].

Dimensionality Kriging is often reported to perform poorly for high-dimensional, real-valued search spaces. There is no strict upper bound, yet $m > 20$ has been used as an approximate threshold, e.g., by Forrester et al. [93]. The behavior of learning algorithms for high-dimensional data has also been linked to the underlying distance functions by Aggarwal et al. [2]. However, it is unclear how this applies to discrete, combinatorial search spaces with arbitrary distances or kernels. What distances and dimensionalities are problematic?
Moreover, what can we do to counteract these problems? Existing methods like feature selection are not necessarily applicable to complex data types, such as graphs. These questions definitely require further investigation. At the same time, recent results for high-dimensional permutation problems investigated by Smith et al. [235] seem to be quite encouraging.

Indefinite Kriging While Chapter 8 outlined different approaches for Kriging with indefinite kernels, it is still unclear what specific configuration should be chosen in practice. It may be difficult to decide what type of spectrum transformation is reasonable, or which repair procedure should be used. The large number of potential configurations presents an optimization problem in itself.
Of course, preliminary tuning experiments may be a reasonable solution. Yet, this may not always be feasible due to the computational costs. A more formal approach could lend a theoretical foundation to the discussed methods.

Visualization A common issue that we encountered throughout this thesis is visualization. Often, visual cues are an important catalyst for a more formal analysis. They can help to facilitate an intuitive understanding of problems and algorithms. This was the main reason why we often resorted to simple, continuous examples, e.g., in Sections 8.3 and 9.3.4.
Visualization methods for combinatorial problems would hence be extremely interesting for researchers as well as practitioners. As a starting point, there already exist a few approaches for the visualization of combinatorial problem landscapes, such as barrier trees [89, 28]. It would be interesting to integrate information like the uncertainty estimate of a Kriging model into such visualizations.

Benchmarks and Test Functions The simulation-based test function generator described in Section 9.3 was noted to have two tightly connected problems. Firstly, we still need a reliable estimate of the required number of simulation samples n_{sim}. This

is critical to the quality of the generated test instances. Secondly, if the value of n_{sim} is large, we require efficient Kriging simulation techniques.
More generally, it would be desirable to have actual real-world problems available. At the very least, the release of data sets from expensive, discrete, real-world optimization problems would allow using these data sets as a basis for simulation-based test functions.

Model Comparisons Comparing a varied set of models in a broad experimental study may be of major interest to practitioners. At the same time, this would provide important insights for algorithm developers. Besides Kriging, future comparisons may include models like Random Forest, or graph models, such as Markov random fields. Of course, these comparisons strongly depend on a meaningful set of test functions. The development of a good benchmark framework is a major precondition for these comparisons.
It would be especially interesting to investigate problems from the field of chemical design. Here, autoencoders have seen some attention in recent years [185, 104, 108]. It is a compelling question, whether models and kernels should operate in the coded space, or whether they should directly operate in the chemical design space.

To summarize, we would like to highlight the three most pressing open issues. Firstly, from the methodological perspective, we need a better understanding of indefinite kernels and their correction in Kriging models. Secondly, from the experimental perspective, better test function generators are needed to be able to evaluate any methodological developments in the field. Finally, from the real-world perspective, we need to experiment with actual applications, to evaluate the relevance of test functions, as well as the quality of the developed methods. From the crowd of open issues, these strongly interrelated challenges should be a major focus of future research.

Appendix A

Analysis Tools

Experiments are central to the design and understanding of modeling and optimization algorithms. Not all aspects of an algorithm can be evaluated or discussed solely based on theoretical reasoning. Often, experiments are needed to verify or falsify the assumptions and expectations of an algorithm designer [20].
Still, experiments by themselves only yield data, not understanding. To understand how algorithms behave, we need a visual or statistical analysis procedure that summarizes the respective data from the experiments. This procedure should provide the means to understand the implications of the data.
The experiments described in this thesis often concern a comparison of many algorithm configurations on many problems. Therefore, we need tools that provide a sound and comprehensible analysis. In that context, we need to discuss two types of problem setups:

- Single Problem: This serves to determine how algorithms behave on a specific problem instance.
- Multiple Problems: An overall evaluation, considering all problems (or all problems of a certain class) at once, to give a general recommendation.

Further, we consider two different types of analysis.

- Single Factor: A distinct set of algorithms is compared, e.g., by performing pairwise comparisons.
- Multiple Factor: The effects of algorithm parameters are of interest. This may concern one or several algorithms.

This is very similar to the distinction made by Bartz-Beielstein and Preuss [25], who also distinguish between single and multiple problems. However, their second distinction is between single/multiple algorithms rather than factors. We make a slightly different distinction to stress the nature of the analysis methods that we require. Multiple factor experiments may require different analysis models than single-factor experiments, yet both may be understood to deal with multiple algorithms.

For a comprehensive discussion on good experimental practice and analysis methods, we refer to the thesis of Bartz-Beielstein [20]. We do not aim to cover this topic in much breadth; neither do we suggest any new methods. Rather, we intend to motivate the methods that we have chosen to analyze our experimental results.

A.1 Visual Analysis of Results

In case of a single factor experiment, we usually want to determine how the performances of algorithms compare to each other. One simple, intuitive approach is to use some form of visual analysis.
We mostly use box plots to visualize the results of our experiments. For instance, the box plot in Fig. A.1 shows results of two algorithms (A, B) that are clearly quite different (boxes do not overlap). A third algorithm (C) cannot be clearly distinguished from the other two (boxes overlap). Box plots also allow determining properties like variability or symmetry of the data. As such, they may indicate whether some group is reasonably close to being normally distributed [154]. Yet, they are best-suited for data sets with many samples [154]. Besides box plots, we occasionally employ other tools to visualize models or behavior of algorithms. These are described where they are used.

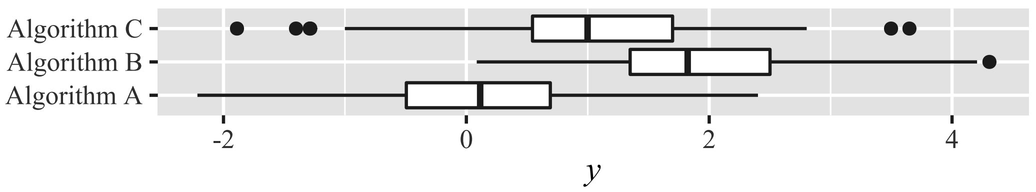


Figure A.1: *Example of a box plot. Here, the boxes indicate the first and third quartile, the bold line is the median, and the outer lines indicate the range of the data without outliers (dots).*

A.2 Statistical Analysis of Results

In the following, we will start with the slightly less complex case of comparing distinct algorithms (single factor), with single and multiple problems. We focus on non-parametric tests, which are frequently suggested for statistical testing in the fields of evolutionary computation and swarm intelligence algorithms [76], as well as machine learning [75].

A.2.1 Single Factor: Comparing Distinct Algorithms

To verify and support the visual analysis, statistical tests should be conducted to answer the following consecutive questions:

Question i) Are there any significant differences present in the data?

Question ii) If yes, which specific algorithms are different from each other?

Question iii) For each detected difference, which is the better algorithm?

Furthermore, our statistical tests have to fulfill at least the following needs:

a) The tests need to account for *multiple pairwise comparisons* between all algorithms.

b) The data contains *replicates*. Each algorithm is run multiple times on the same problem instance.

c) If we evaluate performance on multiple problems, we need tests that account for *blocked* data, where each block accounts for the problem instance.

One option would be to employ the Analysis of Variance (ANOVA) [177] to answer Question i), a post hoc Tukey's Honest Significant Difference (Tukey's HSD) test [177] to answer Question ii) and a comparison of group means (mean performance of each algorithm) to answer Question iii). However, all three methods are *parametric*. They make rather strong assumptions that are not always true [75]. Importantly, independence, normality, and homoscedasticity of the data or rather (in case of the ANOVA) the *residuals* of the corresponding model are often assumed. Roughly speaking, independence implies the outcome of one experiment (algorithm run) does not influence the outcome of another. Normality means that the data (or the residuals of a model) are normally distributed (in each group and block). Homoscedasticity implies that the variances in different groups (or blocks) are the same. Conversely, data is heteroscedastic if variances vary between groups (or blocks). In our context, algorithms are groups and test instances are blocks.
In the case of the experiments presented in this thesis, independence is usually satisfied since each observation is associated to a separate run of an optimization algorithm. The other two assumptions are not guaranteed to hold.
Let us first consider the normality assumption. The measured performance (e.g., attained objective function values) after a limited number of evaluations clearly has a lower bound (i.e., the optimum), and is often skewed. The data distribution may even be multimodal, e.g., if an algorithm is prone to converge to two or more distinct local optima. These facts contradict the assumption of normality because the normal distribution is unbounded, symmetric, and unimodal. Unless additional explanatory variables are available (e.g., the starting point of the optimization run), such properties of the data will necessarily also be present in the residuals of the analysis model.
The homogeneity of variance (homoscedasticity) assumption may also often be violated. For example, a well performing algorithm may have less variance than, e.g., an algorithm that performs similar to random search.
Therefore, we may need *non-parametric* tests, which are able to deal with the following types of data:

d) Data from different and unknown distributions.

e) Data with different variances between groups and/or blocks.

According to Büning and Trenkler [49], non-parametric tests have the advantage of requiring less (or weaker) assumptions. In addition, non-parametric tests are easier to compute, more efficient if parametric assumptions do not hold, and less affected by robustness problems [49]. Still, if parametric assumptions hold, parametric tests are necessarily more efficient [49]. We can confirm our choice of non-parametric tests with the gathered data, to avoid choosing non-parametric tests in a case where they are clearly not required. In general, we would rather err on the side of caution. That is, we prefer risking a loss of test power by choosing an unnecessary non-parametric method [65], rather than risking a false-positive result from an unwarranted parametric test.

We can check the model residuals for normality and homoscedasticity with some visual support. Examples for the corresponding analysis plots can be seen in Fig. A.2. Here, a linear model has been fit with two different data sets. For each set, two groups are compared. The upper two plots show a case where the data is normally distributed.

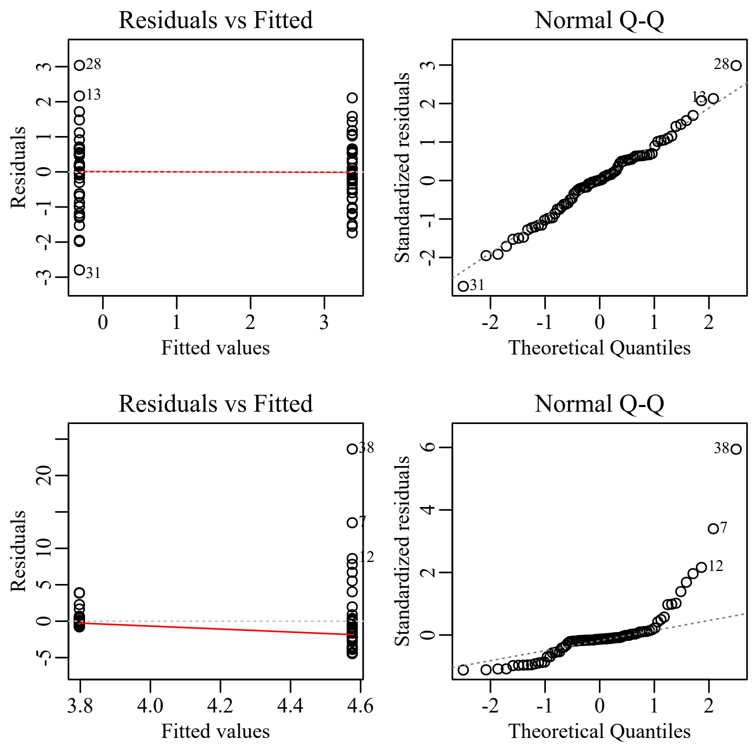


Figure A.2: *Two examples for analysis plots for checking assumptions of the linear regression (or ANOVA) model. The upper two plots show a case with normally distributed, homoscedastic data. The lower plot shows an example where the data is non-normal (here: exponential distributed) and heteroscedastic.*

Since the left-hand side plot has no obvious structure, the data seems to have equal variance between groups (homoscedastic). And since all points in the right-hand side *quantile-quantile* plot (QQ-plot) are close to the diagonal line, it can be concluded that the data is (approximately) normally distributed. The lower plots show an example where the data is both non-normal (here: exponential distributed) and heteroscedastic. The plots depict structure on the left and deviations from the diagonal on the right.
When it is confirmed that these assumptions do not hold, we use the following non-parametric tests. For the non-blocked case (single problem), a Kruskal-Wallis rank sum test [153] is first performed to determine an answer to Question i). According to Kruskal and Wallis [153], the test checks whether there "*is a tendency for observations in at least one of the populations to be larger (or smaller) than all of the observations together.*" If this is confirmed (i.e., the null hypothesis is rejected), a post hoc test procedure for multiple pairwise comparisons by Conover [66] can be used to investigate the pairwise differences between all algorithms, answering Question ii) [65]. The implementations of these tests are taken from the stats and the PMCMR R packages [205, 202]: `kruskal.test` and `posthoc.kruskal.conover.test`.
The same general procedure is used for the blocked case, but with the Friedman test and the post hoc Conover test [66, 65]. The implementations are taken from the PMCMR package [202]: `durbin.test` and `posthoc.durbin.test`[1].
The null-hypothesis of our tests is that there is no difference between the compared algorithm performances. We reject the null-hypothesis if the p-value produced by the test is smaller than the significance level $\alpha = 0.05$.
Note that even the non-parametric tests make assumptions. According to Conover [65], the Kruskal-Wallis test assumes that

1. *"All samples are random samples from their respective populations."*
2. *"In addition to independence within each sample, there is mutual independence among the various samples."*
3. *"The measurement scale is at least ordinal."*
4. *"Either the k population distribution functions are identical, or else some of the populations tend to yield larger values than other populations do."*

Here, Conover denotes the number of compared groups (in our case: algorithms) with k. For the Friedman test, Conover [65] mentions the following assumptions.

1. *"The b k-variate random variables are mutually independent. (The results within one block do not influence the results within the other blocks.)"*
2. *"Within each block the observations may be ranked according to some criterion of interest."*

[1] Since our datasets are usually complete block designs, the Durbin and Friedman tests are identical. The Durbin implementation is chosen because it allows for replications.

Here, b refers to the number of blocks (in our context: the number of test problems). The corresponding multiple pairwise comparison (post hoc) tests for Friedman and Kruskal-Wallis are based on the same assumptions.

In our case, none of these assumptions should be as problematic as the parametric assumptions. In addition, ties, which are quite likely to occur in discrete optimization, are not an issue [65].

Additional, more restrictive assumptions may be relevant, depending on the estimate of location that is used to interpret the test. This relates to the remaining Question iii), about how to determine which of two significantly different algorithms is the better one. Firstly, comparing group means may not be a good idea, since using the mean would again imply normality of the data. If the data is well-behaved, the median is a good alternative. However, this assumes that the compared distributions have the same shape.

Consider the following illustrative example. We compare two groups (a, b) and observe the respective outcome visualized in the histogram in Fig. A.3. The Kruskal-Wallis

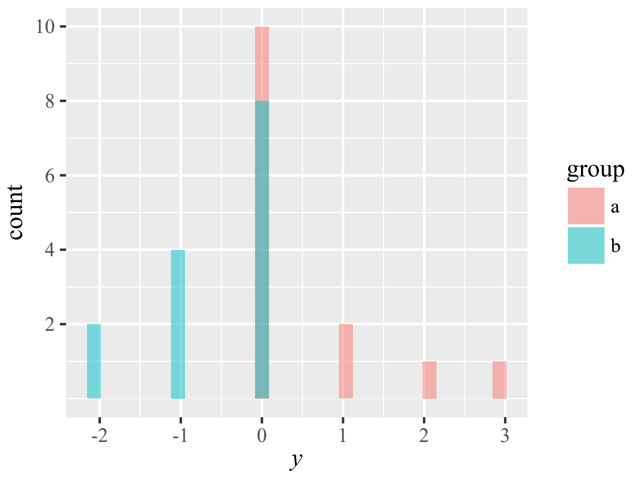


Figure A.3: *Counts of observed outcomes, separated by observed group. This figure illustrates potential problems if the median is used to estimate the location of a distribution.*

rank sum test would indicate that the differences between these groups are significant, with a p-value of approximately 0.0018. At the same time, the median of both groups is zero. Hence, the difference in location (according to the median) is also zero. This counterintuitive result is caused by the fact that the compared distributions have different shapes, being heavily skewed in opposite directions. This demonstrates why the median may be unsuited as an estimate of location.

Therefore, we may need a non-parametric estimate of location, analogous to the non-parametric tests. To that end, we could use, e.g., the *median of all pairwise differences*, which is called the Hodges-Lehman estimate of location [123]. We decided for the intuitive solution of using the basis of the employed tests: rank sums. That is, the sum of all ranked observations (ranked in each block, in case of the Friedman test) is computed. The difference of the sums is used to determine which algorithm is better. If the number of replicated evaluations differs, mean ranks are better suited. For the illustrative example in Fig. A.3, the Hodges-Lehman estimate and the difference in rank sums both indicate that group `(a)` yields larger observations than group `(b)`.

In some use cases, the above-described test procedures may yield a very large number of pairwise comparisons (e.g, the experiments in Chapters 6 and 8). This whole set of pairwise comparisons is often rather hard to read. For example, comparing 20 algorithms results into 190 individual comparisons, which can hardly be represented in an easily readable table. While test results may also be visualized as a graph (cf. [270]), these graphs tend to get equally hard to read once the number of comparisons is large. To produce a condensed result, the pairwise comparisons are used to generate rankings, using a form of comparison sorting. Each algorithm that is never significantly worse than any other algorithm receives the first rank and is removed from the list. Each remaining algorithm that is not worse than any of the other remaining algorithms receives the second rank and is removed from the list. This procedure is repeated until all algorithms are ranked. Clearly, it is easier to digest the resulting 20 ranks than a table with 190 comparisons.

As a drawback, the ranking does not convey the complete information from the tests; it only presents a summary of the results. Notably, a difference in the rank of two algorithms does not imply that the algorithms are evaluated to be different in a direct comparison. Hence, it is recommendable to report all pairwise comparisons as long as their number is small enough.

A.2.2 Multiple Factor: Algorithms and Parameters

In the multiple factor case, the analysis often has a different question to answer. Instead of comparing distinct algorithms, we are interested in the effects that different parameters have on the algorithm performance. For instance, we may want to know how different population sizes or mutation rates affect the performance of an EA.

One way to deduct this information is to use linear regression models for multiple variables [154]. However, just as in the single factor case, assumptions are a potential problem: we may have to deal with non-normality and heteroscedasticity. Even worse, non-parametric methods that relax these assumptions are not easily available for the multiple factor case.

We use two tools that help to adapt standard regression models: When the residuals seem to be non-normally distributed, we can try to transform the data, e.g., with the Box-Cox transformation [154]. Usually, the underlying idea of transformations is to apply a monotonic function to the data, so that the ranks are not changed yet the shape of the distribution becomes approximately normal. To deal with heteroscedasticity, we can try to employ a weighted least squares approach [154]. Intuitively, the weighted least squares approach gives larger weights to samples with a small variance.

Please note that transformations and weighted least squares are not a universal remedy. For instance, transformations may fail to yield good results when the data distribution is not unimodal. Multimodal distributions are not unlikely in the context of optimization. Moreover, effects from a model with transformed observations may not be that easy to interpret. Weighted least squares may be impeded by inaccurate variance estimates, or other properties of the data, such as outliers, may affect the analysis negatively.

Appendix B

Employed Kernels and Distances

This appendix collects various kernels and distance measures that have been employed in this thesis. We provide the following explanations to make sure that all employed kernels can be clearly identified by the reader, and to avoid confusion caused by conflicting naming conventions in the literature. Where available, we also attempt to provide further information on aspects like definiteness. Not all measures are described here in detail. Where they are central to the corresponding discussions, the kernels have been described in the main body of text.
Many of the listed distance measures are available in the R-package `CEGO` [263].

B.1 Real Vectors

While this thesis mostly deals with discrete problems, some parts also involve real-valued distances, e.g., Section 8.4. The following distances are used.

- The Euclidean distance $\mathrm{d}_{\mathrm{Euc}}(x, x') = \sqrt{\sum_{i=1}^{m}(x_i - x'_i)^2}$
- The Chebyshev distance $\mathrm{d}_{\mathrm{Che}}(x, x') = \max\limits_{1 \leq i \leq m} (|x_i - x'_i|)$. The Chebyshev distance is non-CNSD (see Section 7.4.4).
- A distance based on a p-norm with a degenerate value of $p = 0.5$. We denote this measure as the L0.5 distance $\mathrm{d}_{\mathrm{L0.5}}(x, x') = (\sum_{i=1}^{m} |x_i - x'_i|^p)^{1/p}$.

  Proposition 6. *The distance measure $d_{L0.5}(x, x')$ is non-CNSD.*

  Proof. With $x \in \mathbb{R}^2$ and $X = \{[1\ \ 1], [0\ \ 1], [1\ \ 0], [0\ \ 0]\}$, the corresponding distances are $\mathrm{d}_{\mathrm{L0.5}}([1\ \ 1], [0\ \ 1]) = 1$, $\mathrm{d}_{\mathrm{L0.5}}([1\ \ 1], [1\ \ 0]) = 1$, $\mathrm{d}_{\mathrm{L0.5}}([1\ \ 1], [0\ \ 0]) =$

A subset of the distance measures listed in this appendix has already been described in previous publications [273, 271, 264, 266]. Some of these descriptions are taken verbatim. Several descriptions have been revised to integrate them into the structure and notation of this thesis.

4, $\mathrm{d}_{\mathrm{L0.5}}([0\ \ 1],[1\ \ 0]) = 4$, $\mathrm{d}_{\mathrm{L0.5}}([0\ \ 1],[0\ \ 0]) = 1$, and $\mathrm{d}_{\mathrm{L0.5}}([1\ \ 0],[0\ \ 0]) = 1$. With the resulting 4×4 distance matrix $\mathbf{D}$ and $\mathbf{c} = [\text{-}1\ \ 1\ \ 1\ \text{-}1]^{\mathrm{T}}$, we receive $\mathbf{c}^{\mathrm{T}}\mathbf{D}\mathbf{c} = 8 > 0$. This proves that the distance is not CNSD. □

- A distance based on the test statistic of the two-sample independent (unpaired) t-test [203] $\mathrm{d}_{\text{t-test}}(x,x') = (\bar{x} - \bar{x}')/\sqrt{(\mathrm{var}(x) + \mathrm{var}(x'))/m}$.

  Proposition 7. *The distance measure $d_{\text{t-test}}(x,x')$ is non-CNSD.*

  Proof. With $x \in \mathbb{R}^2$ and $X = \{[0.1\ \ 0.1],[0.4\ \ 0.6],[0.9\ \ 0.1]\}$, the corresponding distances are $\mathrm{d}_{\text{t-test}}([0.1\ \ 0.1],[0.4\ \ 0.6]) = 4$,
  $\mathrm{d}_{\text{t-test}}([0.1\ \ 0.1],[0.9\ \ 0.1]) = 1$, and $\mathrm{d}_{\text{t-test}}([0.4\ \ 0.6],[0.9\ \ 0.1]) = 0$. With the resulting 3×3 distance matrix $\mathbf{D}$ and $\mathbf{c} = [1\ \ 1\ \text{-}2]^{\mathrm{T}}$, we receive $\mathbf{c}^{\mathrm{T}}\mathbf{D}\mathbf{c} = 4 > 0$. This proves that the distance is not CNSD. □

- A non-stationary distance $\mathrm{d}_{\text{Non-stat}}(x,x') = \sum_{i=1}^{m} |x_i - x_i'|(|x_i|^3 + |x_i'|^3)/2$.

  Proposition 8. *The distance measure $d_{\text{Non-stat}}(x,x')$ is non-CNSD.*

  Proof. With $x \in \mathbb{R}$ and $X = \{-1, 0, 2\}$, the corresponding distances are $\mathrm{d}_{\text{Non-stat}}(-1,0) = 0.5$, $\mathrm{d}_{\text{Non-stat}}(-1,2) = 13.5$, and $\mathrm{d}_{\text{Non-stat}}(0,2) = 8$. With the resulting 3×3 distance matrix $\mathbf{D}$ and $\mathbf{c} = [1\ \text{-}1.25\ \ 0.25]^{\mathrm{T}}$, we receive $\mathbf{c}^{\mathrm{T}}\mathbf{D}\mathbf{c} = 0.5 > 0$. This proves that the distance is not CNSD. □

B.2 Discrete Vectors and Sequences

We used distances that are applicable to discrete vectors, that is, vectors with elements that are binaries or categorical integers. These distances also apply to strings or sequences. Many distance measures are described in the literature [78]. We used:

- The Hamming distance metric

$$\mathrm{d}_{\mathrm{Ham}}(x,x') = \frac{1}{m}\sum_{i=1}^{m} a_i \ \text{ where } \ a_i = \begin{cases} 0 & \text{if } x_i = x_i', \\ 1 & \text{otherwise.} \end{cases}$$

  Here, m denotes the number of vector elements, i.e., the number of bits. An identical definition can be used, e.g., for categorical integers, sequences, permutations or strings with equal number of elements.

- The Levenshtein distance is usually used for strings. It counts the minimal number of deletions, substitutions, and insertions required to transform one string into another [234]. The Levenshtein distance is non-CNSD, see Section 7.4.4.

- The Longest Common Substring (LCStr) distance is $m - \mathrm{LCS}(x,x')$. Here $\mathrm{LCS}(x,x')$ is the length of the longest common substring [78]. This distance is non-CNSD (see Section 7.4.4).

- The Jaro-Winkler distance is based on the number of matching characters in two strings as well as the number of transpositions required to bring all matches in the same order. We use the implementation in the `stringdist` R package [249]. The Jaro-Winkler distance is non-CNSD (see Section 7.4.4).

- The Optimal String Alignment distance (OSA) is a non-metric edit distance that counts insertions, deletions, substitutions, and transpositions of characters. Each substring can be edited no more than once. It is also called the restricted Damerau-Levenshtein distance [44]. We use an implementation from the R package `stringdist` [249]. The OSA distance is not CNSD (cf. Section 7.4.4).

B.3 Permutations

For permutations, a large variety of distance measures is available [78]. In the following, the distance measures that were used in this thesis are briefly explained. Not all of them are expected to be good measures of distance between permutations. For instance, some may not be invariant to relabeling.
Here, Π^m is the set of all permutations of the numbers $\{1, 2, \dots, m\}$. A permutation has exactly m elements. We denote a single permutation with $x \in \Pi^m$ and $x = \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix}$, where x_i is the i-th element of the permutation. For example, a permutation in this notation is $x = \begin{bmatrix} 3 & 5 & 1 & 4 & 2 \end{bmatrix} \in \Pi^5$.

- The Levenshtein distance is an edit distance that counts the minimum number of deletions, insertions, and substitutions required to transform one string into another. The terms Levenshtein distance and edit distance are sometimes used as synonyms, but the Levenshtein distance is only one example of an edit distance. An implementation is described by Wagner and Fischer [255]. Since the Levenshtein distance is usually applied to strings rather than permutations, it has to be noted that a single edit operation (delete, insert, and substitute) does not produce a valid permutation of the same length. To yield valid permutations, several operations have to be performed in sequence (e.g., delete + insert, or substitution + substitution). The Levenshtein distance is non-CNSD (see Section 7.4.4).

- A swap operation transposes two adjacent elements of a permutation. Hence, the Swap distance can be defined as the minimum number of swaps required to transform one permutation into another. Other names are Precedence distance [221] and Kendall's Tau [147, 229]. The Swap distance is [229]

$$\mathrm{d}_{\text{Swa}}(x, x') = \frac{1}{s} \sum_{i=1}^{m} \sum_{j=1}^{m} z_{ij} \quad \text{where} \quad z_{ij} = \begin{cases} 1 & \text{if } x_i < x_j \text{ and } x'_i > x'_j, \\ 0 & \text{otherwise.} \end{cases}$$

  with scaling factor

$$s = (m^2 - m)/2.$$

Satisfaction of metric conditions can be shown either via the relative order matrix (cf. [178]) or simply by the fact that it is an edit distance with reversible edit operations (see Section 4.3).

- An interchange operation transposes two arbitrary elements of a permutation. Correspondingly, the Interchange distance (or Cayley distance) is the minimum number of interchanges required to transform one permutation into another [221]. An algorithm is given by Schiavinotto and Stützle [221]. The Interchange distance is non-CNSD (see Section 7.4.4).

- The Insert distance is an edit distance. The corresponding edit operation is a combination of insertion and deletion. A single element is moved from one position (delete) to a new position (insert). This distance is related to the longest common subsequence LCSeq(x, x'). The longest common subsequence is the largest number of elements that follow each other in both permutations, with interruptions. The corresponding distance is

$$\mathrm{d}_{\mathrm{Ins}}(x, x') = m - \mathrm{LCSeq}(x, x').$$

An algorithm for this computation is described by Hirschberg [122]. The distance is also called Ulam's metric [221]. It is non-CNSD (see Section 7.4.4).

- If $L_{\mathrm{str}}(x, x')$ is the respective length of the longest substring that two strings have in common, then the Longest Common Substring distance is

$$\mathrm{d}_{\mathrm{LCStr}}(x, x') = \frac{m - L_{\mathrm{str}}(x, x')}{m - 1}.$$

Here, the permutations are interpreted as strings. This distance is non-CNSD (see Section 7.4.4). It has also been called the factor distance [78].

- The R-distance [54, 229] counts how often two elements are neighbors and occur in the same order in one permutation, but not in another permutation. It is identical to the uni-directional adjacency distance [210]. It is computed by

$$\mathrm{d}_{\mathrm{R}}(x, x') = \sum_{i=1}^{m-1} y_i \quad \text{where} \quad y_i = \begin{cases} 0 & \text{if } \exists j : x_i = x'_j \text{ and } x_{i+1} = x'_{j+1}, \\ 1 & \text{otherwise.} \end{cases}$$

Following the reasoning of Moraglio [178], we can easily show that the R-distance is a metric.

Proposition 9. *The R-distance is a metric for permutations.*

Proof. We can represent each permutation with its R-matrix. In this matrix, each row indicates the first and each column the second element in a sequence of two elements. E.g., the permutation $\begin{bmatrix}1 & 2 & 3\end{bmatrix}$ has the R-matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

since 2 follows 1 (second column, first row is set to 1) and 3 follows 2 (third column, second row is set to 1). Correspondingly, the permutation $\begin{bmatrix}1 & 3 & 2\end{bmatrix}$ has the R-matrix

$$\begin{bmatrix}0 & 0 & 1\\ 0 & 0 & 0\\ 0 & 1 & 0.\end{bmatrix}$$

Clearly, the Hamming distance between two R-matrices is the same as the R-distance of the corresponding permutations. Each unique permutation has a unique R-matrix. From each unique R-matrix, a unique permutation can be identified. Hence, the R-matrix is a bijection. R-matrices are a subset of binary matrices. The Hamming distance is metric on binary matrices, and therefore also a metric on the subset. Since the R-matrix is a bijection, it follows that the R-distance has to be a metric. □

- The (bi-directional) adjacency distance [210, 221] counts how often two elements are neighbors in one permutation, but not in the other permutation. The order of the two elements is irrelevant. This distance is similar to the uni-directional adjacency distance. The adjacency distance is a pseudo-metric [178]. Clearly, the adjacency distance of a permutation to the reverse of itself is zero. Hence, the distance violates the condition for identity preservation (see Section 4.3).

- The Position distance [221] is identical to the Deviation distance or Spearman's footrule [229],

$$\mathrm{d}_{\mathrm{Pos}}(x, x') = \frac{1}{s}\sum_{l=1}^{m} |i - j| \text{ where } x_i = x'_j = l$$

with scaling factor

$$s = \begin{cases} (m^2 - 1)/2 & \text{if } m \text{ is odd} \\ m^2/2 & \text{if } m \text{ is even} \end{cases}.$$

Metric conditions can be proved in a similar way as for R-distance.

Proposition 10. *The Position distance is a metric for permutations.*

Proof. To calculate the position distance of permutations x and x', the first step is to calculate the inverse of both permutations $\bar{x}$, $\bar{x}'$ (i.e., determine the ordering). Then, the Position distance is simply the Manhattan distance of the inverse permutations.

$$\mathrm{d}_{\mathrm{Pos}}(x, x') = \mathrm{d}_{\mathrm{Man}}(\bar{x}, \bar{x}')$$

Clearly, the Manhattan distance is a metric for permutations (due to permutations being a subset of all real vectors). Since permutation inversion is a bijection, it follows that the Position distance has to be a metric. □

- The non-metric Squared Position distance corresponds to Spearman's rank correlation coefficient [229]. The square root of this distance is a metric because it is essentially the Euclidean distance on orderings.

$$\mathrm{d}_{\text{Pos2}}(x, x') = \frac{1}{s} \sum_{l=1}^{m} (i - j)^2 \text{ where } x_i = x'_j = l$$

  with scaling factor

$$s = (m^3 - m)/3$$

- The Hamming distance for permutations can be based on the same definition that is used for binary vectors. A scaling factor of $s = m$ scales the distance to the interval $[0; 1]$.

- The Euclidean distance for permutations,

$$\mathrm{d}_{\text{Euc}}(x, x') = \frac{1}{s} \sqrt{\sum_{i=1}^{m} (x_i - x'_i)^2},$$

  is not an intuitive choice. It is more commonly used for continuous parameter spaces. Since permutations can be seen as a subset of all real vectors, the Euclidean distance is also a distance metric for permutations. The only adaption to permutation space is the scaling factor s, to guarantee that the distance yields values between zero and one,

$$s = \begin{cases} \sqrt{4c(c+1)(2c+1)/3} & \text{if } m \text{ is odd, with } c = (m-1)/2 \\ \sqrt{2c(4c^2 - 1)/3} & \text{if } m \text{ is even, with } c = m/2 \end{cases}.$$

- The Manhattan distance for permutations (also called A-Distance [229, 54]) is

$$\mathrm{d}_{\text{Man}}(x, x') = \frac{1}{s} \sum_{i=1}^{m} |x_i - x'_i|$$

  It is usually used for integer spaces, and it is not invariant to relabeling. The only adaption to permutations is the scaling factor s, to guarantee values between zero and one.

$$s = \begin{cases} (m^2 - 1)/2 & \text{if } m \text{ is odd} \\ m^2/2 & \text{if } m \text{ is even} \end{cases}$$

- The Chebyshev distance (also denoted as L_∞ metric) is also not intuitively used for permutations. It is defined as

$$\mathrm{d}_{\text{Che}}(x, x') = \frac{1}{s} \max_{1 \leq i \leq m} (|x_i - x'_i|)$$

  with the trivial scaling factor $s = m - 1$. The Chebyshev distance is non-CNSD (see Section 7.4.4).

- The Lee distance [160] has been developed as a string distance but can be applied to permutations [77].

$$\mathrm{d}_{\text{Lee}}(x,x') = \sum_{i=1}^{m} \min(|x_i - x'_i|, m - |x_i - x'_i|).$$

  It is a metric [213].

- The non-metric Cosine distance is based on the scalar product of two permutations. It is derived from the cosine similarity [233] of two vectors:

$$\mathrm{d}_{\text{Cos}}(x,x') = 1 - \frac{x \cdot x'}{||x||\,||x'||}$$

  Here, $x \cdot x'$ is the dot product, and $||x||$ indicates the length of a vector. In both cases, the permutations are interpreted as real-valued vectors.

- If the position of a permutation x in the lexicographic ordering of all permutations of the same length is Lex(x), then the lexicographic distance metric is

$$\mathrm{d}_{\text{Lex}}(x,x') = \frac{1}{s}|\text{Lex}(x) - \text{Lex}(x')|.$$

  with $s = m! - 1$.

B.4 Signed Permutations

While there are applications where signed permutations are relevant [254], we did not consider them in much detail. We only provided an example for non-CNSDness of the reversal distance in Section 7.4.4. The reversal distance for signed permutations is an edit distance, based on the number of reversals required to transform one signed permutation into another. Unlike the unsigned case, the reversal distance for signed permutations can be computed in linear time [15, 56]. We use the non-cyclic reversal distance provided in the Genome Rearrangements Analysis under Parsimony and other Phylogenetic Algorithms (GRAPPA) library version 2.0 [14].

B.5 Trees

Distances for trees are described in more detail in Section 9.1, we only briefly describe these here.

- The Tree Edit Distance (TED) counts the number of node deletions, node insertions, and node relabelings required to transfer one tree into another [196]. The TED is non-CNSD, see Section 7.4.4.

- The Phenotypic Distance (PhD) is not necessarily a distance on trees. In the form it is used here, it is a distance on the output of symbolic expressions, which may be represented by trees. The specific variant of the PhD we used is non-CNSD (see Section 9.1.3).

- The Structural Hamming Distance (SHD) could be seen as an approximation of the TED. The SHD performs a recursive count of the number of different nodes and labels of two trees. The recursion proceeds as long as the two trees have matching structures. Two variants are discussed. The first variant compares two trees without aligning them (SHD1). The second variant aligns the trees such that the SHD between the trees is minimal (SHD2). The SHD2 variant is non-CNSD (see Section 9.1.3).

B.6 Hierarchical Search Spaces

Hierarchical search spaces contain variables that require some condition to be satisfied. If the condition is satisfied, the corresponding variable is active, or has an impact on the objective function. Such conditional relationships can be represented as graphs. The data type of the variables themselves may vary, e.g., it may be real, categorical, or binary. Because the kernels for hierarchical search spaces are the main subject of Section 9.2, we do not describe them here, but refer to that section for details.

Appendix C

Variation Operators

In the following, we list various variation operators that were used in this thesis. Brief explanations and / or references to the literature are given. Here, x or x' are parents and x^* denotes an offspring (after mutation or recombination). Other publications, e.g., the book by Eiben and Smith [82], give a more detailed overview of variation operators. However, the following remarks are necessary to clarify the terminology and usage of these operators.

C.1 Permutation Operators

We use the same notation as in Appendix B.3. In the following, $a \in \mathbb{N}$ and $b \in \mathbb{N}$.

C.1.1 Mutation

The mutation operators for permutations are defined in the following. They share one parameter, the mutation rate $r_{\text{mut}} \in \mathbb{R}^+$. The mutation rate defines the number of times that a mutation operator is applied consecutively $n_{\text{mut}} = \lceil m r_{\text{mut}} \rceil$.

- **Swap mutation** transposes two adjacent elements of a permutation.

$$x = \begin{bmatrix} x_1 & \dots & x_a & x_b & \dots & x_m \end{bmatrix}$$
$$x^* = \begin{bmatrix} x_1 & \dots & x_b & x_a & \dots & x_m \end{bmatrix},$$

  with $1 \leq a < (m-1)$ and $b = a+1$.
- **Interchange mutation** transposes two arbitrary elements of a permutation.

$$x = \begin{bmatrix} x_1 & \dots & x_{a-1} & x_a & x_{a+1} & \dots & x_{b-1} & x_b & x_{b+1} & \dots & x_m \end{bmatrix}$$
$$x^* = \begin{bmatrix} x_1 & \dots & x_{a-1} & x_b & x_{a+1} & \dots & x_{b-1} & x_a & x_{b+1} & \dots & x_m \end{bmatrix},$$

  with $1 \leq a \leq m$ and $1 \leq b \leq m$.

Please note that swap, interchange and reversal mutation operators for permutations have already been described in this form by Zaefferer et al. [266].

- **Insert mutation** removes a certain element at one position and inserts it somewhere else.

$$\begin{aligned} x &= \begin{bmatrix} x_1 & \dots & x_{a-1} & x_a & x_{a+1} & \dots & x_{b-1} & x_b & x_{b+1} & \dots & x_m \end{bmatrix} \\ x^* &= \begin{bmatrix} x_1 & \dots & x_{a-1} & x_b & x_a & x_{a+1} & \dots & x_{b-1} & x_{b+1} & \dots & x_m \end{bmatrix}, \end{aligned}$$

  with $1 \leq a \leq m$ and $1 \leq b \leq m$.
- **Reversal mutation** reverses the ordering of a sub-sequence of the permutation.

$$\begin{aligned} x &= \begin{bmatrix} x_1 & \dots & x_a & x_{a+1} & \dots & x_{b-1} & x_b & \dots & x_m \end{bmatrix} \\ x^* &= \begin{bmatrix} x_1 & \dots & x_b & x_{b-1} & \dots & x_{a+1} & x_a & \dots & x_m \end{bmatrix}, \end{aligned}$$

  with $1 \leq a < b \leq m$.

C.1.2 Recombination

For the sake of brevity, we only briefly remark on these operators. For more details, we refer to Larranaga et al. [158] and Eiben and Smith [82].

- **Cycle crossover** *"attempts to create an offspring from the parents where every position is occupied by a corresponding element from one of the parents"* [158]. It *"is concerned with preserving as much information as possible about the absolute position in which elements occur"* [82].
- **Order crossover** *"constructs an offspring by choosing a substring of one parent and preserving the relative order of the elements of the other parent."* [158] It intends *"to transmit information about relative order"* [82].
- **Position-based crossover** is a variant of order crossover [158]. It selects a random number of elements from a parent and transmits them to their original positions in the offspring. The remaining elements are inserted in their original order.
- **Alternating position crossover** *"creates an offspring by selecting alternately the next element of the first parent and the next element of the second parent, omitting the elements already present in the offspring."* [158]

C.2 Binary String Operators

This section lists operators for bit strings or binary strings $x \in \mathbb{B}$. We denote a single binary string with m bits as $x = \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix}$ where x_i is a specific bit of the string at position i. We denote $\bar{x}_i$ as the inversion or flip of a bit i, i.e., if $x_i = 0$ then $\bar{x}_i = 1$. For example, a binary string is $x = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \end{bmatrix} \in \mathbb{B}^5$. In the following, $a \in \mathbb{N}$ and $b \in \mathbb{N}$.

C.2.1 Mutation

- **Bit flip** inverses a single bit.

$$
\begin{aligned}
x &= \begin{bmatrix} x_1 & \dots & x_a & \dots & x_m \end{bmatrix} \\
x^* &= \begin{bmatrix} x_1 & \dots & \bar{x}_a & \dots & x_m \end{bmatrix},
\end{aligned}
$$

  with $1 \leq a \leq m$.
- **Block inversion** inverses (flips) all bits in a block, or substring, of a binary string.

$$
\begin{aligned}
x &= \begin{bmatrix} x_1 & \dots & x_a & x_{a+1} & \dots & x_{b-1} & x_b & \dots & x_m \end{bmatrix} \\
x^* &= \begin{bmatrix} x_1 & \dots & \bar{x}_a & \bar{x}_{a+1} & \dots & \bar{x}_{b-1} & \bar{x}_b & \dots & x_m \end{bmatrix},
\end{aligned}
$$

  with $1 \leq a < b \leq m$.
- **Cycle** moves all elements to the right by a positions, cyclically.

$$
\begin{aligned}
x &= \begin{bmatrix} x_1 & \dots & x_{a+1} & \dots & x_m \end{bmatrix} \\
x^* &= \begin{bmatrix} x_{m-(a-1)} & \dots & x_1 & \dots & x_{m-a} \end{bmatrix},
\end{aligned}
$$

  with $1 \leq a \leq (m-1)$.

C.2.2 Recombination

- **1-point crossover** selects a single cut point. The elements of the string before that point are chosen from parent x, the rest from the second parent x'.
- **2-point crossover** selects two cut points. The elements of the string between those points are chosen from parent x, the rest from the second parent x'.
- **Uniform crossover** Elements are chosen uniform randomly from both parents x, x'.
- **AND crossover** The two parents x, x' are combined by an element-wise logical AND operation, that is, $x_i^* = x_i \wedge x_i'$.

C.3 Tree Operators

For the GP application in Section 9.1, we needed variation operators for trees. The corresponding operators were taken from the `rgp` package [91].
For mutation, `mutateSubtreeFast` was used. This operator randomly replaces, deletes, or creates new subtrees. The subtrees are grown up to a maximum tree depth specified by the user. In addition, terminal nodes may be created, which can be constant coefficients or symbolic variables. Each of these optional events involves a parameter that specifies the probability of an event.
Furthermore, the EA employed `crossoverexprFast` for recombination. This recombination operator randomly selects a subtree of the first parent, and replaces it with a random subtree of the second parent.

Appendix D

Additional Figures

D.1 Box Plots: Multi-Kernel Experiments

The Figs. D.1 to D.5 show the remaining box plots of the results from Section 6.3.2, for the TSP, WTP, UNI, QAP, and FSP instances. The ATSP instances are shown in Fig. 6.2.

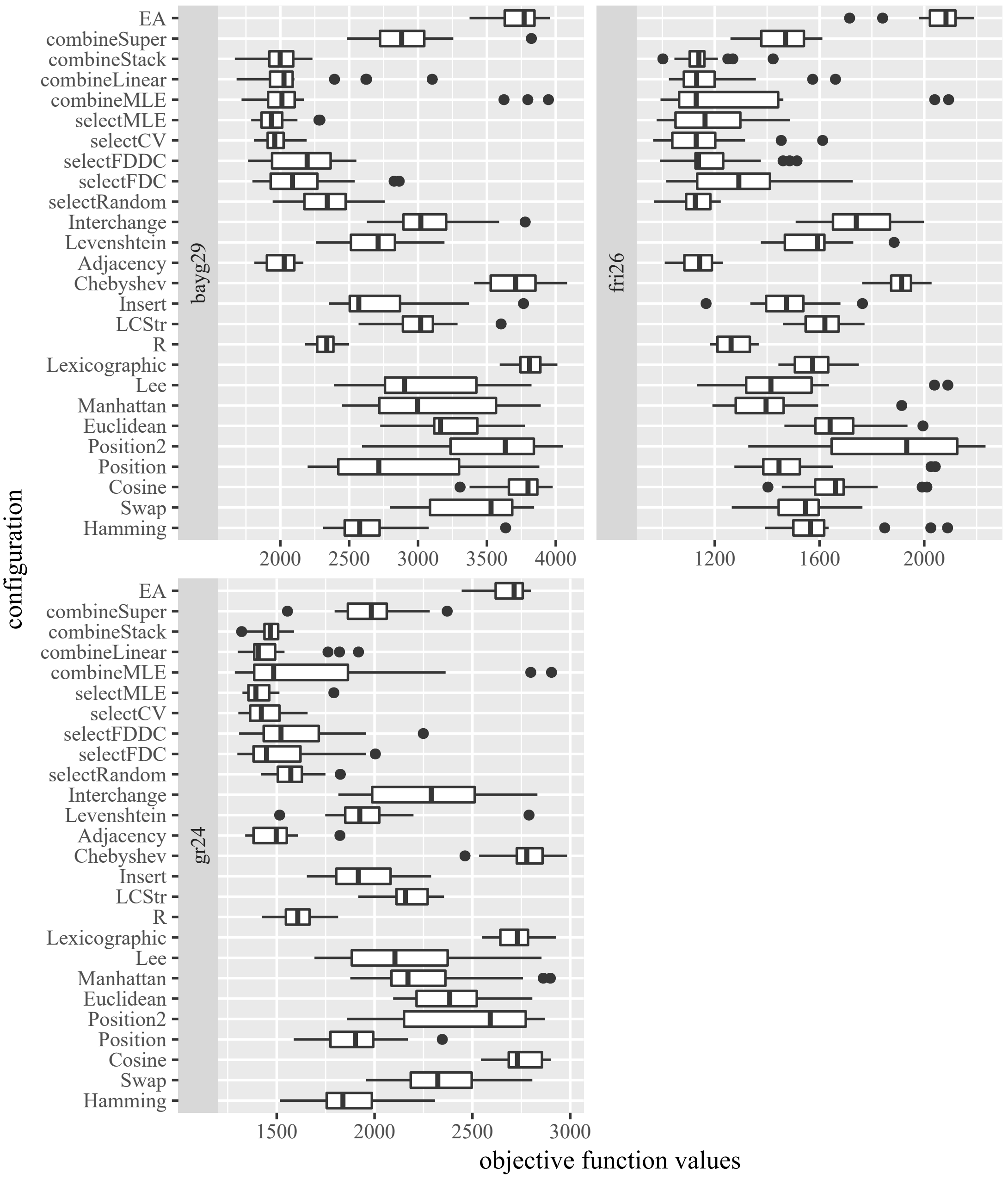


Figure D.1: *Box plots for the results of different optimization runs with different kernels for permutations (or different methods to select/combine kernels). The depicted test problems are all TSP instances. Smaller values are better.*

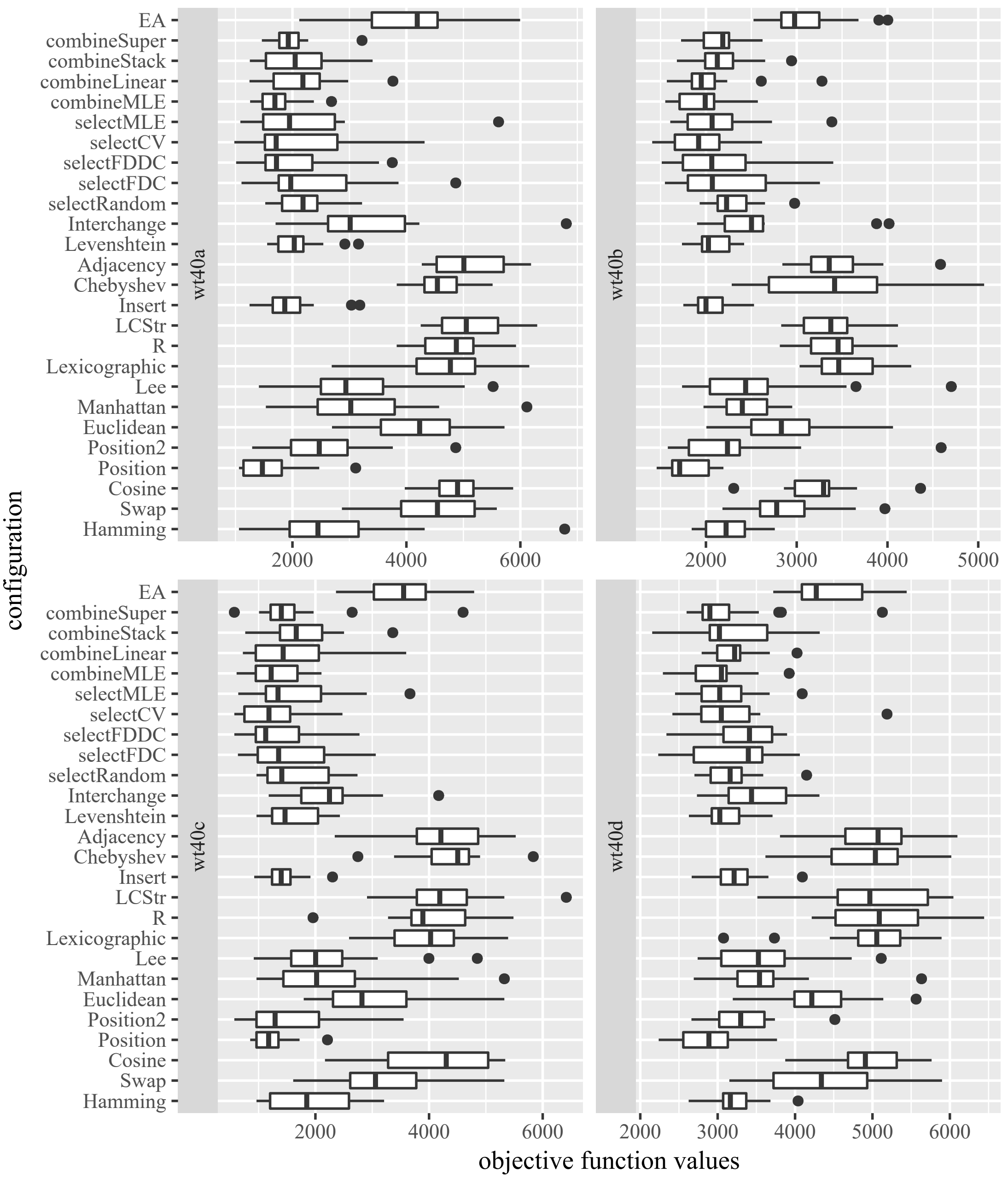


Figure D.2: *Box plots for the results of different optimization runs with different kernels for permutations (or different methods to select/combine kernels). The depicted test problems are all WTP instances. Smaller values are better.*

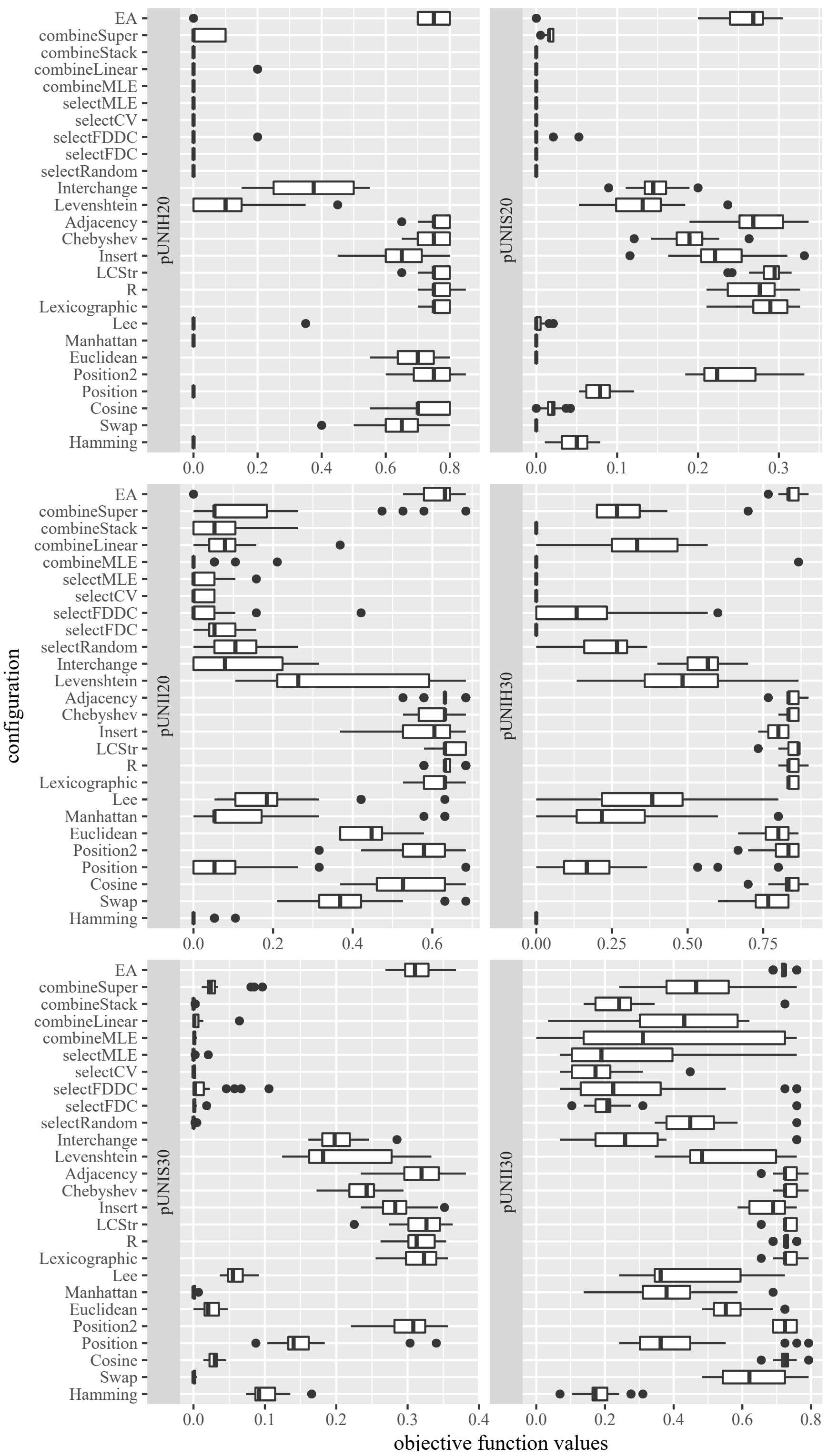


Figure D.3: *Box plots for the results of different optimization runs with different kernels for permutations (or different methods to select/combine kernels). The depicted test problems are all UNI instances. Smaller values are better.*

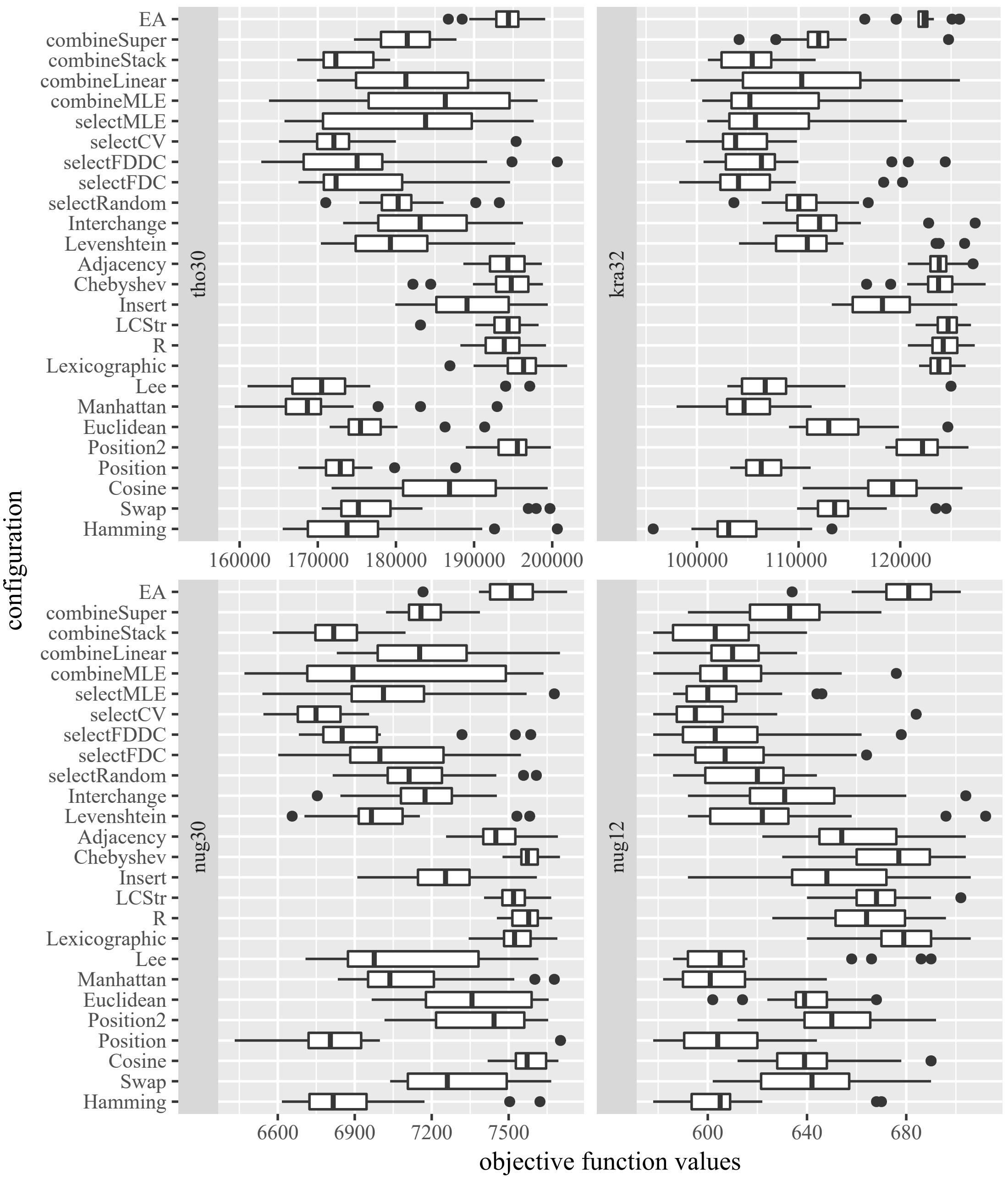


Figure D.4: *Box plots for the results of different optimization runs with different kernels for permutations (or different methods to select/combine kernels). The depicted test problems are all QAP instances. Smaller values are better.*

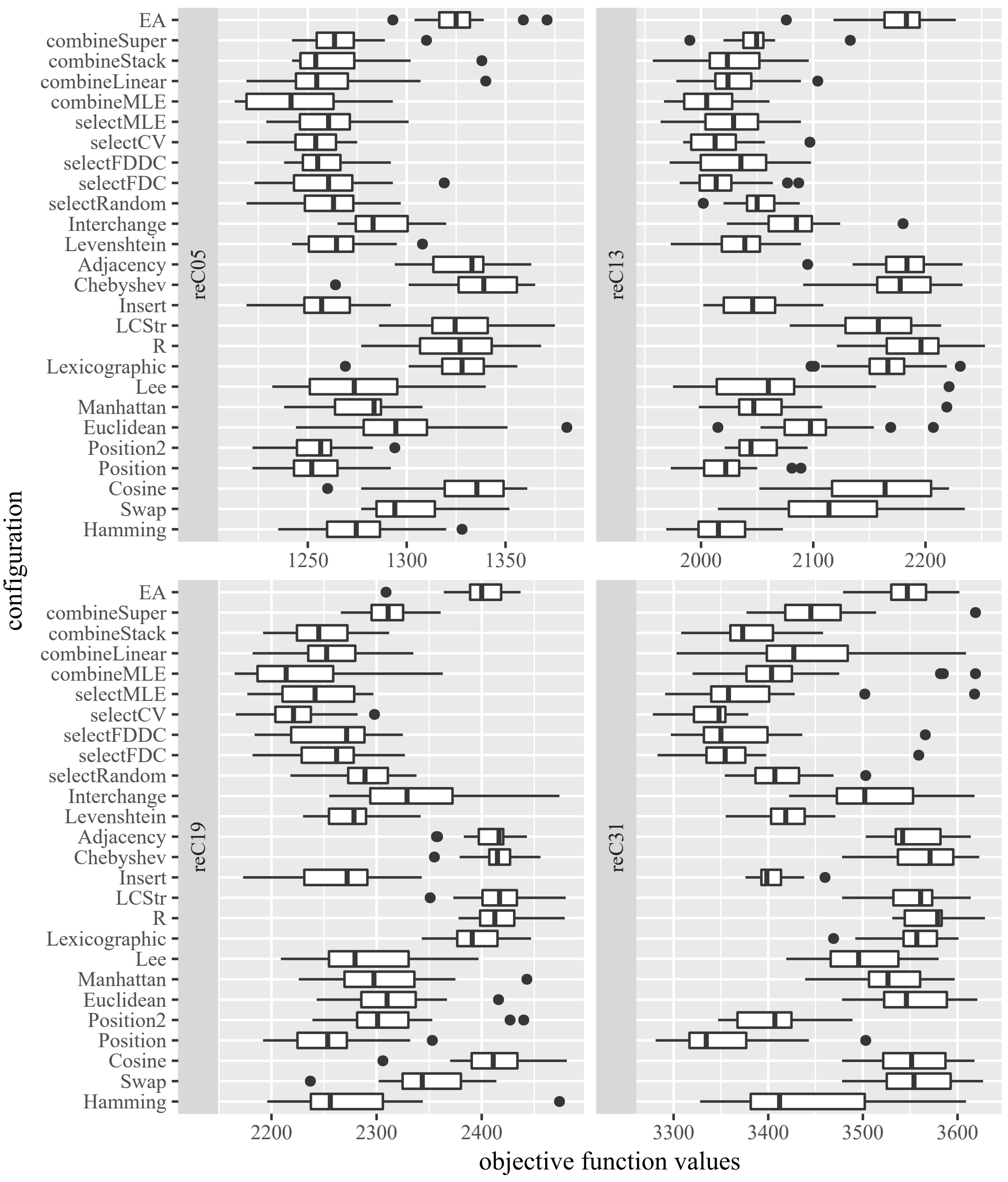


Figure D.5: *Box plots for the results of different optimization runs with different kernels for permutations (or different methods to select/combine kernels). The depicted test problems are all FSP instances. Smaller values are better.*

D.2 Kriging with Indefinite Kernels: Examples

The Figs. D.6 to D.9 show visualizations of models based on different configurations for the correction of indefinite kernels. The depicted models are based on the example presented in Section 8.3.

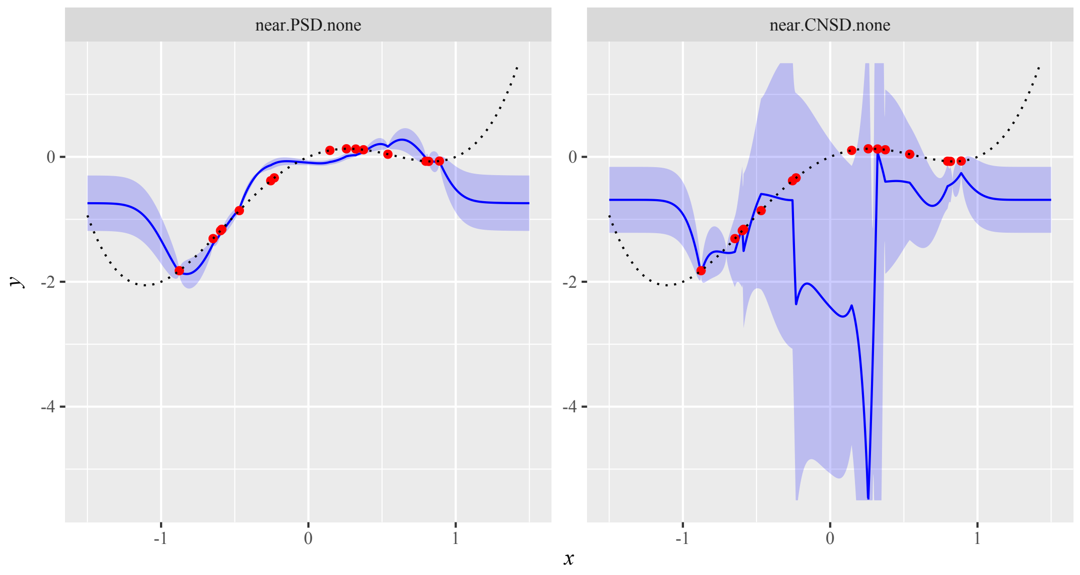


Figure D.6: *Example: Kriging models based on the nearest correlation matrix (PSD) or the nearest Euclidean matrix (CNSD).*

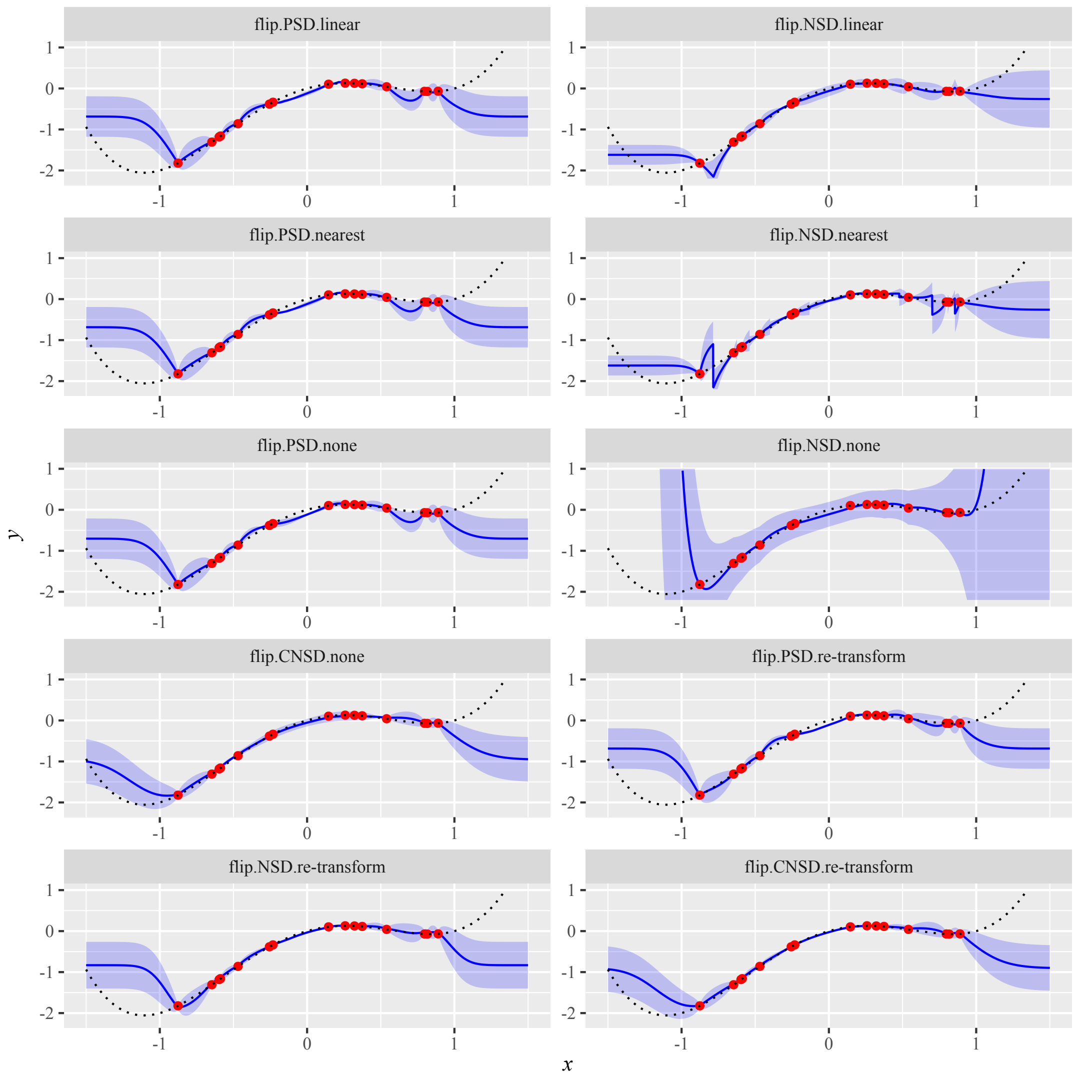


Figure D.7: *Example: Kriging models with spectrum flip, based on different correction types (NSD, CNSD, PSD) and repair methods (none, re-transform, linear, nearest-neighbor).*

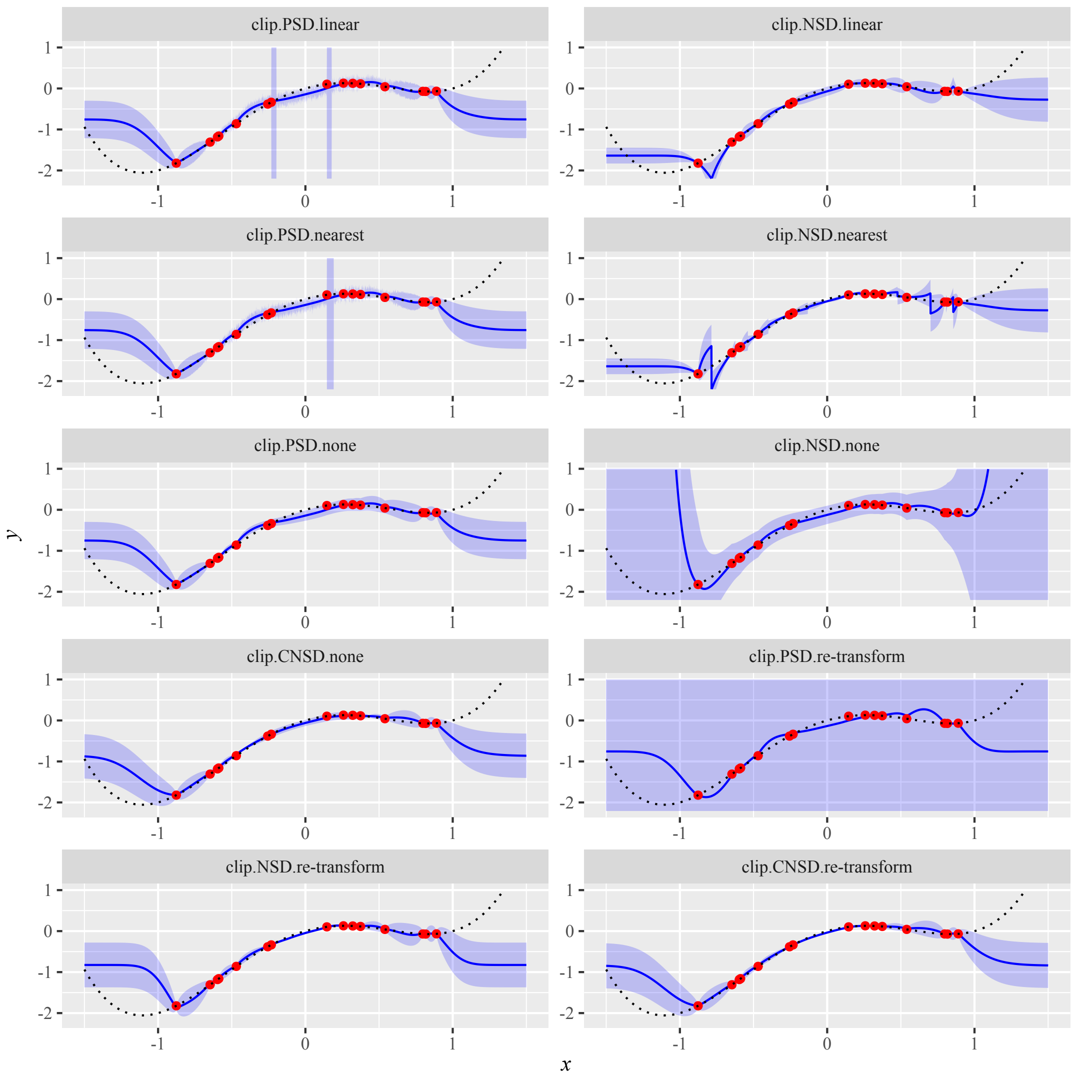


Figure D.8: *Example: Kriging models with spectrum clip, based on different correction types (NSD, CNSD, PSD) and repair methods (none, re-transform, linear, nearest-neighbor).*

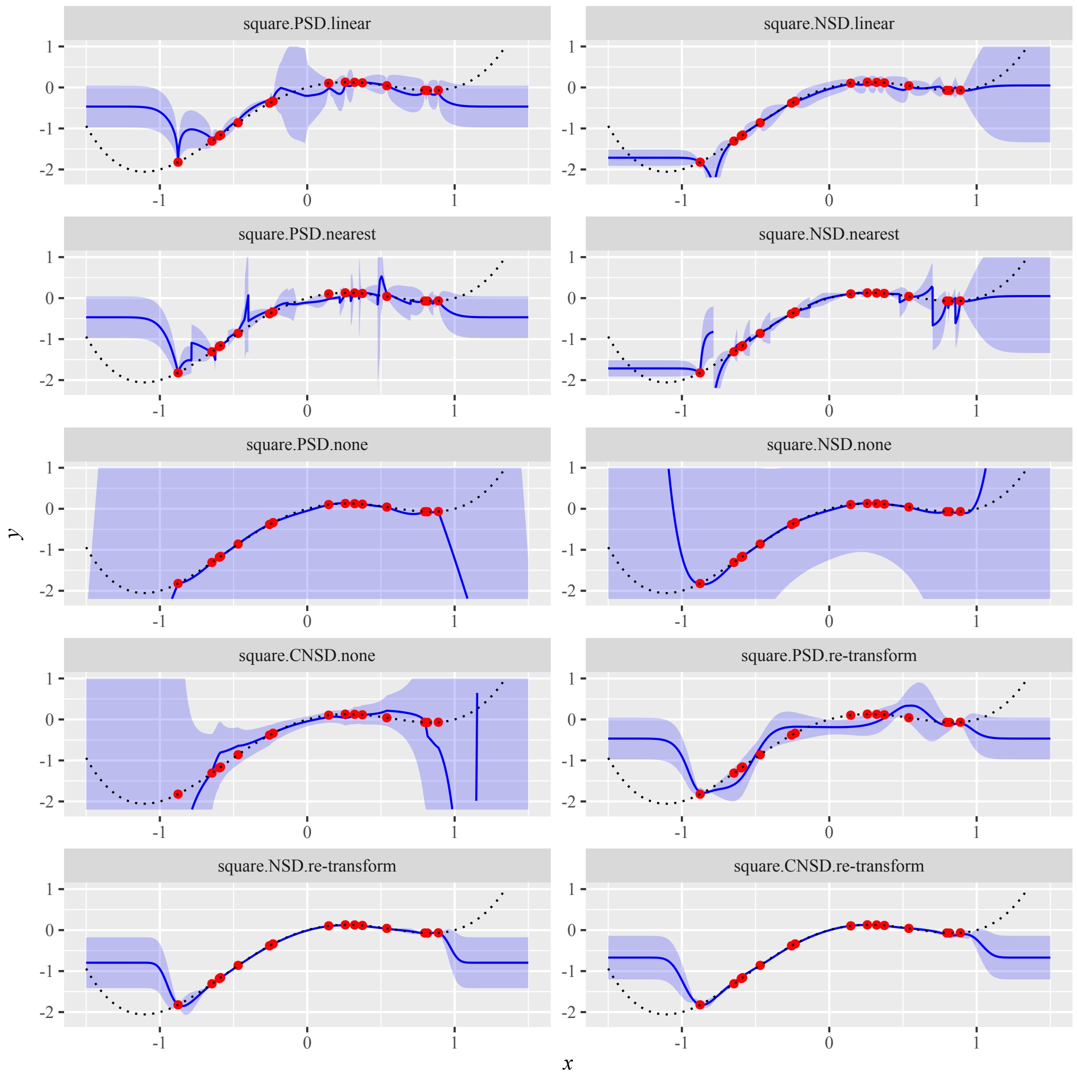


Figure D.9: *Example: Kriging models with spectrum square, based on different correction types (NSD, CNSD, PSD) and repair methods (none, re-transform, linear, nearest-neighbor).*

Bibliography

[1] T. Abdul-Razaq, C. Potts, and L. V. Wassenhove. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*, 26(2–3):235–253, Mar. 1990.

[2] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Database Theory — ICDT 2001: 8th International Conference*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434, London, UK, 2001. Springer.

[3] I. Alabdulmohsin, M. Cisse, X. Gao, and X. Zhang. Large margin classification with indefinite similarities. *Machine Learning*, 103(2):215–237, May 2016.

[4] I. Alabdulmohsin, X. Gao, and X. Zhang. Support vector machines with indefinite kernels. In D. Phung and H. Li, editors, *Proceedings of the Sixth Asian Conference on Machine Learning (ACML 2014)*, volume 39 of *Proceedings of Machine Learning Research*, pages 32–47, Nha Trang City, Vietnam, Nov. 2015. PMLR.

[5] A. Allahverdi, C. Ng, T. E. Cheng, and M. Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, June 2008.

[6] R. Allmendinger, C. A. Coello Coello, M. T. M. Emmerich, J. Hakanen, Y. Jin, and E. Rigoni. Surrogate-assisted multicriteria optimization (WG6). In S. Greco, K. Klamroth, J. D. Knowles, and G. Rudolph, editors, *Understanding Complexity in Multiobjective Optimization (Dagstuhl Seminar 15031) - Dagstuhl Reports*, volume 5(1), pages 151–156, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[7] L. Altenberg. Fitness distance correlation analysis: An instructive counterexample. In T. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 57–64, East Lansing, MI, USA, July 1997. Morgan Kaufmann.

[8] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, Aug. 1992.

[9] S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury. Simulation optimization: a review of algorithms and applications. *Annals of Operations Research*, 240(1):351–380, Sept. 2015.

[10] T. W. Anderson. On the distribution of the two-sample Cramer-von Mises criterion. *Annals of Mathematical Statistics*, 33(3):1148–1159, Sept. 1962.

[11] M. Asadi and J. Goldak. Combinatorial optimization of weld sequence by using a surrogate model to mitigate a weld distortion. *International Journal of Mechanics and Materials in Design*, 7(2):123–139, June 2011.

[12] M. S. Ayhan and C.-H. H. Chu. Towards indefinite Gaussian processes. Technical report, University of Louisiana at Lafayette, 2012. Available: `http://www.cs.cmu.edu/~sbalakri/nonpar_final/IndefGP_NIPS2012_ws_FINAL.pdf`, accessed: 2018-07-11.

[13] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, Mar. 1993.

[14] D. A. Bader, B. M. Moret, T. Warnow, S. K. Wyman, M. Yan, J. Tang, A. C. Siepel, and A. Caprara. Genome rearrangements analysis under parsimony and other phylogenetic algorithms (GRAPPA) 2.0. Online, Oct. 2004. Available: `https://www.cs.unm.edu/~moret/GRAPPA/`, accessed: 2018-07-11.

[15] D. A. Bader, B. M. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8(5):483–491, July 2004.

[16] L. Bajer and M. Holeňa. Surrogate model for continuous and discrete genetic optimization based on RBF networks. In *Intelligent Data Engineering and Automated Learning – IDEAL 2010: 11th International Conference*, volume 6283 of *Lecture Notes in Computer Science*, pages 251–258, Paisley, UK, Sept. 2010. Springer.

[17] P. V. Balachandran, B. Kowalski, A. Sehirlioglu, and T. Lookman. Experimental search for high-temperature ferroelectric perovskites guided by two-step machine learning. *Nature Communications*, 9(1):1–9, Apr. 2018.

[18] R. B. Bapat and T. E. S. Raghavan. *Nonnegative Matrices and Applications*. Cambridge University Press, Mar. 1997.

[19] J. Barrera and C. A. Coello Coello. Test function generators for assessing the performance of PSO algorithms in multimodal optimization. In B. K. Panigrahi, Y. Shi, and M.-H. Lim, editors, *Handbook of Swarm Intelligence: Concepts, Principles and Applications*, volume 8 of *Adaptation, Learning, and Optimization*, pages 89–117. Springer, Berlin, Heidelberg, 2011.

[20] T. Bartz-Beielstein. *New Experimentalism Applied to Evolutionary Computation*. PhD thesis, Universität Dortmund, Dortmund, Germany, Apr. 2005.

[21] T. Bartz-Beielstein. How to create generalizable results. In J. Kacprzyk and W. Pedrycz, editors, *Springer Handbook of Computational Intelligence*, pages 1127–1142. Springer, Berlin, 2015.

[22] T. Bartz-Beielstein. Stacked generalization of surrogate models – a practical approach. Technical Report CIplus, 5/2016, Fakultät für Informatik und Ingenieurwissenschaften (F10), 2016. Available: `https://nbn-resolving.org/urn:nbn:de:hbz:832-cos4-3759`, accessed: 2018-07-11.

[23] T. Bartz-Beielstein, J. Branke, J. Mehnen, and O. Mersmann. Evolutionary algorithms. *WIREs Data Mining and Knowledge Discovery*, 4(3):178–195, May 2014.

[24] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, volume 1, pages 773–780, Edinburgh, Scotland, UK, Sept. 2005. IEEE.

[25] T. Bartz-Beielstein and M. Preuss. Automatic and interactive tuning of algorithms. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO'11)*, pages 1361–1380, Dublin, Ireland, July 2011. ACM.

[26] T. Bartz-Beielstein and M. Zaefferer. Model-based methods for continuous and discrete global optimization. *Applied Soft Computing*, 55:154–167, June 2017.

[27] T. Bartz-Beielstein, M. Zaefferer, and Q. C. Pham. Optimization via multimodel simulation. *Structural and Multidisciplinary Optimization*, pages 1–15, Feb. 2018.

[28] O. Bastert, D. Rockmore, P. F. Stadler, and G. Tinhofer. Landscapes on spaces of trees. *Applied Mathematics and Computation*, 131(2-3):439–459, Sept. 2002.

[29] J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, May 1994.

[30] J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, Nov. 1990.

[31] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *ArXiv e-prints*, Feb. 2014. ArXiv ID: arXiv:1306.6709v4.

[32] C. Berg, J. P. R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups*, volume 100 of *Graduate Texts in Mathematics*. Springer, New York, 1984.

[33] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, volume 28(1) of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, GA, USA, June 2013. PMLR.

[34] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyperparameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, pages 2546–2554, Granada, Spain, 2011. Curran Associates, Inc.

[35] H. S. Bernardino, L. G. Fonseca, and H. J. C. Barbosa. Surrogate-assisted artificial immune systems for expensive optimization problems. In W. P. dos Santos, editor, *Evolutionary Computation*, chapter 10, pages 179–198. IntechOpen, Vienna, Austria, Oct. 2009.

[36] N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, Sept. 2007.

[37] Bioanalytical Sciences Group - Manchester University. Microarray data: Analysis of the complete sequence-fitness landscape of a DNA aptamer. Online, 2011. Available: `http://dbkgroup.org/directed-evolution/`, accessed: 2018-07-11.

[38] B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang. mlrMBO: A modular framework for model-based optimization of expensive black-box functions. *ArXiv e-prints*, Mar. 2017. ArXiv ID: 1703.03373v2.

[39] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[40] T. Blaschke, M. Olivecrona, O. Engkvist, J. Bajorath, and H. Chen. Application of generative autoencoder in de novo molecular design. *Molecular Informatics*, 37(1–2):45–55, Dec. 2017.

[41] J. B. Boisvert and C. V. Deutsch. Programs for Kriging and sequential Gaussian simulation with locally varying anisotropy using non-Euclidean distances. *Computers & Geosciences*, 37(4):495–510, Apr. 2011.

[42] E. Bonnaud. Mitigation of weld residual deformations by weld sequence optimization: limitations and enhancements of surrogate models. In F. Iacoviello, P. M. Moreira, and P. J. Tavares, editors, *2nd International Conference on Structural Integrity, ICSI 2017*, volume 5 of *Procedia Structural Integrity*, pages 310–317, Funchal, Madeira, Portugal, Sept. 2017. Elsevier.

[43] I. Boussaid, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, July 2013.

[44] L. Boytsov. Indexing methods for approximate dictionary searching: Comparative analysis. *Journal of Experimental Algorithmics*, 16:1–91, May 2011.

[45] L. Breiman. Stacked regressions. *Machine Learning*, 24(1):49–64, July 1996.

[46] A. Brownlee, J. McCall, and Q. Zhang. Fitness modeling with Markov networks. *IEEE Transactions on Evolutionary Computation*, 17(6):862–879, Sept. 2013.

[47] A. E. Brownlee. Mining Markov network surrogates for value-added optimisation. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion (GECCO'16)*, pages 1267–1274, Denver, CO, USA, July 2016. ACM.

[48] A. E. Brownlee and J. A. Wright. Constrained, mixed-integer and multi-objective optimisation of building designs by NSGA-II with fitness approximation. *Applied Soft Computing*, 33:114–126, Aug. 2015.

[49] H. Büning and G. Trenkler. *Nichtparametrische Statistische Methoden*. de Gruyter, 2nd edition, Feb. 2013.

[50] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, June 1998.

[51] R. E. Burkard. Quadratic assignment problems. *European Journal of Operational Research*, 15(3):283–289, 1984.

[52] R. E. Burkard, S. E. Karisch, and F. Rendl. QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391–403, June 1997.

[53] F. Camastra and A. Vinciarelli. *Machine Learning for Audio, Image and Video Analysis: Theory and Applications*. Advanced information and knowledge processing. Springer, London, 2nd edition, 2015.

[54] V. Campos, M. Laguna, and R. Martí. Context-independent scatter and tabu search for permutation problems. *INFORMS Journal on Computing*, 17(1):111–122, Feb. 2005.

[55] G. Camps-Valls, J. D. Martín-Guerrero, J. L. Rojo-Álvarez, and E. Soria-Olivas. Fuzzy sigmoid kernel for support vector classifiers. *Neurocomputing*, 62:501–506, Dec. 2004.

[56] A. Caprara. Sorting by reversals is difficult. In *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB'97)*, pages 75–83, Santa Fe, NM, USA, 1997. ACM.

[57] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano. A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Progress in Artificial Intelligence*, 1(1):103–117, Apr. 2012.

[58] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano. A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 18(2):286–300, Apr. 2014.

[59] J. Ceberio, A. Mendiburu, and J. A. Lozano. Introducing the Mallows model on estimation of distribution algorithms. In B.-L. Lu, L. Zhang, and J. Kwok, editors, *Neural Information Processing: 18th International Conference, ICONIP 2011*, volume 7063 of *Lecture Notes in Computer Science*, pages 461–470, Shanghai, China, 2011. Springer.

[60] J. Chen and J. Ye. Training SVM with indefinite kernels. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pages 136–143, Helsinki, Finland, July 2008. ACM.

[61] Y. Chen, M. R. Gupta, and B. Recht. Learning kernels from indefinite similarities. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, pages 145–152, Montreal, Quebec, Canada, June 2009. ACM.

[62] M. Chiarandini and Y. Goegebeur. Mixed models for the analysis of optimization algorithms. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 225–264. Springer, Berlin, 2010.

[63] T. Chugh, K. Sindhya, J. Hakanen, and K. Miettinen. A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. *Soft Computing*, pages 1–30, Dec. 2017.

[64] W. S. Cleveland, S. J. Devlin, and E. Grosse. Regression by local fitting. *Journal of Econometrics*, 37(1):87–114, Jan. 1988.

[65] W. J. Conover. *Practical Nonparametric Statistics*. Wiley, New York, NY, 3rd edition, 1999.

[66] W. J. Conover and R. L. Iman. On multiple-comparisons procedures. Technical Report LA-7677-MS, Los Alamos Sci. Lab., 1979. Available: `http://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-07677-MS`, accessed: 2018-07-11.

[67] G. Constantine. Lower bounds on the spectra of symmetric matrices with nonnegative entries. *Linear Algebra and its Applications*, 65:171–178, Feb. 1985.

[68] C. Cortes, P. Haffner, and M. Mohri. Rational kernels: Theory and algorithms. *Journal of Machine Learning Research*, 5:1035–1062, Aug. 2004.

[69] N. A. Cressie. *Statistics for Spatial Data*. Wiley, New York, NY, 1993.

[70] F. Curriero. On the use of non-Euclidean distance measures in geostatistics. *Mathematical Geology*, 38(8):907–926, Nov. 2006.

[71] F. L. Custódio, H. J. Barbosa, and L. E. Dardenne. Full-atom ab initio protein structure prediction with a genetic algorithm using a similarity-based surrogate model. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 1–8, Barcelona, Spain., July 2010. IEEE.

[72] N. Dang, L. Pérez Cáceres, P. De Causmaecker, and T. Stützle. Configuring irace using surrogate configuration benchmarks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 243–250, Berlin, Germany, July 2017. ACM.

[73] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[74] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, Apr. 2002.

[75] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, Jan. 2006.

[76] J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, Mar. 2011.

[77] M. Deza and T. Huang. Metrics on permutations, a survey. *Journal of Combinatorics, Information and System Sciences*, 23(1-4):173–185, 1998.

[78] M. M. Deza and E. Deza. *Encyclopedia of Distances*. Springer, Berlin, 3rd edition, 2014.

[79] S. Dowdy, S. Wearden, and D. Chilko. *Statistics for Research*. Wiley, New York, NY, 2004.

[80] K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 1114–1120, Austin, Texas, Jan. 2015. AAAI Press.

[81] A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, July 1999.

[82] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, Berlin, 2nd edition, 2015.

[83] M. Emmerich, A. Giotis, M. Özdemir, T. Bäck, and K. Giannakoglou. Metamodel-assisted evolution strategies. In J. J. Merelo-Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, and J. Fernández-Villacañas, editors, *Parallel Problem Solving from Nature – PPSN VII: 7th International Conference*, volume 2439 of *Lecture Notes in Computer Science*, pages 361–370, Granada, Spain, Sept. 2002. Springer.

[84] M. Emmerich, M. Grötzner, B. Groß, and M. Schütz. Mixed-integer evolution strategy for chemical plant optimization with simulators. In I. C. Parmee, editor, *Evolutionary Design and Manufacture*, pages 55–67. Springer, London, 2000.

[85] S. Fatima and A. Kattan. Evolving optimal agendas for package deal negotiation. In N. Krasnogor, editor, *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO'11)*, pages 505–512, Dublin, Ireland, July 2011. ACM.

[86] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. Wiley, New York, NY, 1971.

[87] R. Filomeno Coelho, M. Herrera, M. Xiao, and W. Zhang. On-line metamodel-assisted optimization with mixed variables. In J. Magalhães Mendes and D. Greiner, editors, *Evolutionary Algorithms and Metaheuristics in Civil Engineering and Construction Management*, volume 39 of *Computational Methods in Applied Sciences*, pages 1–15. Springer, Cham, 2015.

[88] A. Fischbach, M. Zaefferer, J. Stork, M. Friese, and T. Bartz-Beielstein. From real world data to test functions. In F. Hoffmann, E. Hüllermeier, and R. Mikut, editors, *Proceedings 26. Workshop Computational Intelligence*, pages 159–177, Dortmund, Germany, Nov. 2016. KIT Scientific Publishing, Karlsruhe, Germany.

[89] C. Flamm, I. L. Hofacker, P. F. Stadler, and M. T. Wolfinger. Barrier trees of degenerate landscapes. *Zeitschrift für Physikalische Chemie*, 216(2):155–173, Jan. 2002.

[90] O. Flasch. *A modular genetic programming system.* PhD thesis, Technische Universität Dortmund, Dortmund, Germany, May 2015.

[91] O. Flasch, O. Mersmann, T. Bartz-Beielstein, J. Stork, and M. Zaefferer. rgp: R genetic programming framework, version 0.4-1. Online, Aug. 2014. Available: `https://cran.r-project.org/src/contrib/Archive/rgp/`, accessed: 2018-07-11.

[92] L. G. Fonseca, H. J. C. Barbosa, and A. C. C. Lemonge. A similarity-based surrogate model for expensive evolutionary optimization with fixed budget of simulations. In *2009 IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 867–874, Trondheim, Norway, May 2009. IEEE.

[93] A. Forrester, A. Sobester, and A. Keane. *Engineering Design via Surrogate Modelling*. Wiley, New York, NY, 2008.

[94] J. Gablonsky and C. Kelley. A locally-biased form of the DIRECT algorithm. *Journal of Global Optimization*, 21(1):27–37, Sept. 2001.

[95] C. Gagné, M. Schoenauer, M. Sebag, and M. Tomassini. Genetic programming for kernel-based learning with co-evolving subsets selection. In T. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervós, L. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature – PPSN IX: 9th International Conference*, volume 4193 of *Lecture Notes in Computer Science*, pages 1008–1017, Reykjavik, Iceland, 2006. Springer.

[96] A. Gaier, A. Asteroth, and J.-B. Mouret. Data-efficient neuroevolution with kernel-based surrogate models. In H. Aguirre, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 85–92, Kyoto, Japan, July 2018. ACM.

[97] M. Gallagher and B. Yuan. A general-purpose tunable landscape generator. *IEEE Transactions on Evolutionary Computation*, 10(5):590–603, Oct. 2006.

[98] T. Gärtner. A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5(1):49–58, July 2003.

[99] T. Gärtner, J. Lloyd, and P. Flach. Kernels for structured data. In S. Matwin and C. Sammut, editors, *Inductive Logic Programming: 12th International Conference, ILP 2002*, volume 2583 of *Lecture Notes in Computer Science*, pages 66–83, Sydney, Australia, 2003. Springer.

[100] T. Gärtner, J. Lloyd, and P. Flach. Kernels and distances for structured data. *Machine Learning*, 57(3):205–232, Dec. 2004.

[101] W. Gasarch. Classifying problems into complexity classes. In A. Memon, editor, *Advances in Computers*, volume 95, pages 239–292. Elsevier, 2014.

[102] W. Glunt, T. L. Hayden, S. Hong, and J. Wells. An alternating projection algorithm for computing the nearest Euclidean distance matrix. *SIAM Journal on Matrix Analysis and Applications*, 11(4):589–600, 1990.

[103] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, Nov. 1995.

[104] R. Gómez-Bombarelli, J. N. Wei, D. K. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, Jan. 2018.

[105] T. Graepel, R. Herbrich, P. Bollmann-Sdorra, and K. Obermayer. Classification on pairwise proximity data. In M. J. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11 (NIPS 1998)*, pages 438–444, Denver, CO, USA, May 1999. MIT Press.

[106] R. B. Gramacy, H. K. H. Lee, and W. G. Macready. Parameter space exploration with Gaussian process trees. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML 2004)*, pages 45–52, Banff, Alberta, Canada, July 2004. ACM.

[107] R. B. Gramacy and M. A. Taddy. Categorical inputs, sensitivity analysis, optimization and importance tempering with tgp version 2, an R package for treed Gaussian process models. *Journal of Statistical Software*, 33(6):1–48, Feb. 2010.

[108] R.-R. Griffiths and J. M. Hernández-Lobato. Constrained Bayesian optimization for automatic chemical design. *ArXiv e-prints*, June 2018. ArXiv ID: 1709.05501v5.

[109] S. Gu and Y. Guo. Learning SVM classifiers with indefinite kernels. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI'12)*, pages 942–948, Toronto, Ontario, Canada, July 2012. AAAI Press.

[110] A. Gupta, Y.-S. Ong, and L. Feng. Insights on transfer optimization: Because experience is the best teacher. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):51–64, Feb. 2018.

[111] B. Haasdonk. Feature space interpretation of SVMs with indefinite kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):482–492, Apr. 2005.

[112] N. Hansen, A. Auger, O. Mersmann, T. Tusar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *ArXiv e-prints*, Aug. 2016. ArXiv ID: 1603.08785v3.

[113] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Research Report RR-6829, inria-00362633, INRIA, Feb. 2009. Available: `https://hal.inria.fr/inria-00362633`, accessed: 2018-07-11.

[114] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York, 1st edition, 2001.

[115] M. Hauschild and M. Pelikan. An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation*, 1(3):111–128, Sept. 2011.

[116] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, Department of Computer Science, University of California at Santa Cruz, Jan. 1999. Available: `https://www.soe.ucsc.edu/research/technical-reports/UCSC-CRL-99-10`, accessed: 2018-07-11.

[117] T. Hemker. *Derivative Free Surrogate Optimization for Mixed-Integer Nonlinear Black Box Problems in Engineering*. PhD thesis, Technische Universität Darmstadt, Dec. 2008.

[118] J. M. Hernández-Lobato, J. Requeima, E. Pyzer-Knapp, and A. Aspuru-Guzik. Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, volume 70 of *Proceedings of Machine Learning Research*, pages 1470–1479, Sydney, Australia, Aug. 2017. PMLR.

[119] M. Herrera, A. Guglielmetti, M. Xiao, and R. Filomeno Coelho. Metamodel-assisted optimization based on multiple kernel regression for mixed variables. *Structural and Multidisciplinary Optimization*, 49(6):979–991, June 2014.

[120] N. J. Higham. Computing the nearest correlation matrix – a problem from finance. *IMA Journal of Numerical Analysis*, 22(3):329–343, July 2002.

[121] T. Hildebrandt and J. Branke. On using surrogates with genetic programming. *Evolutionary Computation*, 23(3):343–367, June 2015.

[122] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, June 1975.

[123] J. L. Hodges and E. L. Lehmann. Estimates of location based on rank tests. *The Annals of Mathematical Statistics*, 34(2):598–611, June 1963.

[124] H. H. Hoos and T. Stützle. Evaluating Las Vegas algorithms: Pitfalls and remedies. In G. F. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI'98)*, pages 238–245, Madison, WI, USA, July 1998. Morgan Kaufmann.

[125] D. Horn and B. Bischl. Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, Athens, Greece, Dec. 2016. IEEE.

[126] F. Hutter. *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University of British Columbia, Vancouver, Canada, Oct. 2009.

[127] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration (extended version). Technical Report TR-2010-10, University of British Columbia, Department of Computer Science, 2010. Available: `http://www.cs.ubc.ca/~hutter/papers/10-TR-SMAC.pdf`, accessed: 2018-07-11.

[128] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. A. Coello Coello, editor, *Learning and Intelligent Optimization: 5th International Conference, LION 5*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523, Rome, Italy, Jan. 2011. Springer.

[129] F. Hutter and M. A. Osborne. A kernel for hierarchical parameter spaces. *ArXiv e-prints*, Oct. 2013. ArXiv ID: 1310.5738.

[130] K. Ikramov and N. Savel'eva. Conditionally definite matrices. *Journal of Mathematical Sciences*, 98(1):1–50, 2000.

[131] D. Izzo, C. Sprague, and D. Tailor. Machine learning and evolutionary techniques in interplanetary trajectory design. *ArXiv e-prints*, Feb. 2018. ArXiv ID: 1802.00180v1.

[132] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer, New York, 8th edition, 2017.

[133] R. Jenatton, C. Archambeau, J. González, and M. Seeger. Bayesian optimization with tree-structured dependencies. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, volume 70 of *Proceedings of Machine Learning Research*, pages 1655–1664, Sydney, Australia, Aug. 2017. PMLR.

[134] S. Jiang, A. Ziver, J. Carter, C. Pain, A. Goddard, S. Franklin, and H. Phillips. Estimation of distribution algorithms for nuclear reactor fuel management optimisation. *Annals of Nuclear Energy*, 33(11–12):1039–1057, Aug. 2006.

[135] Y. Jiao and J.-P. Vert. The Kendall and Mallows kernels for permutations. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, volume 37 of *Proceedings of Machine Learning Research*, pages 1935–1944, Lille, France, July 2015. PMLR.

[136] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, Jan. 2005.

[137] Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, June 2011.

[138] S. G. Johnson. The NLopt nonlinear-optimization package. Online, 2008. Available: `http://ab-initio.mit.edu/nlopt`, accessed: 2018-07-11.

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

[140] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, Dec. 1998.

[141] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, Pittsburgh, PA, USA, July 1995. Morgan Kaufmann.

[142] A. G. Journel and C. J. Huijbregts. *Mining Geostatistics*. Academic Press, London, 1978.

[143] C. Jung, M. Zaefferer, T. Bartz-Beielstein, and G. Rudolph. Metamodel-based optimization of hot rolling processes in the metal industry. *The International Journal of Advanced Manufacturing Technology*, 90(1):421–435, 2017.

[144] L. Kallel and M. Schoenauer. Fitness distance correlation for variable length representations. Technical Report 363, CMAP, Ecole Polytechnique, 1996. Available: `www.cmap.polytechnique.fr/preprint/repository/old/RI/1997/kallel_schoenauer_363.sept.ps.gz`, accessed: 2018-07-11.

[145] A. Kattan and Y.-S. Ong. Surrogate genetic programming: A semantic aware evolutionary search. *Information Sciences*, 296:345–359, Mar. 2015.

[146] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, New York, 1993.

[147] M. Kendall and J. Gibbons. *Rank Correlation Methods*. Charles Griffin Book Series. Oxford University Press, London, 5th edition, 1990.

[148] H.-S. Kim and S.-B. Cho. An efficient genetic algorithm with less fitness evaluation by clustering. In *Proceedings of the Congress on Evolutionary Computation 2001 (CEC 2001)*, volume 2, pages 887–894, Seoul, Korea, May 2001. IEEE.

[149] J. Klockgether and H. P. Schwefel. Two-phase nozzle and hollow core jet experiments. In D. G. Elliott, editor, *Proceedings of the 11th Symposium on Engineering Aspects of Magnetohydrodynamics*, pages 141–148, Pasadena, CA, USA, Mar. 1970.

[150] P. Koch, B. Bischl, O. Flasch, T. Bartz-Beielstein, C. Weihs, and W. Konen. Tuning and evolution of support vector kernels. *Evolutionary Intelligence*, 5(3):153–170, May 2012.

[151] R. I. Kondor and J. D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In C. Sammut and A. G. Hoffmann, editors, *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'02)*, pages 315–322, Sydney, Australia, July 2002. Morgan Kaufmann.

[152] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Complex Adaptive Systems. MIT Press, Cambridge, MA, 1992.

[153] W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, Dec. 1952.

[154] M. H. Kutner, C. J. Nachtsheim, J. Neter, and W. Li. *Applied Linear Statistical Models*. McGraw-Hill/Irwin, Boston, 5th edition, 2005.

[155] F. Lane, R. M. A. Azad, and C. Ryan. Principled evolutionary algorithm search operator design and the kernel trick. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–9, Athens, Greece, Dec. 2016. IEEE.

[156] C. Lantuéjoul. *Geostatistical Simulation: Models and Algorithms*. Springer-Verlag Berlin Heidelberg, Berlin, 2002.

[157] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Combinatorial optimization by learning and simulation of Bayesian networks. In C. Boutilier and M. Goldszmidt, editors, *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'00)*, pages 343–352, Stanford, CA, USA, July 2000. Morgan Kaufmann.

[158] P. Larrañaga, C. Kuijpers, R. Murga, and Y. Yurramendi. Learning Bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 26(4):487–493, July 1996.

[159] M. LeBlanc and R. Tibshirani. Combining estimates in regression and classification. *Journal of the American Statistical Association*, 91(436):1641–1650, Dec. 1996.

[160] C. Lee. Some properties of nonbinary error-correcting codes. *IRE Transactions on Information Theory*, 4(2):77–82, June 1958.

[161] D. Li and E. Kanoulas. Bayesian optimization for optimizing retrieval systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining - WSDM '18*, pages 360–368, Marina Del Rey, CA, USA, Feb. 2018. ACM.

[162] H. Li and T. Jiang. A class of edit kernels for SVMs to predict translation initiation sites in eukaryotic mRNAs. In *Proceedings of the Eighth Annual International Conference on Research in Computational Molecular Biology (RECOMB'04)*, pages 262–271, San Diego, CA, USA, Mar. 2004. ACM.

[163] R. Li, M. T. Emmerich, J. Eggermont, T. Bäck, M. Schütz, J. Dijkstra, and J. Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary Computation*, 21(1):29–64, Mar. 2013.

[164] R. Li, M. T. M. Emmerich, J. Eggermont, E. G. P. Bovenkamp, T. Bäck, J. Dijkstra, and J. Reiber. Metamodel-assisted mixed integer evolution strategies and their application to intravascular ultrasound image analysis. In *2008 IEEE Congress on Evolutionary Computation (CEC 2008)*, pages 2764–2771, Hong Kong, China, June 2008. IEEE.

[165] S. Liwicki, S. Zafeiriou, G. Tzimiropoulos, and M. Pantic. Efficient online subspace learning with an indefinite kernel for visual tracking and recognition. *IEEE Transactions on Neural Networks and Learning Systems*, 23(10):1624–1636, Oct. 2012.

[166] G. Loosli, S. Canu, and C. Ong. Learning SVM in Krein spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(6):1204–1216, June 2015.

[167] R. Luss and A. d'Aspremont. Support vector machine classification with indefinite kernels. *Mathematical Programming Computation*, 1(2-3):97–118, Oct. 2009.

[168] C. L. Mallows. Non-null ranking models. *Biometrika*, 44(1–2):114–130, June 1957.

[169] J. G. Manchuk and C. V. Deutsch. Robust solution of normal (Kriging) equations. Technical report, CCG Alberta, 2007. Available: `www.ccgalberta.com/ccgresources/report09/2007-401_robust_solution_to_kriging_equations.pdf`, accessed: 2018-07-11.

[170] O. L. Mangasarian. Generalized support vector machines. In A. J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 135–146. MIT Press, Cambridge, MA, 2000.

[171] P.-F. Marteau and S. Gibet. On recursive edit distance kernels with application to time series classification. *IEEE Transactions on Neural Networks and Learning Systems*, 26(6):1121–1133, July 2014.

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

[173] E. A. Mehmet Gönen. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12:2211–2268, July 2011.

[174] Y. Mei, S. Nguyen, B. Xue, and M. Zhang. An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(5):339–353, Sept. 2017.

[175] J. Mockus, V. Tiešis, and A. Žilinskas. The application of Bayesian methods for seeking the extremum. In L. C. W. Dixon and G. P. Szegö, editors, *Towards Global Optimization*, volume 2, pages 117–129. North-Holland Publishing Company, Amsterdam, Netherlands, 1978.

[176] H. Mohammadi, R. Le Riche, N. Durrande, E. Touboul, and X. Bay. An analytic comparison of regularization methods for Gaussian processes. *ArXiv e-prints*, May 2017. ArXiv ID: 1602.00853v3.

[177] D. C. Montgomery. *Design and Analysis of Experiments*. Wiley, New York, NY, 5th edition, 1997.

[178] A. Moraglio. *Towards a Geometric Unification of Evolutionary Algorithms*. PhD thesis, Department of Computer Science, University of Essex, Colchester, Essex, UK, Nov. 2007.

[179] A. Moraglio and A. Kattan. Geometric generalisation of surrogate model based optimisation to combinatorial spaces. In P. Merz and J.-K. Hao, editors, *Evolutionary Computation in Combinatorial Optimization: 11th European Conference, EvoCOP 2011*, volume 6622 of *Lecture Notes in Computer Science*, pages 142–154, Torino, Italy, 2011. Springer.

[180] A. Moraglio and A. Kattan. Geometric surrogate model based optimisation for genetic programming: Initial experiments. Technical report, University of Birmingham, 2011. Available: `www.ahmedkattan.com/index_files/surrogate-gp-tech-rep.pdf`, accessed: 2018-07-11.

[181] A. Moraglio, Y.-H. Kim, and Y. Yoon. Geometric surrogate-based optimisation for permutation-based problems. In N. Krasnogor, editor, *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO'11)*, pages 133–134, Dublin, Ireland, July 2011. ACM.

[182] A. Moraglio and R. Poli. Geometric landscape of homologous crossover for syntactic trees. In *2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 427–434, Edinburgh, Scotland, UK, Sept. 2005. IEEE.

[183] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, Mar. 1981.

[184] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, Jan. 2009.

[185] J. Mueller, D. Gifford, and T. Jaakkola. Sequence to better sequence: Continuous revision of combinatorial structures. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, volume 70 of *Proceedings of Machine Learning Research*, pages 2536–2544, Sydney, Australia, Aug. 2017. PMLR.

[186] A. Muñoz and I. M. de Diego. From indefinite to positive semi-definite matrices. In D.-Y. Yeung, J. T. Kwok, A. Fred, F. Roli, and D. de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops, SSPR 2006 and SPR 2006*, volume 4109 of *Lecture Notes in Computer Science*, pages 764–772, Hong Kong, China, 2006. Springer.

[187] K. P. Murphy. *Machine Learning*. MIT Press, Cambridge, MA, 2012.

[188] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley, New York, NY, 2016.

[189] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, Jan. 1965.

[190] S. Nguyen, Y. Mei, and M. Zhang. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems*, 3(1):41–66, Feb. 2017.

[191] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan. Selection schemes in surrogate-assisted genetic programming for job shop scheduling. In G. Dick, W. N. Browne, P. Whigham, M. Zhang, L. T. Bui, H. Ishibuchi, Y. Jin, X. Li, Y. Shi, P. Singh, K. C. Tan, and K. Tang, editors, *Simulated Evolution and Learning: 10th International Conference, SEAL 2014*, volume 8886 of *Lecture Notes in Computer Science*, pages 656–667, Dunedin, New Zealand, 2014. Springer.

[192] S. Nguyen, M. Zhang, and K. C. Tan. Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. *IEEE Transactions on Cybernetics*, 47(9):2951–2965, May 2016.

[193] C. S. Ong, X. Mary, S. Canu, and A. J. Smola. Learning with non-positive kernels. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML 2004)*, pages 81–88, Banff, Alberta, Canada, July 2004. ACM.

[194] B. Paaßen, C. Göpfert, and B. Hammer. Gaussian process prediction for time series of structured data. In M. Verleysen, editor, *Proceedings of the ESANN, 24th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 41–46, Bruges, Belgium, Apr. 2016. ESANN.

[195] M. Pawlik and N. Augsten. APTED release 0.1.1. Online, Feb. 2016. Available: `https://github.com/DatabaseGroup/apted`, accessed: 2018-07-11.

[196] M. Pawlik and N. Augsten. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157–173, Mar. 2016.

[197] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz. BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. M. Daida, A. E. Eiben, M. H. Garzon, and V. Honavar, editors, *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation (GECCO'99)*, volume 1, pages 525–532, Orlando, FL, USA, July 1999. Morgan Kaufmann.

[198] L. Pérez Cáceres, B. Bischl, and T. Stützle. Evaluating random forest models for irace. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 1146–1153, Berlin, Germany, July 2017. ACM.

[199] L. Pérez Cáceres, M. López-Ibáñez, and T. Stützle. Ant colony optimization on a limited budget of evaluations. *Swarm Intelligence*, 9(2–3):1–22, Sept. 2015.

[200] M. Pilát and R. Neruda. Feature extraction for surrogate models in genetic programming. In J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, editors, *Parallel Problem Solving from Nature – PPSN XIV: 14th International Conference*, volume 9921 of *Lecture Notes in Computer Science*, pages 335–344, Edinburgh, UK, 2016. Springer.

[201] A. Pintore and C. C. Holmes. Spatially adaptive non-stationary covariance functions via spatially adaptive spectra. Technical report, University of Oxford, Department of Statistics, 2004. Available `http://www.stats.ox.ac.uk/~cholmes/Reports/spectral_tempering.pdf`, accessed: 2017-09-11.

[202] T. Pohlert. The pairwise multiple comparison of mean ranks package (PMCMR) R package, version 4.3. Online, May 2018. Available: `http://CRAN.R-project.org/package=PMCMR`, accessed: 2018-07-11.

[203] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. Cambridge University Press, Cambridge, 2nd edition, 1992.

[204] M. Preuss, G. Rudolph, and S. Wessing. Tuning optimization algorithms for real-world problems by means of surrogate modeling. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO'10)*, pages 401–408, Portland, OR, USA, July 2010. ACM.

[205] R Core Team. R: A language and environment for statistical computing. Online, 2018. Available: `https://www.R-project.org/`, accessed: 2018-07-11. R Foundation for Statistical Computing. Vienna, Austria.

[206] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

[207] R. Rebonato and P. Jäckel. The most general methodology to create a valid correlation matrix for risk management and option pricing purposes. *Journal of Risk*, 2(2):17–27, Jan. 2000.

[208] I. Rechenberg. *Evolutionsstrategie '94*. Frommann-Holzboog, Stuttgart, 1994.

[209] C. R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13, Jan. 1995.

[210] C. R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86(0):473–490, Jan. 1999.

[211] G. Reinelt. TSPLIB – a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, Nov. 1991.

[212] P. A. Romero, A. Krause, and F. H. Arnold. Navigating the protein fitness landscape with Gaussian processes. *Proceedings of the National Academy of Sciences*, 110(3):193–201, 2013.

[213] R. Roth. *Introduction to Coding Theory*. Cambridge University Press, New York, 2006.

[214] W. Rowe, M. Platt, D. C. Wedge, P. J. Day, D. B. Kell, and J. Knowles. Analysis of a complete DNA-protein affinity landscape. *Journal of The Royal Society Interface*, 7(44):397–408, July 2009.

[215] W. Rowe, D. C. Wedge, M. Platt, D. B. Kell, and J. Knowles. Predictive models for population performance on real biological fitness landscapes. *Bioinformatics*, 26(17):2145–2152, Sept. 2010.

[216] G. Rudolph. Global optimization by means of distributed evolution strategies. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature – 1st Workshop, PPSN I, 1990*, volume 496 of *Lecture Notes in Computer Science*, pages 209–213, Dortmund, Germany, 1991. Springer.

[217] G. Rudolph. Self-adaptive mutations may lead to premature convergence. *IEEE Transactions on Evolutionary Computation*, 5(4):410–414, Aug. 2001.

[218] G. Rudolph, M. Preuss, and J. Quadflieg. Two-layered surrogate modeling for tuning optimization metaheuristics. Technical Report TR09-2-005, TU Dortmund, Dortmund, Germany, Sept. 2009. Algorithm Engineering Report.

[219] H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*, volume 104 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London, 2005.

[220] J. Sacks, W. J. Welch, T. J. Mitchell, H. P. Wynn, et al. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423, Nov. 1989.

[221] T. Schiavinotto and T. Stützle. A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34(10):3143–3153, Oct. 2007.

[222] F.-M. Schleif and P. Tino. Indefinite proximity learning: A review. *Neural Computation*, 27(10):2039–2096, Oct. 2015.

[223] B. Schölkopf. The kernel trick for distances. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13 (NIPS 2000)*, pages 301–307, Breckenridge, CO, USA, 2001. MIT Press.

[224] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, 2001.

[225] H.-P. Schwefel. Experimentelle Optimierung einer Zweiphasendüse, Teil I. Technical Report 11.034/68, AEG Forschungsinstitut, Berlin, Oct. 1968. Bericht Nr. 35 zum ProjektMHD – Staustrahlrohr.

[226] H.-P. Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkhäuser, Basel, 1977.

[227] H.-P. Schwefel. Collective phenomena in evolutionary systems. In P. Checkland and I. Kiss, editors, *Problems of Constancy and Change – The Complementarity of Systems Approaches to Complexity, 31st Annual Meeting*, volume 2, pages 1025–1033, Budapest, Hungary, June 1987. International Society for General Systems Research.

[228] H.-P. Schwefel. *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology. Wiley, New York, NY, 1995.

[229] M. Sevaux and K. Sörensen. Permutation distance measures for memetic algorithms with population management. In *Proceedings of 6th Metaheuristics International Conference (MIC 2005)*, pages 832–838, Vienna, Austria, Aug. 2005. University of Vienna.

[230] S. Shakya. *DEUM: A Framework for an Estimation of Distribution Algorithm based on Markov Random Fields*. PhD thesis, School of Computing, Faculty of Design and Technology, The Robert Gordon University, Aberdeen, UK, Apr. 2006.

[231] Q. Shao, T. Xu, T. Yoshino, and N. Song. Multi-objective optimization of gas metal arc welding parameters and sequences for low-carbon steel (Q345D) T-joints. *Journal of Iron and Steel Research, International*, 24(5):544–555, May 2017.

[232] T. Simpson, V. Toropov, V. Balabanov, and F. Viana. Design and analysis of computer experiments in multidisciplinary design optimization: A review of how far we have come – or not. In *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, pages 1–22, Victoria, BC, Canada, Sept. 2008. American Institute of Aeronautics and Astronautics.

[233] A. Singhal. Modern information retrieval: A brief overview. *IEEE Bulletin of the Technical Committee on Data Engineering*, 24(4):35–43, Dec. 2001.

[234] M. A. Skala. *Aspects of Metric Spaces in Computation*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, June 2008.

[235] J. Smith, C. Stone, and M. Serpell. Exploiting diverse distance metrics for surrogate-based optimisation of ordering problems: A case study. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO'16)*, pages 701–708, Denver, CO, USA, July 2016. ACM.

[236] R. E. Smith, B. A. Dike, and S. A. Stegmann. Fitness inheritance in genetic algorithms. In K. M. George, J. Carroll, and D. Oppenheim, editors, *Proceedings of the 1995 ACM Symposium on Applied Computing (SAC '95)*, pages 345–350, Nashville, TE, USA, Feb. 1995. ACM.

[237] A. J. Smola and R. Kondor. Kernels and regularization on graphs. In B. Schölkopf and M. K. Warmuth, editors, *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003*, volume 2777 of *Lecture Notes in Computer Science*, pages 144–158, Washington, DC, USA, 2003. Springer.

[238] A. J. Smola, Z. L. Ovári, and R. C. Williamson. Regularization with dot-product kernels. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13 (NIPS 2000)*, pages 308–314, Breckenridge, CO, USA, 2001. MIT Press.

[239] J. Stork, M. Zaefferer, and T. Bartz-Beielstein. Distance-based kernels for surrogate model-based neuroevolution. *ArXiv e-prints*, July 2018. Accepted to the Developmental Neural Networks Workshop at the Parallel Problem Solving from Nature 2018 (PPSN XV) conference. ArXiv ID: 1807.07839.

[240] J. R. Stroud, M. L. Stein, and S. Lysen. Bayesian and maximum likelihood estimation for Gaussian processes on an incomplete lattice. *Journal of Computational and Graphical Statistics*, 26(1):108–120, Feb. 2017.

[241] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. Osborne. Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. *ArXiv e-prints*, Sept. 2014. NIPS workshop on Bayesian Optimization in Theory and Practice (BayesOpt2013), ArXiv ID: 1409.4011v1.

[242] L. P. Swiler, P. D. Hough, P. Qian, X. Xu, C. Storlie, and H. Lee. Surrogate models for mixed discrete-continuous variables. In M. Ceberio and V. Kreinovich, editors, *Constraint Programming and Decision Making*, volume 539 of *Studies in Computational Intelligence*, pages 181–202. Springer, Cham, 2014.

[243] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, July 1990.

[244] C. Teixeira, J. Covas, T. Stützle, and A. Gaspar-Cunha. Optimization of co-rotating twin-screw extruders using Pareto local search. In X.-Z. Gao, G.-C. António, M. Köppen, G. Schaefer, and J. Wang, editors, *Advances in Intelligent and Soft Computing*, volume 75 of *Advances in Intelligent and Soft Computing*, pages 3–10. Springer, Berlin, 2010.

[245] C. Teixeira, J. A. Covas, T. Stützle, and A. Gaspar-Cunha. Multi-objective ant colony optimization for the twin-screw configuration problem. *Engineering Optimization*, 44(3):351–371, Feb. 2012.

[246] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA. In R. Ghani, T. E. Senator, P. Bradley, R. Parekh, and J. He, editors, *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pages 847–855, Chicago, IL, USA, Aug. 2013. ACM.

[247] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[248] M. W. Trosset. The Krigifier: A procedure for generating pseudorandom nonlinear objective functions for computational experimentation. Technical Report 35NASA/CR-1999-209000, ICASE-IR-35, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA, USA, Feb. 1999. Available: `https://ntrs.nasa.gov/search.jsp?R=19990028613`, accessed: 2018-07-11.

[249] M. P. van der Loo. The stringdist package for approximate string matching. *The R Journal*, 6(1):111–122, Apr. 2014.

[250] B. van Stein, H. Wang, W. Kowalczyk, T. Bäck, and M. Emmerich. Optimally weighted cluster Kriging for big data regression. In E. Fromont, T. De Bie, and M. van Leeuwen, editors, *Advances in Intelligent Data Analysis XIV, 14th International Symposium, IDA 2015*, volume 9385 of *Lecture Notes in Computer Science*, pages 310–321, Saint Etienne, France, 2015. Springer.

[251] V. N. Vapnik. *Statistical Learning Theory*. Wiley, New York, NY, 1998.

[252] S. Verel, B. Derbel, A. Liefooghe, H. Aguirre, and K. Tanaka. A surrogate model based on walsh decomposition for pseudo-boolean functions. In A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, editors, *Parallel Problem Solving from Nature – PPSN XV: 15th International Conference*, volume 11102 of *Lecture Notes in Computer Science*, pages 181–193, Coimbra, Portugal, Sept. 2018. Springer.

[253] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, Apr. 2010.

[254] I. Voutchkov, A. Keane, A. Bhaskar, and T. M. Olsen. Weld sequence optimization: The use of surrogate models for solving sequential combinatorial problems. *Computer Methods in Applied Mechanics and Engineering*, 194(30–33):3535–3551, Aug. 2005.

[255] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, Jan. 1974.

[256] T. Wagner. *Planning and Multi-Objective Optimization of Manufacturing Processes by Means of Empirical Surrogate Models*. PhD thesis, Technische Universität Dortmund, Dortmund, Germany, July 2013.

[257] G. G. Wang and S. Shan. Review of metamodeling techniques in support of engineering design optimization. *Journal of Mechanical Design*, 129(4):370–380, 2007.

[258] H. Wang, B. van Stein, M. Emmerich, and T. Bäck. Time complexity reduction in efficient global optimization using cluster Kriging. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 889–896, Berlin, Germany, July 2017. ACM.

[259] L. D. Whitley, K. E. Mathias, S. Rana, and J. Dzubera. Building better test functions. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 239–247, Pittsburgh, PA, USA, July 1995. Morgan Kaufmann.

[260] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13 (NIPS 2000)*, pages 682–688, Breckenridge, CO, USA, 2001. MIT Press.

[261] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, July 1992.

[262] G. Wu, E. Y. Chang, and Z. Zhang. An analysis of transformation on non-positive semidefinite similarity matrix for kernel machines. Technical report, University of California, Santa Barbara, CA, USA, June 2005.

[263] M. Zaefferer. Combinatorial efficient global optimization in R - CEGO v2.2.0. Online, 2017. Available: `https://cran.r-project.org/package=CEGO`, accessed: 2018-11-07.

[264] M. Zaefferer and T. Bartz-Beielstein. Efficient global optimization with indefinite kernels. In J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, editors, *Parallel Problem Solving from Nature – PPSN XIV: 14th International Conference*, volume 9921 of *Lecture Notes in Computer Science*, pages 69–79, Edinburgh, UK, 2016. Springer.

[265] M. Zaefferer, T. Bartz-Beielstein, B. Naujoks, T. Wagner, and M. Emmerich. A case study on multi-criteria optimization of an event detection software under limited budgets. In R. C. Purshouse, P. J. Fleming, C. M. Fonseca, S. Greco, and J. Shaw, editors, *Evolutionary Multi-Criterion Optimization: 7th International Conference, EMO 2013*, volume 7811 of *Lecture Notes in Computer Science*, pages 756–770, Sheffield, UK, 2013. Springer.

[266] M. Zaefferer, T. Bartz-Beielstein, and G. Rudolph. An empirical approach for probing the definiteness of kernels. *Soft Computing*, 2018. DOI: 10.1007/s00500-018-3648-1.

[267] M. Zaefferer, B. Breiderhoff, B. Naujoks, M. Friese, J. Stork, A. Fischbach, O. Flasch, and T. Bartz-Beielstein. Tuning multi-objective optimization algorithms for cyclone dust separators. In C. Igel, editor, *Proceedings of the 2014*

Annual Conference on Genetic and Evolutionary Computation (GECCO'14), pages 1223–1230, Vancouver, BC, Canada, July 2014. ACM.

[268] M. Zaefferer, A. Fischbach, B. Naujoks, and T. Bartz-Beielstein. Simulation based test functions for optimization algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 905–912, Berlin, Germany, July 2017. ACM.

[269] M. Zaefferer, D. Gaida, and T. Bartz-Beielstein. Multi-fidelity modeling and optimization of biogas plants. *Applied Soft Computing*, 48:13–28, Nov. 2016.

[270] M. Zaefferer and D. Horn. A first analysis of kernels for kriging-based optimization in hierarchical search spaces. In A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, editors, *Parallel Problem Solving from Nature – PPSN XV: 15th International Conference*, volume 11102 of *Lecture Notes in Computer Science*, pages 399–410, Coimbra, Portugal, Sept. 2018. Springer.

[271] M. Zaefferer, J. Stork, and T. Bartz-Beielstein. Distance measures for permutations in combinatorial efficient global optimization. In T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith, editors, *Parallel Problem Solving from Nature – PPSN XIII: 13th International Conference*, volume 8672 of *Lecture Notes in Computer Science*, pages 373–383, Ljubljana, Slovenia, 2014. Springer.

[272] M. Zaefferer, J. Stork, O. Flasch, and T. Bartz-Beielstein. Linear combination of distance measures for surrogate models in genetic programming. In A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, editors, *Parallel Problem Solving from Nature – PPSN XV: 15th International Conference*, volume 11102 of *Lecture Notes in Computer Science*, pages 220–231, Coimbra, Portugal, Sept. 2018. Springer.

[273] M. Zaefferer, J. Stork, M. Friese, A. Fischbach, B. Naujoks, and T. Bartz-Beielstein. Efficient global optimization for combinatorial problems. In C. Igel, editor, *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO'14)*, pages 871–878, Vancouver, BC, Canada, July 2014. ACM.

[274] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, Aug. 2009.

[275] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-norm support vector machines. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16 (NIPS 2003)*, pages 49–56, Whistler, BC, Canada, 2004. MIT Press.

[276] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131(1-4):373–395, Oct. 2004.

Notation

$a, b, c, \ldots$ scalars

x special case: arbitrary data type, e.g., a scalar, vector, string, permutation, or graph.

$\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$ vectors

a_i i-th element of a vector, string, or sequence

$\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots$ matrices

$\mathbf{A}_{ij}$ element in i-th row and j-th column of a matrix

$A, B, C, \ldots$ sets

$|A|$ cardinality of a set (number of elements in the set)

$a^{(i)}$ i-th element of the set A

$a_j^{(i)}$ j-th vector element of $a^{(i)}$

$a', a'', a^*, a_{\text{text}}$ different instances of a

$\{a, b, c\}$ a set with the elements a, b, and c.

$\begin{bmatrix} a & b & c \end{bmatrix}$ a row vector (or sequence, permutation) with the elements a, b, and c.

$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} a & b & c \end{bmatrix}^{\mathrm{T}}$ respectively, a column vector

$[a, b]$ an interval, running from a to b, including a and b

$a = 1, 2, \ldots, b$ shorthand for $a = 1$, $a = 2$, …, $a = b$

List of Symbols

Acronyms

k-NN	k-nearest neighbors, page 30
ACO	ant colony optimization, page 36
ANN	artificial neural network, page 21
ANOVA	analysis of variance, page 172
ATSP	asymmetric traveling salesperson problem, page 51
CEGO	combinatorial efficient global optimization, page 49
CNSD	conditionally negative semi-definite, page 43
COCO	comparing continuous optimizers, page 148
CPSD	conditionally positive semi-definite, page 43
CV	cross-validation, page 66
CVM	Cramèr-von Mises, page 109
DIRECT	dividing rectangles algorithm, page 54
EA	evolutionary algorithm, page 16
ECDF	empirical cumulative distribution function, page 159
EDA	estimation of distribution algorithm, page 36
EGO	efficient global optimization, page 27
EI	expected improvement, page 27
ES	evolution strategy, page 18
FDC	fitness distance correlation, page 64
FDDC	fitness-difference distance correlation, page 65

FSP	flow-shop scheduling problem, page 51
GLG	Gaussian landscape generator, page 148
GP	genetic programming, page 120
KR	Kriging, page 157
LASSO	least absolute shrinkage and selection operator, page 68
LCB	lower confidence bound, page 27
LCStr	longest common substring, page 52
LOESS	locally weighted regression, page 131
MC	max-cut, page 52
MIES	mixed-integer evolution strategy, page 19
MLE	maximum likelihood estimation, page 23
MRF	Markov random fields, page 37
MSE	mean square error, page 130
ND	negative definite, page 43
NKL	NK-landscapes, page 52
NSD	negative semi-definite, page 43
OPD	optimization for probing definiteness, page 85
OSA	optimal string alignment, page 180
PD	positive definite, page 43
PhD	phenotypic distance, page 124
PI	probability of improvement, page 27
PSD	positive semi-definite, page 41
QAP	quadratic assignment problem, page 51
QQ	quantile-quantile, page 174
RBFN	radial basis function network, page 21
RF	random forest, page 21

RMSE	root mean square error, page 92
RS	random search, page 52
RSPD	random sampling for probing definiteness, page 84
SHD	structural Hamming distance, page 125
SHD1	structural Hamming distance, original variant without alignment, page 125
SHD2	structural Hamming distance, with alignment, page 125
SMBO	surrogate model-based optimization, page 9
SR	symbolic regression, page 120
STR	strategy, page 29
SVM	support vector machine, page 21
TED	tree edit distance, page 125
TSP	traveling salesperson problem, page 51
UNI	unimodal problem, page 52
WTP	weighted tardiness problem, page 51

Scalars

α	significance level, page 174
β	model coefficient, page 39
γ	parameter that specifies a chosen distance measure or kernel function, page 66
Δy	fitness difference, page 65
ϵ	error term in a model, page 22
ϵ_{tol}	a small tolerance value that is used when checking for whether a matrix is definite, page 85
η	nugget, regularization constant, page 25
θ	parameter of an exponential kernel function, page 22
κ	condition number of a matrix, page 45
λ	eigenvalue, page 43

λ_1	smallest eigenvalue, page 43
$\hat{\lambda}$	a critical eigenvalue that determines whether a matrix is CNSD, page 44
λ_n	largest eigenvalue, page 43
μ	mean value, page 17
ρ	parameter of a kernel for conditional variables, page 138
σ^2	variance, or step-size, page 17
σ_d	standard deviation of distances, page 64
σ_y	standard deviation of observations, page 64
τ	learning rate for the self-adaptive control of a numerical EA parameter, page 19
a	an index of a string or vector, $a \in \mathbb{N}$, page 186
b	an index of a string or vector, $b \in \mathbb{N}$, page 186
c	arbitrary real-valued coefficient, page 43
$d_{\mathbf{c}}$	number of constant coefficients in a symbolic regression tree, page 122
l_i	lower bound of the i-th variable, page 138
m	dimensionality of a search or input space, number of elements in a vector, page 10
n	number of samples or observations, page 22
n_{bit}	number of bits that contribute to fitness, in case of NK-Landscapes, page 52
n_{b}	number of branches of a tree node, arity, page 125
n_{cat}	number of levels of a categorical variable, page 32
$n_{\lambda+}$	number of solutions that yield indefinite distance matrices, page 84
n_o	number of samples in the context of a larger kernel matrix approximated by the Nyström method, page 106
n_{off}	number of offspring, page 17
n_{pop}	population size, page 17
n_{sim}	number of simulation samples, page 150
n_{test}	number of test functions, page 152

p	parameter of an exponential kernel function, page 22
$p_{\lambda+}$	proportion of solutions that yield indefinite distance matrices, page 84
p_{s}	probability to change a categorical parameter of an EA in self-adaptive parameter control, page 19
q	reciprocal of a distance, page 103
r	number of kernels, or distances, page 64
r_{mut}	mutation rate, page 53
s	input variable of a regression problem, in the context of symbolic regression, page 122
t	number of sets, or matrices, page 84
u_i	upper bound of the i-th variable, page 138
v	scaled variable value, used in the Wedge-kernel, page 141
w	number of reference solutions, page 109
y	observation, outcome of a model or experiment, page 22
y^*	difference between a predicted value and the observed minimum, page 28

Vectors

$\mathbf{\Delta y}$	vector of fitness differences, page 65
$\boldsymbol{\epsilon}$	vector of independent random samples, page 150
$\mathbf{0}$	vector of zeros, page 43
$\mathbf{1}$	vector of ones, page 23
$\mathbf{c}$	arbitrary real-valued vector of coefficients, page 43
$\mathbf{d}$	vector of distances, page 64
$\mathbf{e}$	vector of ones, except for the last element that is the square root of the number of elements, page 102
$\mathbf{k}$	vector of correlations or kernel values between training data and a new data sample, page 24
$\mathbf{y}$	a vector of observations, or fitness values, page 22
$\mathbf{z}$	a vector of independent random samples, page 17

$\hat{\mathbf{y}}$	vector result of estimation or prediction, page 151
$\hat{\mathbf{y}}_{\mathrm{cs}}$	vector result of a conditional simulation, page 151
$\hat{\mathbf{y}}_{\mathrm{s}}$	vector result of a simulation, page 150

Matrices

$\mathbf{\Lambda}$	diagonal matrix of eigenvalues, page 99
$\mathbf{A}$	a square, symmetric, real matrix, page 43
$\mathbf{B}$	matrix used for checking conditional definiteness, page 43
$\mathbf{C}$	covariance matrix, page 23
$\mathbf{C}_{\mathrm{cs}}$	conditioned covariance matrix of simulation samples, page 151
$\mathbf{C}_{\mathrm{s}}$	covariance matrix of simulation samples, page 150
$\mathbf{D}$	distance matrix, page 41
$\hat{\mathbf{D}}$	a matrix computed with Eq. (4.7), to determine whether a distance matrix is CNSD, page 83
$\mathbf{I}$	identity matrix, page 44
$\mathbf{K}$	correlation matrix or kernel matrix, page 22
$\mathbf{K}_{\mathrm{cs}}$	conditioned correlation matrix of simulation samples, page 151
$\mathbf{K}_{\mathrm{s}}$	correlation matrix of simulation samples, page 150
$\mathbf{K}_{\mathrm{x}}$	cross-correlation matrix of simulation samples and training samples, page 151
$\mathbf{P}$	matrix used for checking conditional definiteness, page 43
$\mathbf{Q}$	matrix required for transformation of an indefinite to a CNSD matrix, page 102
$\mathbf{U}$	matrix of eigenvectors, page 99

Sets and Spaces

Π	permutation space, page 180
$\mathbb{B}$	binary string space, page 187
$\mathbb{N}$	natural numbers, page 41
$\mathbb{R}$	real numbers, page 10

$\mathbb{R}^+$	non-negative real numbers, page 41
S^n	search space in the context of OPD, page 85
$\mathcal{X}$	search space, page 10
X	set of candidate solutions, population, $X \subset \mathcal{X}$, page 16
X_s	set of simulation samples, page 150
Z	population, or solution set for OPD, page 85

Functions

$\beta_j(x^*)$	data dependent weight coefficient for the superposition of Kriging models, page 71
$\Delta(x, x')$	recursive term in the structural Hamming distance for trees, page 125
$\delta_i(x)$	function that determines the activity of the i-th conditional variable, page 137
$\epsilon(x)$	the error associated with x , page 22
$\Phi(.)$	normal cumulative distribution function, page 28
$\phi(.)$	normal probability density function, page 28
$\text{arity}(x)$	arity of a tree node x, page 125
$\text{corr}(.,.)$	Pearson correlation of two vectors, page 65
$\text{d}(x, x')$	Distance measure, page 41
$\text{diag}(\mathbf{a})$	returns a diagonal matrix, with the diagonal specified by $\mathbf{a}$, page 99
$\text{EI}(x)$	expected improvement of candidate solution x, page 28
$f(x)$	fitness, objective function, page 10
$F(x, \mathbf{c})$	An upper level fitness/objective function in a bi-level problem, page 122
$\text{HD}(x, x')$	Hamming distance of two labels, or categorical variables, page 125
$\text{h}_i(x)$	a mapping function used to prove definiteness of a kernel, page 138
$I(\hat{y})$	improvement of a value $\hat{y}$ over the best-observed value $\min(\mathbf{y})$, page 27
$\mathbb{I}(.)$	indicator function, page 99
$\text{k}(x, x')$	kernel function, page 10

$L(.)$	likelihood function, page 23
$\min(.)$	minimal element of a vector, page 28
$N(\mu, \sigma^2)$	normal distribution, page 17
sign (.)	sign function, returning a vector with elements 1 for each positive values of the argument and -1 for all negative values, page 99
$\hat{s}^2(x)$	uncertainty estimate of a Kriging model, page 25
$\hat{y}(x)$	predictor of a Kriging model, page 24
$\hat{\mathbf{y}}(x, \mathbf{c})$	output of the symbolic expression x with coefficients $\mathbf{c}$ for a given data set, page 122

Other

x	candidate solution or sample of unspecified data type, $x \in \mathcal{X}$, page 10
$x[0]$	root node of a tree x, page 125
$x[i]$	i-th subtree of the root node of a tree x, page 125
x_{ref}	a reference sample, or reference candidate solution, page 52

Index

(1+1)-ES, 18

ant colony optimization, 35, **36**
archive, 53
artificial neural network, 21
autoencoder, 33

Bayesian networks, 37
Bayesian optimization, 28
bi-level problem, 122
binary string, 187
bit string, *see* binary string
black-box, **10**, 16, 20
box plot, 171

candidate solution, **10**, 16
Cholesky decomposition, 110
combinatorial efficient global optimization, 49, 50
condition number, 45
condition repair, **102**, 107
  linear combination, 103
  nearest neighbor, 103
  re-transformation, 103
constructive solid geometry, 135
contrast, 32
correlation, 22
  fitness distance, 64
  fitness-difference distance, 65
  function, **22**, 41
  Pearson, 64
  Spearman, 64
covariance
  function, 41
  matrix, 23
  population, 64
  sample, 64
cross-validation, 25, **66**
  k-fold, 66

definite, 12, 35, **43**, 82
  adapting, 83
  CNSD-correction, 102
  conditionally, **43**, 44
  designing, 83
  impact on model quality, 90
  indefinite, 12, **43**, 45
    correcting, 98
  negative, **43**, 44
  negative semi-, 43
  NSD-correction, 102
  positive, **43**, 44
  positive semi-, 22, 34, 41, **43**, 44, 60
  probing, 83
    optimization, 85
    random sampling, 84
  proving, 83
  PSD-correction, 101
  strictly, 44
dissimilarity measure, 41
distance, 10, **41**, 48, 178
  adjacency, 182
  Arc-kernel, 138
  Chebyshev, 109, **178**, 183
  cosine, 184
  edit distance, 42
  Euclidean, **178**, 183
  function, 41
  genotypic, 34, **125**
  Hamming, **179**, 183

  hierarchical, 185
  Ico-kernel, 139
  Imp-kernel, 140
  insert, 181
  interchange, 181
  Jaro-Winkler, 180
  L0.5, 109, **178**
  Lee, 184
  Levenshtein, 42, **179**, 180
  lexicographic, 184
  longest common substring, 181
  Manhattan, 183
  measure, 41
  metric, 42
  non-stationary, 106, 109, **179**
  optimal string alignment, 180
  phenotypic, 34, **124**, 185
  position, 182
  pseudo metric, 42
  R-distance, 181
  restricted Damerau Levenshtein, 43
  reversal, 184
  squared position, 183
  structural Hamming distance, **125**, 185
  swap, 48, 180
  t-test, 109, **179**
  tree edit distance, **125**, 184

efficient global optimization, **27**, 35, 49
eigen-decomposition, **98**, 150
estimation of distribution algorithm, 36
evolutionary algorithm, **16**, 50
expected improvement, **27**, 33, 70, 102
experimental design, **20**, 49
  latin hypercube, 21
  max-min, 49
  random sampling, 49

feature embedding, 98, **104**
feature space, **38**, 45
fitness, 16
fitness function, 16
flow-shop scheduling problem, 51

Gaussian process, *see* Kriging
genetic programming, 120

hierarchical search space, 136

identity preservation, 42
ill-conditioning, 26, **45**
individual, 16
infill criterion, **20**, 27

kernel, 10, 11, 22, **38**, 48, 50, 60, 178
  Arc-kernel, 137
  combination, 66
    combineLinear, 68
    combineMLE, 69
    combineStack, 71
    combineSuper, 72
  exponential, 48
  Ico-kernel, 138
  IcoCor-kernel, 139
  Imp-kernel, 139
  multi-kernel, 61
  selection, 63
    selectCV, 66
    selectFDC, 65
    selectFDDC, 65
    selectMLE, 66
    selectRandom, 63
  sigmoid, 45
  single-kernel, 61
  time-warp, 45
  triangular, 141
  trick, **38**, 45
  Wedge-kernel, 141
Kriging, 12, **22–28**, 30, 35, 39, 41, 43, 47, 48, 149
  conditional simulation, 150
  Krigifier, 148
  nugget effect, **25**, 99, 102, 119
  predictor, **24**, 49, 70, 71
  re-interpolation, 26
  simulation, 145, **149**
  superposition, 67, **71**
  test function generator, 151
  treed Gaussian processes, 32

uncertainty estimate, **24**, 49, 71, 105

LASSO, **68**, 69, 70
likelihood, **23**
function, 23
linear regression, **39**, 56, 67
local optimum, 20, **28**
LOESS model, 131
lower confidence bound, 27

Mallows model, 37
Markov random fields, 37
max-cut problem, 52
maximum likelihood estimation, **23**, 49, 65, 68, 110, 153
mixed-integer evolution strategy, 19
model
customized, 30, **31**
feature extraction, 30, **34**
graph, 37
inherently discrete, 30, **32**
kernel-based, 30, **34**, 137
mapping, 30, **32**, 137
naive approach, 29, **30**, 137
probabilistic, 36
simulation, 31
surrogate, 9, **20**
Moore-Penrose inverse, *see* pseudoinverse
multi-dimensional scaling, 100
multi-objective, **22**, 96
multiple kernel learning, **61**, 66, 67
mutation, 16, **17**, 186, 188
bit flip, 188
block inversion, 188
cycle, 188
interchange, 186
rate, 186
reversal, 187
swap, 186

nearest matrix, 98, **104**
NK-landscape, 52
non-stationarity, 105
non-stationary, 71, 179
normal
cumulative distribution function, 28
distribution, 172
multivariate, 23
probability density function, 23, 28
numerical stability, 45
Nyström approximation, 105

objective function, **10**, 16
offspring, 16
one-fifth rule, 18
optimization problem, **10**

parameter
control, 18
learning parameters, 19
meta-parameters, 18
of a kernel, 23
of an EA, 17
problem parameters, 18
self-adaptation, 18
tuning, 18
parent selection, 17
permutation, 10, 48, **180**
population, 16
probability of improvement, 27
pseudoinverse, **99**, 100

QQ plot, *see* quantile-quantile plot
quadratic assignment problem, 51
quantile-quantile plot, 174

radial basis function network, 21, 30, 35, 41
random forest, **21**, 32, 157, 169
random keys, 33
re-transformation, 102
recombination, **16**, 187, 188
1-point crossover, 188
2-point crossover, 188
AND-crossover, 188
alternating position crossover, 187
cycle crossover, 187
dominant crossover, 19
intermediate crossover, 16

    order crossover, 187
    position-based crossover, 187
    uniform crossover, 188
regularization, 26

search space, **10**, 16, 47
selection, 16
self-adaptation, 19
sequential parameter optimization, 18, 21
similarity measure, 10, 34, **41**
spectrum transformation, 12, **98**, 101, 132, 139
    clip, 99
    diffusion, 100
    flip, 99
    shift, 99
    square, 99
stacked generalization, 69
stacking, *see* stacked generalization
standardized residuals, 109
statistical testing, 171
    analysis of variance, 172
    ANOVA, *see* analysis of variance
    assumption of independence, **172**
    assumptions, 172
    Conover post hoc test, 174
    Cramèr-von Mises, 109
    estimate of location, 175
    Friedman test, 174
    heteroscedastic, 172
    homoscedastic, 172
    Kruskal-Wallis rank sum test, 174
    non-parametric, 172
    parametric, 172
    Tukey's HSD test, 172
strings, 87, **156**, 179
support vector machine, **21**, 30, 43
surrogate model-based optimization, **20**, 49
survival selection, 17
symbolic regression, 120, **122**
symmetric, 38, **42**

test function, 51, 106, 143, 168
    cost, 147
    criteria, 145
    difficulty, 145
    diversity, 146
    flexibility, 147
    generator, **145**, 169
    relevance, 144, **147**
    ruggedness, 147
    simulation-based, **145**, 169
    textbf, 144
traveling salesperson problem, 32, **51**
    asymmetric, 51
tree, 120
    arity, 125
    node, 122
triangle inequality, **42**

unimodal problem, **52**, 92

variable
    conditional, 136
    dummy, 32
    importance, 149
    mixed, 32
    selection, 68
variogram, 64

weighted least squares, 57, **176**
weighted tardiness problem, 51
weld sequence optimization, 31