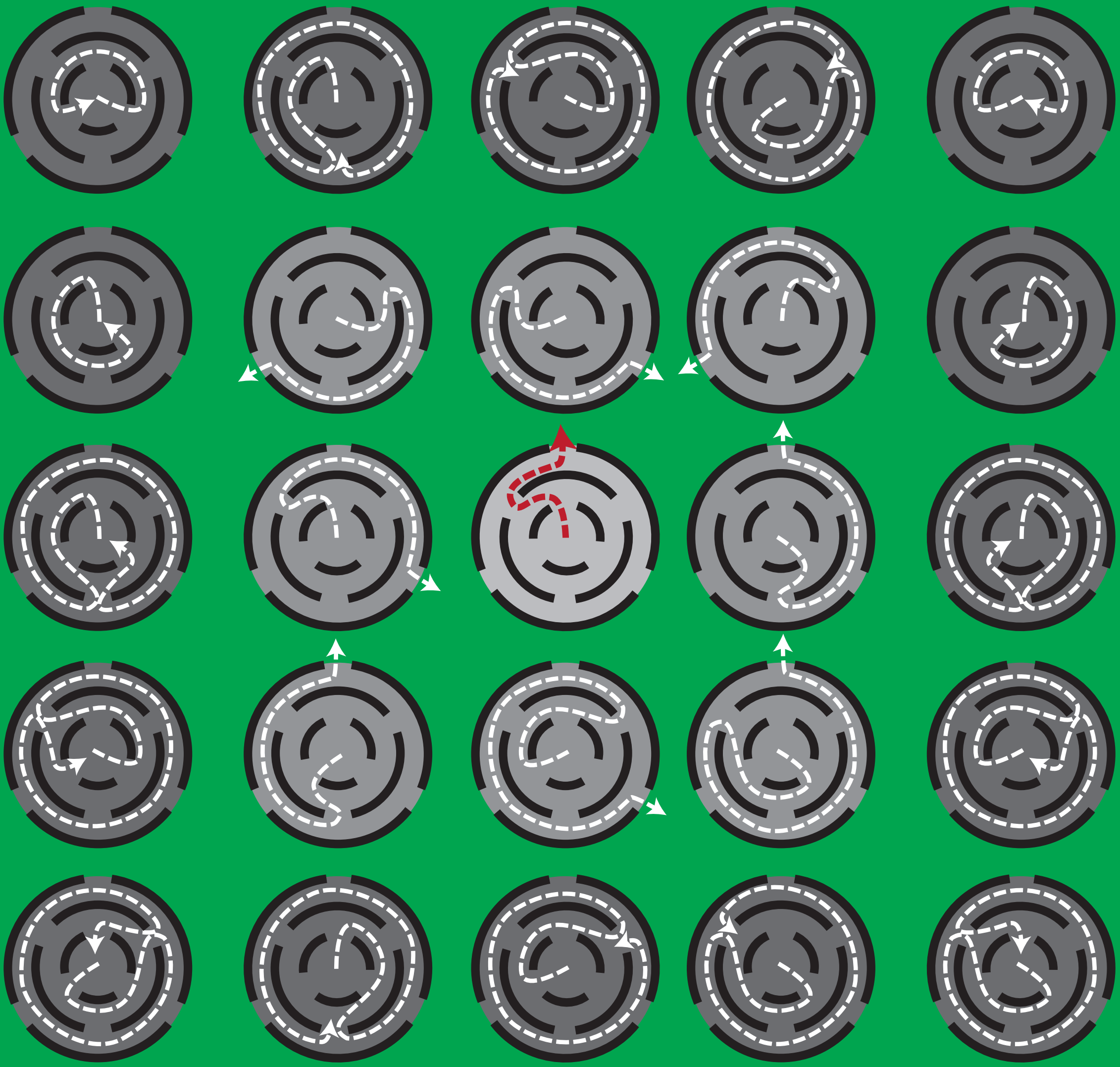
Way of the Fittest

Optimization by Behavioral Evolution



Jörg Willi Stork

Way of the Fittest

Optimization by Behavioral Evolution

Jörg Willi Stork

M.Eng.

Department of Computer Science
Faculty of Sciences, Vrije Universiteit Amsterdam

2021

VRIJE UNIVERSITEIT

Way of the Fittest

Optimization by Behavioral Evolution

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor of Philosophy
aan de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. C.M. van Praag,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Bètawetenschappen
op dinsdag 18 januari 2022 om 9.45 uur
in een bijeenkomst van de universiteit,
De Boelelaan 1105

door

Jörg Willi Stork

geboren te Wipperfürth, Duitsland

promotor:	prof.dr. A.E. Eiben
copromotor:	prof.dr. T. Bartz-Beielstein
promotiecommissie:	prof.dr. T. Bäck
	dr.-habil. C. Doerr
	prof.dr. B. Filipič
	prof.dr. G.M. Koole
	prof.dr. E.-G. Talbi
	dr. M. Wagner

Contents

Acknowledgements

I dedicate this work to my parents Dagmar and Willi Stork. They have always supported me unconditionally in so many different ways.

My sincere gratitude goes to my beloved fiancée Joana. She has always been extremely patient with me, even when I was required to put work in front of my private life.

First of all, I would like to thank Thomas Bartz-Beielstein for his constant support and guidance throughout my academic career. He sparked my interest in optimization, evolutionary algorithms, and research. Furthermore, he allowed me to prove myself in the academic environment. I am incredibly grateful to him for that.

Secondly, I would like to thank Gusz Eiben for his experience and assistance in implementing new research ideas. Our fruitful discussions have always opened up new perspectives and have been highly beneficial for my research. His positive energy has always inspired me.

Furthermore, I would like to thank many amazing colleagues and friends. This includes Martin Zaefferer, without whose experience and expertise I would never have come this far. A special thank you goes to Andreas Fischbach, Frederik Rehbach, and Margarita Rebolledo. Our numerous discussions about research and life have constantly enriched me and made academic life more fun.

My great gratitude goes to Alexander Hagg, who has opened my eyes to entirely new perspectives. I would also like to thank Boris Naujoks, who supported me with his expertise, openness, and empathy. Not to forget all my other colleagues, I really enjoyed working with you!

The publications presented in this thesis were partly supported by the German Federal Ministry of Education and Research under the grant numbers 13FH007IB6 and 17002X1, and by the European Union Horizon 2020 research and innovation program under grant agreement no. 692286. Further, I received funding by the Federal Ministry of Education and Research under the grant numbers 13FH007IB6 and 13N15173.

1 Introduction

Preamble. *Every human being has different behavioral traits. They define how we act and interact with our surroundings, such as whether we seek friendliness or antagonism in our interactions, whether we act according to common sense or ignorance, and many more. In total, they build a definition of behavior and express our impact in life, as a citizen, as a human. Our behaviors have led us to succeed in natural selection and evolved modern humankind.*

This notion of behavior can be extended to human-designed and acting entities, such as software agents or robots embedded in a nature-inspired evolutionary optimization process, and describes their way of acting in environments.

In which ways can we shape the behavior of such entities to advance their way of acting, their task performance, and ultimately, their ability to succeed?

1.1 Motivation

The use of evolutionary computation (EC) methods has demonstrated indisputable success in many domains of optimization, where they belong to the state-of-the-art for solving problems with complex characteristics [Bäck et al., 1997; Eiben and Smith, 2015a,b]. EC is inspired by Darwin's theory of the survival of the fittest. In competitive evolutionary processes, individuals who are best adapted to their environment and have the best task performance succeed. Their employment in applications in artificial intelligence thus seems only natural and has been approached in different ways.

One outstanding paradigm is neuroevolution, a method to generate artificial neural networks (ANN) and optimize their weights, parameters, topologies, and functions [Stanley et al., 2019; Miller and Thomson, 2000; Turner and Miller, 2013]. The evolutionary optimization in neuroevolution is based on fitness selection with a customizable, i.e., user-selected, notion of an ANN's quantitative performance. In contrast, standard supervised learning methods employ gradients based upon back-propagation that require a clear notion of correct input-output samples. Neuroevolution is favorable if gradient information is challenging to compute, sparse, vague, or even unavailable, such as in evolutionary robotics or reinforcement learning [Bongard, 2013]. However, the resulting neuroevolutionary optimization problems are challenging in their nature due to complex and costly environments, multi-modal fitness landscapes, and the number of degrees of freedom in generating ANNs. Under these circumstances, the steering of the evolutionary process in the direction of successful candidate ANNs is challenging. Frequently, the process requires significant amounts of evaluations to discover high-performing candidates.

In this context, surrogate model-based optimization (SMBO) can be employed to improve the sample efficiency by partially replacing fitness evaluations [Forrester et al., 2008]. The surrogate approximates the complex fitness landscape and predicts high-performing candidate solutions with the help of distance metrics that determine the correlation of a candidate solution's fitness to that of similar individuals. The employment of such distance metrics in neuroevolution is challenging as the correlation of fitness on a genotypic level might be subject to non-linearities. A possible solution is to consider the correlation of an individual's fitness to their expressed phenotype and their derived behavior in an environment [Doncieux and Mouret, 2010; Moraglio et al., 2012; Hildebrandt and Branke, 2015; Gaier et al., 2018].

This thesis presents results from cooperative research carried out at the VU Amsterdam and TH Köln. This research focuses on optimizing complex structures through EC and SMBO, employing phenotypic and behavioral distance measures. The central concept of this thesis is the behavioral optimization of individuals by the identification and utilization of the intense connection between an individual's behavior, environment, and fitness. The exploitation of an entity's behavior can significantly support and improve an evolutionary optimization process.

1.2 Behavioral Evolution

In genetics, an organism's genotype is the hereditary information, whereas the phenotype is the set of all observable characteristics, including morphology, biochemical and physiological properties. This notion was widened by Dawkins [1982] to the *extended phenotype*, which puts the behavior in the center of the definition and describes how genes ultimately influence their surroundings in the exerted behavior. Phenotypic variation emerges from the genotype with a strong influence from an environment, i.e., the definition of a phenotype requires the context of an environment. Evolution is then driven by the fitness of individual genes, expressed by their comprehensive effects on their environment.

In the context of computer science, a differentiation between applications with passive entities and active entities with agency embedded in space and time exists [Eiben and Smith, 2015b]. For example, in a classic traveling salesperson problem, a solution's phenotype is depicted by the route connecting several cities. This phenotype can be directly evaluated and awarded with a fitness, resulting in a passive, non-acting entity without any notion of time. In contrast, a phenotype from evolutionary robotics (ER) consists of the robot's physical body and its controller that cannot be directly evaluated. Instead, its behavior in an environment has to be observed to assess its fitness. Eiben and Smith [2015b] conclude that *"in EC we have a 3-step chain, genotype–phenotype–fitness, while in ER the chain is 4-fold, genotype–phenotype–behavior–fitness"*. In their 4-fold chain definition, behavior is a unique property of an entity, supplemental to the phenotype, and the fitness is the quantification of the quality of this behavior.

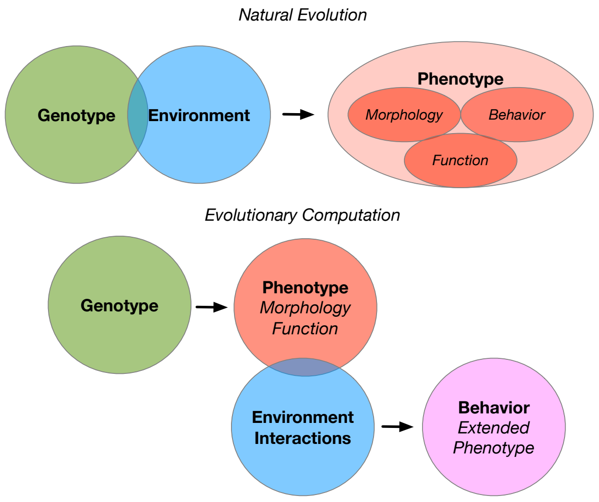


Figure 1.1: *Relationship of genotype, phenotype, and behavior in genetics and evolutionary computation.*

A concept that combines the perspective of EC and genetics is presented by Hagg [2021], which bases upon the principle of the *extended phenotype* by Dawkins: the behavior is seen as an extension of the phenotype, which describes how it interacts with its environment. However, the influence of the environment differs from its role in genetics, as visualized in Figure 1.1. In EC, the genotype and related phenotypes are typically designed for a specific environment. The environment is the application scope of an individual and does not influence the transition to the phenotypic expression, which remains fixed. The behavior of a phenotype can only be observed in an interaction with the environment. Without these task-dependent interactions, behavior is not observable.

For example, the genes of a particular individual express the specific topology and weights of an ANN for the task of controlling a specific robot. The genotype is designed such that the phenotype has the required number of sensor inputs and actor outputs for this robot. The controller's behavior can only be measured related to its task, e.g., maze solving, whereas the fitness is a measurement of this behavior, e.g., how fast it moves to a target position. Considering fitness optimization, the performance of a robot is measured based on its behavior. This leads to the idea of *behavioral optimization*: If a method optimizes behavior, it consequently optimizes fitness. Optimizing behavior instead of optimizing the genotype might be a more appropriate methodology for robotics, as the behavioral space is much "closer" to fitness.

Following the definitions of the *4-fold chain* and *extended phenotype*, a behavior is ultimately influenced by changes in the genotype, which is also the typical target for any variation operators, such as mutation or crossover in EC. However, conducting variation operations that drive the behavior in the desired direction of optimal fitness remains challenging. The genotype-to-phenotype mapping does not ensure a one-to-one relation because identical phenotypes can be represented by multiple genotypes. This can be caused by gene neutrality: specific gene mutations do not affect the phenotype, or competing conventions, i.e., the existence of several ways to represent a functionally equal phenotype. Moreover, a specific phenotype's behavior is equal for identical environment interactions, but equal behaviors can originate from entirely different entities, e.g., different models or concepts. Ultimately, the purposeful adaption of genotypes is challenging, as the complex transitions in the *4-fold chain* do not imply direct relationships between genotype and fitness variation.

This issue led to the idea of moving to the end of the chain, directly searching in the space of behaviors. To realize this search, methods are required to operate directly in this space. To enable efficient search methods like SMBO, it is essential to be able to predict behaviors and their fitness, forming the central concept of approximating the behavior-fitness correlation. The envisioned behavioral optimization methods led to a set of requirements:

(I) Firstly, the desired methods are based on EC and SMBO methodologies. Thus, it is required to acquire an in-depth understanding of these algorithms and their concepts, particularities, and different available implementations.
(II) Secondly, in large parts of this thesis, Kriging models are employed for the task of SMBO as a flexible predictor. Kriging is a data-driven, distance-based model based upon Gaussian processes. However, the application of Kriging was previously limited to continuous optimization, while other problem domains were less investigated. An essential requirement is the advancement of Kriging to support custom distance measures that can be non-continuous, i.e., combinatorial, genotypic, or behavioral measures.
(III) Thirdly, it is required to design, test, and analyze new methods operating in the phenotypic space based upon the principles of EC and SMBO. A particular focus lies on establishing and comparing genotypic and phenotypic distance measures applicable for modeling topology-changing ANN.
(IV) Fourthly, the developed mechanism needs to be extended in its applicability to time-dependent reinforcement learning. It requires a task-dependent and precise notion of an individual's behavior with an in-depth analysis of ways to compare, control, and steer these behaviors. Finally, it is required to design efficient behavioral optimization algorithms and prove their performance.

This thesis presents several works, which are in their entirety a fundamental step towards fulling these requirements and establishing behavioral optimization. The main contributions of the research in this thesis are:
(1) An extensive introduction to EC and SMBO in the greater context of global optimization, including a new taxonomy for algorithm classification and recommendations for practitioners.
(2) Demonstrating the sample efficiency gain of applying SMBO to the task of neural network weight optimization in an applied study, including a comparison with common evolutionary algorithms.
(3) The definition of a custom Kriging kernel and a comparison of distance metrics for combinatorial search spaces, illustrating their applicability and performance in SMBO for permutation problems, including their capacity to increase the sampling efficiency of the optimization.

(4) The introduction of a phenotypic distance metric for the task of tree-based genetic programming models, fixed-topology ANN, as well as graph-based, topology-changing ANN in neuroevolution. Moreover, a rigorous comparison against genotypic distance metrics in different empirical studies was conducted, including investigating their performance for modeling and SMBO processes.
(5) Development of behavioral distance measures in the context of reinforcement learning, including their implementation in SMBO with empirical studies of their performance. Further, an investigation of influences by task and environment and the implications of different ways to generate behavior.
(6) Development of a behavioral optimization framework for reinforcement learning, utilizing a diverse set of optimization and training methods exploiting behavior. It is based on topology-changing control policies in a gradient-free neuroevolutionary algorithm.

1.3 Scope

This thesis presents a collection of scientific publications, with each being presented in a designated chapter. The work's content is primarily equivalent to their published versions, except for necessary layout changes, harmonizing expressions, and language and grammar corrections.

As an exception, Chapter 4 presents a summary of papers [I] and [II], focussing on the most relevant content for the scope of this thesis. If any noteworthy changes were applied (e.g., the renaming of a specific expression), they are marked as footnotes in the text.

Figure 1.2 illustrates the path starting from classic *evolutionary and surrogate-based optimization*, to the realization of suitable *genotypic and phenotypic distance measures* for complex combinatorial spaces, to the successful application of *behavioral optimization* in the domain of reinforcement learning.

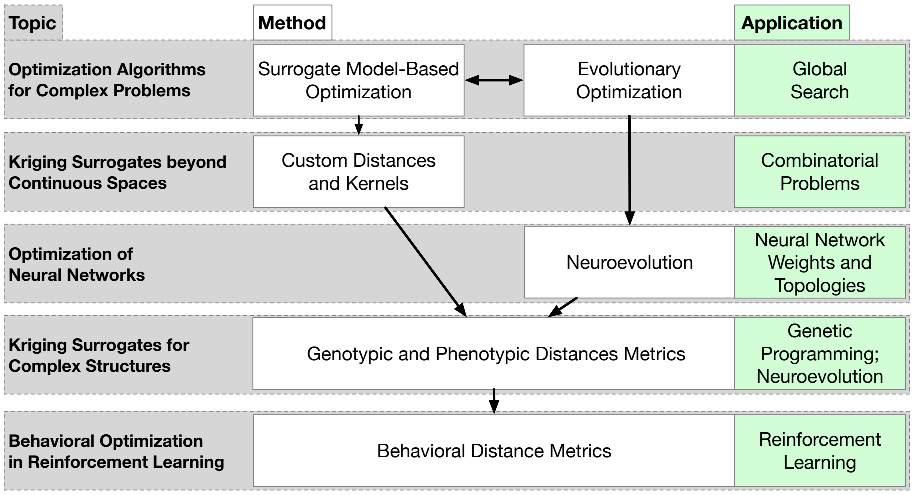

Topic
Method
Application
Optimization Algorithms for Complex Problems
Surrogate Model-Based Optimization
Evolutionary Optimization
Global Search
Kriging Surrogates beyond Continuous Spaces
Custom Distances and Kernels
Combinatorial Problems
Optimization of Neural Networks
Neuroevolution
Neural Network Weights and Topologies
Kriging Surrogates for Complex Structures
Genotypic and Phenotypic Distances Metrics
Genetic Programming; Neuroevolution
Behavioral Optimization in Reinforcement Learning
Behavioral Distance Metrics
Reinforcement Learning

Figure 1.2: *Overview of the scope of this thesis.*

Evolutionary and Surrogate Model-Based Optimization

The concepts of EC and SMBO form the basis for all research papers presented in this thesis. Article [IX] explains the algorithms, embedded in an overview of optimization methods, and illustrates their design and search behavior compared to other available local and global search strategies. A particular focus of the research lies in their abilities to solve complex and costly problems in the domain of real-world problems. Paper [III] outlines an applied study comparing the performance of evolutionary and SMBO algorithms to optimize neural networks. The study employs a conventional version of *neuroevolution*, which optimizes ANN connection weights.

Kriging Surrogates beyond Continuous Spaces

Chapter 4 explains Kriging models and their adaptation from continuous to custom search spaces and analyzes combinatorial distances. Custom distance measures are employed in all remaining papers and serve as a basis for the research of *genotypic*, *phenotypic*, and *behavioral* surrogate models.

Neuroevolution

Most research papers in this thesis employ a version of neuroevolution, either in the shape of conventional weight optimization or extended to neural network topology evolution. While conventional neuroevolution is addressed in paper [III], the remaining research focuses on optimizing changing topologies. For this task, cartesian genetic programming for artificial neural networks (CGP-ANN) is employed based on graph-based ANN structures derived from genotypic representations organized and arranged in cartesian coordinate systems with a fixed size [Miller and Thomson, 2000; Turner and Miller, 2013]. These fixed-size representations allow direct comparisons if employed in a *genotypic distance.*

Genotypic and Phenotypic Distances

The definition of suitable distance metrics for modeling and optimization is a challenging task. A central pillar in this thesis is the identification, implementation, benchmarking, and comparison of genotypic and phenotypic distance measures in model-based optimization processes. The difficulty of the interactions and transitions from an individual's genotype to their phenotypes and behavior, which ultimately defines their fitness in an environment, are examined and discussed for different domains. The applications range from genetic programming [IV], over the performance analysis of modeling neural networks [VII], to the surrogate-based optimization of topology-changing neural networks [V]. The gathered insights ultimately lead to the concept of environment-specific *behavioral distances.*

Behavioral Optimization

The final part of this thesis is dedicated to researching distances and models in the behavioral space based on reinforcement learning agents. In this context, paper [VI] defines *behavior* as the reactions of ANN to the consecutive environment states embedded in time. A central research question is how a mutual behavioral space can be defined in this context of different trajectories. Paper [VIII] advances this investigating by comparing behavior for specific state sets, including their importance and impacts on the distance metric and modeling quality. Finally, paper [X] integrates the prior insights to create an algorithm for behavioral optimization of topology-changing neural networks. The algorithm employs different definitions of behavioral metrics in a hybrid optimization process, including significant influences by value-based methods from standard reinforcement learning.

List of Papers

This thesis is based on ten papers published between 2014 and 2021. The papers in this thesis are listed below, along with details of the contribution to the publication in question.

Part	**2014**	**2017**	**2018**	**2019**	**2020**	**2021**
Optimization of Complex Problems		[III]			[IX]	
Combinatorial Distance Metrics	[I] [II]					
Neuroevolution		[III]		[V,VI]	[VIII]	[X]
Genotypic and Phenotypic Metrics			[IV]	[V,VII]		
Behavioral Optimization				[VI]	[VIII]	[X]

[I] Zaefferer M, Stork J, Friese M, Fischbach A, Naujoks B, & Bartz-Beielstein T (2014, July). Efficient global optimization for combinatorial problems. *In Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 871-878). ACM, New York.

[II] Zaefferer M, Stork J, & Bartz-Beielstein T (2014, September). Distance measures for permutations in combinatorial efficient global optimization. *In International Conference on Parallel Problem Solving from Nature* (pp. 373-383). Springer, Cham.

Papers [I] and [II] were crafted in the early stage of my Ph.D. trajectory and present the concepts for employing surrogates in combinatorial search spaces, a significant milestone for my subsequent research. I supported the crafting, researching, and discussing of the novel idea behind the paper and assisted in analyzing the experiments for these papers.

[III] Stork J, Zaefferer M, Fischbach A, Rehbach F, & Bartz-Beielstein T (2017) Surrogate-assisted learning of neural networks. *In Proceedings of the 27. Workshop Computational Intelligence* (pp. 195-210), KIT Scientific Publishing.

Paper [III] presents applied research and compares evolutionary and surrogate-based optimization for solving an elevator group control problem. I designed the work, conducted and analyzed the experiments, and wrote most of the text.

[IV] Zaefferer M, Stork J, Flasch O, & Bartz-Beielstein T (2018, September). Linear combination of distance measures for surrogate models in genetic programming. *In International Conference on Parallel Problem Solving from Nature* (pp. 220-231). Springer, Cham.

Together with the first author, I developed the idea behind paper [IV] and the experimental design. I elaborated on the idea of using a phenotypic distance measure for the comparison of complex structures. I further supported conducting and analyzing the experiments and wrote parts of the text.

[V] Stork J, Zaefferer M, & Bartz-Beielstein T (2019, April). Improving neuroevolution efficiency by surrogate model-based optimization with phenotypic distance kernels. *In International Conference on the Applications of Evolutionary Computation (Part of EvoStar)* (pp. 504-519). Springer, Cham.
Nominated Outstanding Student Award

The paper [V] compares different genotypic and phenotypic distances for modeling Kriging surrogates of ANNs with changing topologies. I designed the idea, implemented the algorithm, conducted and analyzed the experiments, and wrote most of the text.

[VI] Stork J, Zaefferer M, Bartz-Beielstein T, & Eiben AE (2019, July). Surrogate models for enhancing the efficiency of neuroevolution in

reinforcement learning. *In Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 934-942). ACM, New York.
Nominated Best Paper Award

The paper [VI] explores the idea of applying behavior-based surrogates for neuroevolution for the task of reinforcement learning. I designed the idea, implemented the algorithm, conducted and analyzed the experiments, and wrote most of the text.

[VII] Hagg A, Zaefferer M, Stork J, & Gaier A (2019, July). Prediction of neural network performance by phenotypic modeling. *In Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1576-1582). ACM, New York.

The paper [VII] intensively discusses the performance of genotypic and phenotypic distance metrics for modeling complex search spaces. All authors contributed equally to the idea behind the paper, the planning of the experiments, and the discussion of the results. My unique contribution encompasses the use of phenotypic distances and parts of the writing.

[VIII] Stork J, Zaefferer M, Bartz-Beielstein T, & Eiben AE (2020). Understanding the behavior of reinforcement learning agents. *In Bioinspired Optimization Methods and Their Applications: 9th International Conference*, (pp. 148-160). Springer Nature, Cham.

The paper [VIII] analyzes general behavior, behavioral distance metrics, and behavioral models in reinforcement learning. I designed the idea, conducted and analyzed the experiments, and wrote most of the text.

[IX] Stork J, Eiben, AE, & Bartz-Beielstein, T (2020). A new taxonomy of global optimization algorithms. *Natural Computing*, 1-24. Springer, Cham.

The article [IX] presents an extensive overview and new taxonomy for modern optimization algorithms. I contributed to this article by investigating literature, compiling the information, creating the new taxonomy, figures, overviews, and overall writing.

[X] Stork J, Zaefferer M, Eisler N, Tichelmann P, Bartz-Beielstein T, & Eiben AE (2021). Behavior-based neuroevolutionary training in reinforcement learning. *(to appear) In 2021 Genetic and Evolutionary Computation Conference Companion.* ACM, New York.

The paper [X] combines previous research to employ an algorithm for behavioral optimization in reinforcement learning. I designed and implemented the algorithm, planned, conducted, and analyzed the experiments, and wrote the text.

Further Publications

For the following papers, I have co-authorship and decided not to add their content to the thesis.

Heinerman J, Stork J, Coy MAR, Hubert J, Bartz-Beielstein T, Eiben AE, & Haasdijk E (2017, September). Can social learning increase learning speed, performance or both?. *In Artificial Life Conference Proceedings 14* (pp. 200-207). MIT Press.

The paper illustrates the impact of sharing behavior among robot controllers. I prepared the experimental plan and analyzed the results of the tuning part of the paper. I wrote the tuning part.

Rehbach F, Zaefferer M, Stork J, & Bartz-Beielstein T (2018, July). Comparison of parallel surrogate-assisted optimization approaches. *In Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1348-1355). ACM, New York.

In this paper, different ways to parallel evaluations in the context of surrogates are highlighted. I supported discussing the underlying idea, planning and analyzing the experiments, and writing the paper.

Stork J, Friese M, Zaefferer M, Bartz-Beielstein T, Fischbach A, Breiderhoff B, Naujoks B, & Tušar T (2020). Open issues in surrogate-assisted optimization. *In High-Performance Simulation-Based Optimization* (pp. 225-244). Springer.

This overview book chapter discusses open issues in SMBO. I wrote a part of the text and investigated the literature.

1.4 Frequently Used Acronyms

ANN	Artificial Neural Network
BD	Behavioral Distance
CGP	Cartesian Genetic Programming
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EGO	Efficient Global Optimization
EI	Expected Improvement
ES	Evolution Strategy
GO	Global Optimization
GP	Genetic Programming
MLE	Maximum Likelihood Estimation
NE	NeuroEvolution
PHD	Phenotypic Distance
RL	Reinforcement Learning
RS	Random Search
SMBO	Surrogate Model-Based Optimization
SMB-NE	Surrogate Model-Based NeuroEvolution

2

An Introduction to Evolutionary and Surrogate Model-Based Optimization for Global Search

Chapter 2 was published as:

Stork J, Eiben, AE, & Bartz-Beielstein, T (2020). A new taxonomy of global optimization algorithms. *Natural Computing*, 1-24. Springer, Cham.
DOI: 10.1007/s11047-020-09820-4

Surrogate model-based optimization, nature-inspired metaheuristics, and hybrid combinations have become state-of-the-art in algorithm design for solving real-world optimization problems. Still, it is difficult for practitioners to get an overview that explains their advantages in comparison to a large number of available methods in the scope of optimization. Available taxonomies lack the embedding of current approaches in the larger context of this broad field. This article presents a taxonomy of the field, which explores and matches algorithm strategies by extracting similarities and differences in their search strategies. A particular focus lies on algorithms using surrogates, nature-inspired designs, and those created by automatic algorithm generation. The extracted features of algorithms, their main concepts, and search operators, allow us to create a set of classification indicators to distinguish between a small number of classes. The features allow a deeper understanding of components of the search strategies and further indicate the close connections between the different algorithm designs. We present intuitive analogies to explain the basic principles of the search algorithms, particularly useful for novices in this research field. Furthermore, this taxonomy allows recommendations for the applicability of the corresponding algorithms.

2.1 Introduction

Modern applications in industry, business, and information systems require a tremendous amount of optimization. Global optimization (GO) tackles various severe problems emerging from the context of complex physical systems, business processes, and particularly from applications of artificial intelligence. Challenging problems arise from industry on the application level, e.g., regarding manufacturing speed, part quality, or energy efficiency, or on the business level, such as optimizing production plans, purchase, sales, and after-sales. Further, they emerge from artificial intelligence and information engineering, for example, the optimization of machine learning models such as neural networks for different applications. Their complex nature connects all these problems: they tend to be expensive to solve and with unknown objective function properties, as the underlying mechanisms are often not well described or unknown.

Solving optimization problems of this kind relies necessarily on performing costly computations, such as simulations, or even real-world experiments, which are frequently considered being black-box. In this context, the commonly high costs of function evaluations are a considerable challenge. Whether we are probing a real physical system, querying the simulator, or creating a new complex data model, many resources are needed to fulfill these tasks. GO methods for such problems thus need to fulfill a particular set of requirements. They need to work with black-box-style probes only, so without any further information on the structure of the problem. Further, they must find the best possible improvement within a limited number of function evaluations.

The improvement of computational power in the last decades has been influencing the development of algorithms. A massive amount of computational power became available for researchers worldwide through multi-core desktop machines, parallel computing, and high-performance computing clusters. This has contributed to the following fields of research: Firstly, the development of more complex, nature-inspired, and generally applicable heuristics, so-called metaheuristics. Secondly, it facilitated significant progress in the field of accurate, data-driven approximation models, so-called surrogate models, and their embodiment in an optimization process. Thirdly, the upcoming of approaches that combine several optimization algorithms and seek towards automatic combination and configuration of the optimal optimization algorithm, known as hyperheuristics. The hyperheuristic approach shows the close connections between different named algorithms, in particular in the area of bio-inspired metaheuristics. All these GO algorithms differ broadly from standard approaches, define new classes of algorithms, and are not well integrated into available taxonomies.

Hence, we propose a new taxonomy, which:

I describes a comprehensive overview of GO algorithms, including surrogate-based, model-based, and hybrid algorithms,

II can generalize well and connects GO algorithm classes to show their similarities and differences,

III focusses on simplicity, which enables an easy understanding of GO algorithms,

IV can be used to provide underlying knowledge and best practices to select a suitable algorithm for a new optimization problem.

Our new taxonomy is created based on algorithm key features and divides the algorithms into a small number of intuitive classes: *Hill-Climbing, Trajectory, Population, Surrogate, and Hybrid.* Further, *Exact* algorithms are shortly reviewed but not an active part of our taxonomy, which focuses on heuristic algorithms. We further utilize extended class names as descriptions founded on the abstracted human behavior in pathfinding. The analogies *Mountaineer, Sightseer, Team, Surveyor* create further understanding by using the image of a single or several persons hiking a landscape in search of the desired location utilizing the shortest path. This utilized abstraction allows us to present comprehensible ideas on how the individual classes differ and, moreover, how the respective algorithms perform their search.

This chapter mainly addresses different kinds of readers: Beginners will find an intuitive and comprehensive overview of GO algorithms, especially concerning common metaheuristics and developments in the field of surrogate-based and hybrid and hyperheuristic optimization. For advanced readers, we also discuss the applicability of the algorithms to tackle specific problem properties and provide essential knowledge for reasonable algorithm selection. We provide an extensive list of references for experienced users. The taxonomy can be used to create realistic comparisons and benchmarks for the different classes of algorithms. It further provides insights for practitioners who aim to develop new search strategies, operators, and algorithms.

In general, most GO algorithms were developed for a specific search domain, e.g., discrete or continuous. However, many algorithms and their fundamental search principles can be successful for different problem spaces with reasonably small changes. For example, evolution strategies (ES), which are popular in continuous optimization (see [Hansen et al., 2003]), have their origin in the discrete problem domain. Beyer and Schwefel [2002] describe how the ES moved from discrete to continuous decision variables. Based on this consideration, the article is focused but not limited to illustrating the algorithm variants for the continuous domain while they have their origins or are most successful in the discrete domain.

Moreover, we focused this taxonomy on algorithms for objective functions without particular characteristics, such as multi-objective, constrained, noisy, or dynamic. These function characteristics pose additional challenges to any algorithm, which are often faced by enhancing available search schemes with specialized operators or even completely dedicated algorithms. We included the former as part of the objective function evaluation in our general algorithm scheme and provide references to selected overviews. Dedicated algorithms, e.g., for multi-objective search, are not discussed in detail. However, their accommodation in the presented taxonomy is possible if their search schemes are related to the algorithms described in our taxonomy. If further required, we outlined the exclusive applicability of algorithms and search operators to specific domains or problem characteristics.

We organized the remainder of this chapter as follows: Section 2.2 presents the development of optimization algorithms and their core concepts. Section 2.3 motivates a new taxonomy by reviewing the history of available GO taxonomies, illustrates algorithm design aspects, and presents extracted classification features. Section 2.4 introduces the new intuitive classification with examples. Section 2.5 introduces best practices suggestions regarding the applicability of algorithms. Section 2.6 summarizes and concludes the article with the recent trends and challenges in GO and currently essential research fields.

2.2 Modern Optimization Algorithms

This section describes the fundamental principles of modern search algorithms, particularly the elements and backgrounds of surrogate-based and hybrid optimization. Global optimization aims to find the overall best solution, i.e., for the common task of minimization, to discover decision variable values that minimize the objective function value.

We denote the global search space as compact set $\mathcal{S} = \{\mathbf{x} \mid \mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u\}$ with $\mathbf{x}_l, \mathbf{x}_u \in \mathbb{R}^n$ being the explicit, finite lower and upper bounds on $\mathbf{x}$. Given a single-objective function f: $\mathbb{R}^n \to \mathbb{R}$ with real-valued input vectors $\mathbf{x}$ we attempt to find the location $\mathbf{x}^* \in \mathbb{R}^n$ which minimizes the function: $\arg\min f(\mathbf{x}), \mathbf{x} \in \mathcal{S}$.

Finding a global optimum is always the ultimate goal and desirable. However, for many practical problems, discovering a solution that improves the current best solution in a given budget of evaluations or time will be considered a success. Notably, in continuous GO domains, an optimum commonly cannot be identified exactly; thus, modern heuristics are designed to spend their resources as efficiently as possible to find the best possible improvement in the objective function value while finding a global optimum is never guaranteed.

Törn and Zilinskas [1989] mention three principles for the construction of an optimization algorithm:

1. An algorithm utilizing all available *a priori* information will outperform a method using less information.
2. If no *a priori* information is available, the information is completely based on evaluated candidate points and their objective values.
3. Given a fixed number of evaluated points, optimization algorithms will only differ from each other in the distribution of candidate points.

If *a priori* information about the function is accessible, it can significantly support the search and should be considered during the algorithm design. Current research on algorithm designs that include structural operators, such as function decomposition, is known as *grey-box optimization* [Whitley et al., 2016; Santana, 2017]. However, many modern algorithms focus on handling black-box problems where the problem includes little or no *a priori* information.

The principles displayed above lead to the conclusion that the most crucial design aspect of any black-box algorithm is finding a strategy to distribute the initial candidates in the search space and generate new candidates based on a variation of solutions. These procedures define the *search strategy*, which needs to follow the two competing goals of *exploration* and *exploitation.* The balance between these two competing goals is usually part of the algorithm configuration. Any algorithm needs to be adapted to the structure of the problem at hand to achieve optimal performance. This can be considered during the construction of an algorithm, before the optimization by parameter *tuning* or during the run by parameter *control* [Bartz-Beielstein et al., 2005a; Eiben et al., 1999]. In general, the main goal of any method is to reach their target with high efficiency, i.e., to discover optima fast and accurately with as few resources as possible. Moreover, for many demanding and expensive tasks the goal is to identify a valuable solution or improvement, instead of the global optimum. We will discuss the design of modern optimization algorithms in Section 2.3.2.

2.2.1 Exact Algorithms

Exact algorithms, also referred to as *non-heuristic* or *complete* algorithms [Neumaier, 2004], are a special class of *deterministic, systematic* or *exhaustive* optimization techniques. They can be applied in discrete or combinatorial domains, where the search space has a finite number of possible solutions or for continuous domains, if an optimum is searched within some given tolerances. Exact algorithms have a guarantee to find a global optimum by using a predictable amount of resources, such as function evaluations or computation time [Neumaier, 2004; Fomin and Kaski, 2013; Woeginger, 2003]. This guarantee often requires sufficient *a priori* information about the objective function, e.g., the best possible objective function value. Without available *a priori* information, the stopping criterion needs to be defined by a heuristic approach, which softens the guarantee of solving to optimality. Well-known exact algorithms are based on the *branching* principle, i.e., splitting a known problem into smaller sub-problems, which each can be solved to optimality. The

Branch-and-bound algorithm is an example for exact algorithms [Lawler and Wood, 1966].

2.2.2 Heuristics and Metaheuristics

In modern computer-aided optimization, heuristics and metaheuristics are established solution techniques. Although presenting solutions that are not guaranteed to be optimal, their general applicability and ability to present fast sufficient solutions make them very attractive for applied optimization. Their inventors built them upon the principle of systematic search, where solution candidates are evaluated and rewarded with a *fitness*. The term *fitness* has its origins in EC, where fitness describes an individual's competitive ability in the reproduction process. The fitness is in its purest form the objective function value $y = f(\mathbf{x})$ concerning the optimization goal, e.g., in a minimization problem, smaller values have higher fitness than larger values. Moreover, it can be part of the search strategy, e.g., scaled or adjusted by additional functions, particularly for multi-objective or constrained optimization.

Heuristics can be defined as problem-dependent algorithms, which are developed or adapted to the particularities of a specific optimization problem or problem instance [Pearl, 1985]. Typically, heuristics perform systematic evaluations, although utilizing stochastic elements. Heuristics use stochastic principles to provide fast, not necessarily exact (i.e., not optimal) numerical solutions to optimization problems.

Metaheuristics can be defined as problem independent, general-purpose optimization algorithms. They apply to a wide range of problems and problem instances. The term *meta* describes the higher-level method utilized to guide the underlying heuristic strategy [Talbi, 2009].

Metaheuristics share the following characteristics [Boussaïd et al., 2013]:

- The algorithms are nature-inspired; they follow certain natural principles or behaviors (e.g., biological evolution, physics, social behavior).
- The search process involves stochastic parts; it utilizes probability distributions and random processes.

– As they are meant to be generally applicable solvers, they include a set of control parameters to adjust the search strategy.
– They do not rely on the information of the process which is available before the start of the optimization run, so-called *a priori* information. Still, they can benefit from such information (e.g., to set up control parameters)

During the remainder of this article, we will focus on heuristic, respectively, metaheuristic algorithms.

2.2.3 Surrogate Model-Based Optimization Algorithms [1]

Surrogate model-based optimization algorithms are designed to process expensive and complex problems, which arise from real-world applications and sophisticated computational models. These problems are commonly black-box, which means that they only provide very sparse domain knowledge. Consequently, problem information needs to be exploited by experiments or function evaluations. SMBO is intended to model available, i.e., evaluated information about candidate solutions to utilize it to the full extent. A surrogate model is an approximation that substitutes the original expensive objective function, real-world process, physical simulation, or computational process during the optimization. In general, surrogates are either simplified *physical* or numerical models based on knowledge about the physical system, or empirical *functional* models based on knowledge acquired from evaluations and sparse sampling of the parameter space [Søndergaard, 2003]. In this work, we focus on the latter, so-called data-driven models. The terms *surrogate model*, *meta-model*, *response surface model* and also *posterior distribution* are used synonymously in the common literature [Mockus, 1974; Jones, 2001; Bartz-Beielstein and Zaefferer, 2017]. Furthermore, we assume that it is crucial to distinguish between the use of an explicit surrogate of the objective function and general *model-based* optimization [Zlochin et al., 2004], which additionally refers to methods where a statistical model is used to generate new candidate solutions (cf. Section 2.3.2). We thus distinguish between the two different terms *surrogate model* and *model*

[1] Instead of "surrogate-based" (original article), the term "surrogate model-based optimization (SMBO)" is used in the thesis.

to avoid confusion and will briefly refer to a *surrogate model* as a *surrogate*. Another term present in the literature is *surrogate-assisted* optimization, which mostly refers to the application of surrogates in combination with population-based EC [Jin, 2011]. Important publications featuring overviews or surveys on surrogates and SMBO were presented by Sacks et al. [1989], Jones [2001], Queipo et al. [2005], Forrester and Keane [2009]. SMBO is commonly defined for but not limited to the case of complex real-world optimization applications. We define a typical SMBO process by three layers, where the first two are considered as *problem* layers, while the latter one is the surrogate, i.e., an approximation of the problem layers. We could transfer the defined layers to different computational problems with expensive function evaluations, such as complex algorithms or machine learning tasks. Each layer can be the target of optimization or used to retrieve information to guide the optimization process. Figure 2.1 illustrates the different layers of objective functions and the SMBO process for real-world problems. The objective function layers, from the bottom up, are:

L1 The real-world application $f_1(\mathbf{x})$, given by the physical process itself or a physical model. Direct optimization is often expensive or even impossible due to evaluations involving resource-demanding prototype building or even dangerous experiments.

L2 The computational model $f_2(\mathbf{x})$, given by a simulation of the physical process or a complex computational model, e.g., a computational fluid dynamics model or structural dynamics model. A single computation may take minutes, hours, or even weeks to compute.

L3 The surrogate $s(\mathbf{x})$, given by a data-driven regression model. The accuracy heavily depends on the underlying surrogate type and amount of available information (i.e., function evaluations). The optimization is, compared to the other layers, typically cheap. Surrogates are constructed either for the real-world application $f_1(\mathbf{x})$ or the computational model $f_2(\mathbf{x})$.

Furthermore, the SMBO cycle includes the optimization process itself, which is given by an adequate optimization algorithm for the selected objective function layer. No SMBO is performed, if the optimization is directly applied to $f_1(\mathbf{x})$

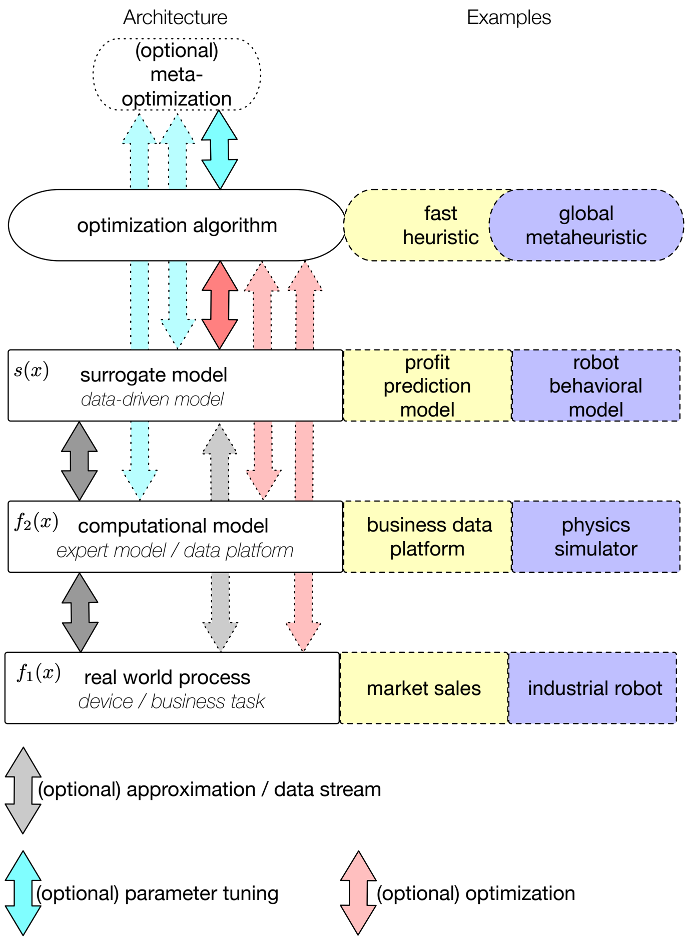


Figure 2.1: *A SMBO process with the different objective function layers: real-world process, computational model, and surrogate. The arrows mark different possible data/application streams. Dotted arrows are in the background, i.e., they pass through elements; each connection always terminates with an arrow. Typically, surrogates are models either for the simulation or real-world function. Direct optimization of the problem layers is also possible. Two examples of processes are given to outline the use of the architecture in a business data and robot control task.*

or $f_2(\mathbf{x})$. The SMBO uses $f_1(\mathbf{x})$ or $f_2(\mathbf{x})$ for verification of promising solution candidates. Moreover, the control parameters of the optimization process and the SMBO cycle, including the surrogate modeling process, can be tuned.

Each layer imposes different evaluation costs and fidelities: the real-world problem is the most expensive to evaluate but has the highest fidelity, while the surrogate is the cheapest to evaluate but has a lower fidelity. The main benefit of using surrogates is thus the reduction of needed expensive function evaluations on the objective function $f_1(\mathbf{x})$ or $f_2(\mathbf{x})$ during the optimization. The studies by Loshchilov et al. [2012], Marsden et al. [2004], Ong et al. [2005] and Won and Ray [2004] feature benchmark comparisons of SMBO. Nevertheless, the model construction and updating of the surrogates also require computational resources, as well as evaluations for verification on the more expensive function layers. An advantage of SMBO is the availability of the surrogate, which can be utilized to gain further global insight into the problem, which is particularly valuable for black-box problems. It can be utilized to identify important decision variables or visualize the nature of the problem, i.e., the fitness landscape.

2.2.4 Meta-Optimization and Hyperheuristics

Meta-optimization or *parameter tuning* [Mercer and Sampson, 1978] describes the process of finding the optimal parameter set for an optimization algorithm. It is also an optimization process itself, which can become very costly in terms of objective function evaluations, as they are required to evaluate the parameter set of a specific algorithm. Hence, particular SMBO algorithms have become very successful meta-optimizer [Bartz-Beielstein et al., 2005a]. Figure 2.1 shows where the meta-optimization is situated in an optimization process. If the algorithm adapts parameters during the active run of optimization, it is called parameter control [Eiben et al., 1999]. Algorithm parameter tuning and control is further discussed in Section 2.3.2.

A *hyperheuristic* [Cowling et al., 2000, 2002] is a high-level approach that selects and combines low-level approaches (i.e., heuristics, elements from meta-heuristics) to solve a specific problem or problem class. It is an optimization algorithm that automates the algorithm design process by searching an ample space of pre-defined algorithm components. A hyperheuristic can also be utilized in an online fashion, e.g., trying to find the most suitable algorithm at each state of a search process [Vermetten et al., 2019]. We regard hyperheuristics as hybrid algorithms (cf. Section 2.4.5).

2.3 A New Taxonomy

The term *taxonomy* is defined as a consistent procedure or classification scheme for separating objects into classes or categories based on specific features. The term taxonomy is mainly present in natural science for establishing hierarchical classifications. A taxonomy fulfills the task of distinction and order; it provides explanations and a greater understanding of the research area by identifying coherence and the differences between the classes.

Several reasons drive our motivation for a new taxonomy: The first reason I) is that considering available GO taxonomies (Section 2.3.1, cf. Figure 2.2), we can conclude that during the last decades, several authors developed new taxonomies for optimization algorithms. However, new classes of algorithms have become state-of-the-art in modern algorithm design, particularly model-based, surrogate-based, and hybrid algorithms dominate the field. Existing taxonomies of GO algorithms do not reflect this situation. Although there are surveys and books which handle the broad field of optimization and give general taxonomies, they are outdated and lack the integration of the new designs. Available up-to-date taxonomies often address a particular subfield of algorithms and discuss them in detail. However, a generalized taxonomy, which includes the above-described approaches and allows to connect these optimization strategies, is missing.

This gap motivated our second reason II) the development of a generalization scheme for algorithms. We argue that the search concepts of many algorithms are built upon each other and are related. While the algorithms have apparent differences in their strategies, they are not overall different. Many examples of similar algorithms can be found in different named nature-inspired metaheuristics that follow the same search concepts. However, certain elements are characteristic of algorithms, which allow us to define classes based on their search elements. Even different classes share a large amount of these search elements. Thus our new taxonomy is based on a generalized scheme of five crucial algorithm design elements (Section 2.3.2, cf. Figure 2.3), which allows us to take a bottom to top approach to differentiate, but also connect the different algorithm classes. The recent developments in hybrid algorithms drive the urge to generalize search strategies, where we no longer use specific, individual algorithms but combinations of search elements and operators of different classes to find and establish new strategies, which cannot merely be categorized.

Our third reason III) is the importance of simplicity. Our new taxonomy is not only intended to divide the algorithms into classes but also to provide an intuitive understanding of the working mechanisms of each algorithm to a broad audience. To support these ideas, we will draw *analogies* between the algorithm classes and the behavior of a human-like individual in each of the descriptive class sections.

Our last reason IV) is that we intend our taxonomy to be helpful in practice. A common issue is the selection of an adequate optimization algorithm if faced with a new problem. Our algorithm classes are connected by individual properties, which allows us to utilize the new taxonomy to propose suitable algorithm classes based on a small set of problem features. These suggestions, in detail discussed in Section 2.5 and illustrated in Figure 2.4 shall help users to find best practices for new problems.

2.3.1 History of Taxonomies

In the literature, one can find several taxonomies trying to shed light on the vast field of optimization algorithms. The identified classes are often named by a significant feature of the algorithms in the class, with the names either being informative or descriptive. For example, Leon [1966] presented one of the first overviews on global optimization. It classified algorithms into three categories: 1. *Blind search*, 2. *Local search* 3. *Non-local search*. *Blind search* refers to simple

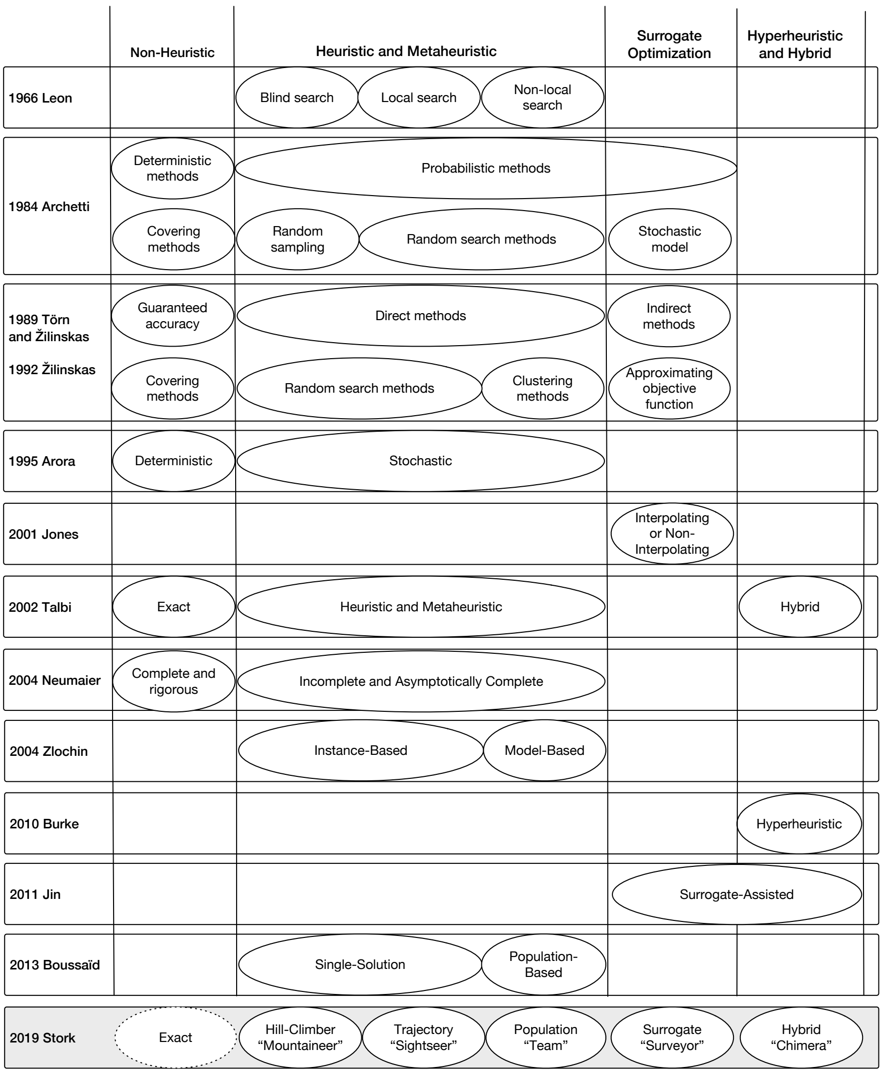


Figure 2.2: *Global Optimization Taxonomy History. Information from Leon [1966], Archetti and Schoen [1984], Törn and Zilinskas [1989],Arora et al. [1995], Jones [2001], Talbi [2002], Neumaier [2004], Zlochin et al. [2004],Burke et al. [2010], Jin [2011] and Boussaïd et al. [2013] are illustrated and compared. Different distinctions between the large set of algorithms were drawn. A comprehensive taxonomy is missing and introduced by our new taxonomy, which concludes the diagram and is further presented in Section 2.3.*

search strategies that select the candidates at random over the entire search space but following a built-in sequential selection strategy. During *local search*, new candidates are selected in the immediate neighborhood of the previous candidates, which leads to a trajectory of small steps. Finally, *non-local search* allows to escape from local optima and thus enables a global search. Archetti and Schoen [1984] extends the above scheme by also adding the class of *deterministic* algorithms, i.e., those who are guaranteed to find the global optimum with a defined budget. The paper stands out in establishing a taxonomy, which for the first time includes the concepts to construct surrogates, as they describe *probabilistic* methods based on statistical models, which are iteratively utilized to perform the search. Törn and Zilinskas [1989] reviewed existing classification schemes and presented their classifications. They made that the most crucial distinction between two non-overlapping main classes, namely those methods with or without guaranteed accuracy. The main new feature of their taxonomy is the clear separation of the heuristic methods in those with *direct* and *indirect* objective function evaluation. Mockus [1974] also discussed the use of Bayesian optimization. Today's high availability of computational power did not exist; therefore, Törn and Zilinskas [1989] concluded the following regarding Bayesian models and their applicability for (surrogate-based) optimization:

> *Even if it is very attractive, theoretically it is too complicated for algorithmic realization. Because of the fairly cumbersome computations involving operations with the inverse of the covariance matrix and complicated auxiliary optimization problems the resort has been to use simplified models.*

Still, we find the scheme of dividing algorithms into non-heuristic (or exact), random (or stochastic), and further surrogate-based frequently. Several following taxonomies added different algorithm features to their taxonomies, such as metaheuristic approaches [Arora et al., 1995], SMBO Jones [2001], non-heuristic methods Neumaier [2004], hybrid methods Talbi [2002], direct search methods [Audet, 2014; Kolda et al., 2003], model-based optimization Zlochin et al. [2004], hyperheuristics Burke et al. [2010], surrogate-assisted algorithms Jin

[2011], nature-inspired methods [Rozenberg et al., 2012], or population-based approaches Boussaïd et al. [2013]. We created an overview of selected taxonomies and put them into the comparison in Figure 2.2.

2.3.2 The Four Elements of Algorithm Design

Any modern optimization algorithm, as defined in Section 2.2, can be reduced to the three key search strategy elements *initialization*, *generation* and *selection*. A fourth element controls all these key elements: the *control* of the different functions and operators in each element. The underlying terminology is generic and based on typical concepts from the field of EC. We could easily exchange it with wording from other standard algorithm classes (e.g., evaluate= test/trial, generate=produce/variate). Algorithm 2.3.1 displays the principal elements and the abstracted fundamental structure of optimization algorithms [Bartz-Beielstein and Zaefferer, 2017]. We could map this structure and elements to any modern optimization algorithm. Even if the search strategy is inherently different or elements do not follow the illustrated order or appear multiple times per iteration. The **initialization** of the search defines starting locations or a schema for the initial candidates. Two common strategies exist:

1. If there is no available a priori knowledge about the problem, the best option is to use well-distributed starting points. The initial distribution target is often exploration, i.e., a broad distribution of the starting points if possible. Particularly interesting for SMBO are systematic initialization schemes by methods from the field of *design of experiments* [Crombecq et al., 2011; Bossek et al., 2020].

2. Suppose domain knowledge or other a priori information is available, such as information from the data or process from previous optimization runs. In that case, it is beneficial to utilize this information, e.g., by using a selection of solutions, such as those with the best fitness. However, known solutions can also bias the search towards them. Thus, e.g., restart strategies intentionally discard them. In SMBO, the initial modeling can use available data.

The initial candidates have a significant impact on the balance between exploration and exploitation. Space-filling designs with large amounts of random candidates or sophisticated *design of experiments* methods will lead to an initial exploration of the search space. Starting with a single candidate will presumably lead to an exploitation of the neighborhood of the selected candidate location. Hence, algorithms using several candidates are in general more robust, while single candidate algorithms are sensitive to the selection of the starting candidate, particularly in multi-modal landscapes. Multi-start strategies can further increase the robustness and are particularly common for single-candidate algorithms and are frequently recommended for population-based algorithms [Hansen et al., 2010b].

Algorithm 2.3.1: General Optimization Algorithm

```
1  set initial control parameters
2  begin
3      $t = 0$
4      initialize candidate(s)
5      evaluate initial candidate(s)
6      while not termination-condition do
7          $t = t + 1$
8          generate new candidate(s)
9          evaluate new candidate(s)
10         select solution(s) for next iteration
11         optional: update control parameters
12     end
13 end
```

The **generation** during the search process defines the methods for finding new candidates, with particular regard to how they use available or obtained information about the objective function. A standard approach is the variation of existing observations, as it utilizes, and to a certain extent, preserves previous iterations' information. Even by the simplest *hill-climber* class algorithms, which do not require any global information or stored knowledge of former iterations (Section 2.4.1), use the last obtained solution to generate new candidate(s). Sophisticated algorithms generate new candidates based on exploited and stored global knowledge about the objective function and fitness landscape. This knowledge is stored by either keeping an archive of all available or selected observations or implicitly using distribution or data models of available observations. Another option to generate new candidates is combining information of multiple candidates by dedicated functions or operators, mainly present in the trajectory class (Section 2.4.2). The exact operators for the generation and variation of candidate solutions are various and an essential aspect of keeping the balance between exploration and exploitation in a search strategy.

The **selection** defines the principle of choosing the solutions for the next iteration. We use the term *selection*, which has its origins in EC. Besides the most straightforward strategy of choosing the solution(s) with the *best* fitness, advanced selection strategies have emerged, which are mainly present in metaheuristics [Boussaïd et al., 2013]. These selection strategies are widespread in algorithms with several candidates per *generation* step; thus, EC introduced the most sophisticated selection methods [Eiben and Smith, 2015b]. The use of absolute differences in fitness or their relative difference is the most common strategy and called *ranked* selection, i.e., based on methods such as *truncation, tournament, or proportional* selection.

The **control parameters** determine how the search can be adapted and improved by controlling the above-mentioned key elements. We distinguish between *internal* and *external* parameters: External parameters, also known as offline parameters, can be adjusted by the user and need to be set a priori to the optimization run. Typical external parameters include the number of candidates and settings influencing the above-mentioned key elements. Besides standard theory-based defaults [Schwefel, 1993], they are usually set by either utilizing available domain knowledge, extensive a priori benchmark experiments [Gämperle et al., 2002], or educated guessing. Sophisticated *meta-optimization* methods were developed to exploit the correct parameter settings in an automated fashion. Well-known examples are sequential parameter tuning [Bartz-Beielstein et al., 2005a], iterated racing for automatic algorithm tuning [López-Ibáñez et al., 2016], *bonesa* [Smit and Eiben, 2011] or *SMAC* [Hutter et al., 2011]. In comparison to external parameters, internal ones are not meant to be changed by the user. They are either fixed to an absolute value, which is usually based on physical constants or extensive testing by the authors of the algorithm, or are *adaptive*, or even *self-adaptive*. Adaptive parameters are changed during the search process based on fixed strategies and exploited problem information [Eiben et al., 1999] without user influence. Self-Adaptive parameters are optimized during the run, e.g., by including them into the candidate vector x as an additional decision value. Algorithms using adaptive schemes tend to have better generalization abilities than those with fixed parameters. Thus, they are wildly successful for black-box problems, where no prior information about the objective function properties is available to set up parameters in advance [Hansen et al., 2003]. In general, the settings of algorithm control parameters directly affect the balance between exploration and exploitation during the search and are crucial for the search strategies and their performance.

Further, the *evaluation* step computes the fitness of the candidates. The evaluation is a crucial aspect, as it defines how and which information about any candidate solution is gathered by querying the objective function, which can significantly influence the search strategy and the utilized search operators. However, as important aspects of the evaluation are mostly problem-dependent, such as *noise*, *constraints* and *multiple objectives*. The handling of these aspects sometimes requires unique strategies, operators, or even specialized algorithm designs. These unique algorithms will not be covered in our taxonomy. However, strategies for handling these particular characteristics are often enhanced versions of in this taxonomy presented algorithms, e.g., for handling multiple objectives. Multi-objective problems include several competing goals, i.e., an improvement in one objective leads to a deterioration in another objective. Thus, no single optimal solution is available, but a set of equivalent quality, the non-dominated solutions, or so-called Pareto-set, where reasonable solutions need to be selected from [Fonseca et al., 1993; Naujoks et al., 2005]. A so-called decision-maker is needed to select the final solutions, which is often the user himself. Further, multi-objective algorithms can include special search operators, such as hyper-volume-based selection or non-dominated sorting for rank-based selection [Deb et al., 2002; Beume et al., 2007]. While most computer experiments are deterministic, i.e., iterations using the same value set for the associated decision variables should deliver the same results, real-world problems are often non-deterministic. They include non-observable disturbance variables and stochastic *noise*. Typical noise handling techniques include multiple evaluations of solutions to reduce the standard deviation and special sampling techniques. The interested reader can find a survey on noise handling by Arnold and Beyer [2003]. Moreover, optimization problems frequently include different constraints, which we need to consider during the optimization process. Constraint handling techniques can be directly part of the optimization algorithm, but most algorithms are designed to minimize the objective function and add constraint handling on top. Thus, algorithms integrate it by adjusting the fitness, e.g., by penalty terms. Different techniques for constraint handling are discussed by Coello [2002] and Arnold and Hansen [2012].

2.4 The Definition of Intuitive Algorithm Classes

In his work about evolution strategies, Rechenberg [1994] illustrated a visual approach to an optimization process: a mountaineer in an alpine landscape, attempting to find and climb the highest mountain. The usage of analogies to the natural world is a valuable method to explain the behavior of search algorithms. In the area of metaheuristics, the behavior of nature and animals inspired the search procedure of the algorithms: Evolutionary algorithms follow the evolution theory [Rechenberg, 1994; Eiben and Smith, 2015b]; particle swarm optimization [Kennedy and Eberhart, 1995; Shi and Eberhart, 1998] utilizes a strategy similar to the movement of bird flocks; ant colony optimization [Dorigo et al., 2006] mimics, as the name suggests, the ingenious pathfinding and food search principles of ant populations.

We take up the idea of optimization processes being human-like individuals and use it in the definition of our extended class names: the *mountaineer*, *sightseer*, *team*, *surveyor* and *chimera*. This additional naming shall accomplish the goal of presenting an evident and straightforward idea of the search strategies of the algorithms in the associated class.

2.4.1 Hill-Climbing Class: "The Mountaineer"

Intuitive Description 1 (The Mountaineer)
The mountaineer is a single individual who hikes through a landscape, concentrating on achieving his ultimate goal: finding and climbing the highest mountain. He is utterly focused on his goal to climb up that mountain. So while he checks different paths, he will always choose the ascending way and not explore the whole landscape.

Hill-Climbing algorithms focus their search strategy on greedy exploitation with minimal exploration. Hence, this class encompasses fundamental optimization algorithms with direct search strategies, including gradient-based algorithms and deterministic or stochastic hill-climbing algorithms. Gradient-based algorithms, also known as first-order methods, are in the first case applicable to differentiable functions, where the gradient information is available. If the gradient is not directly available, it can be approximated or estimated, for example, by *stochastic gradient descent* (SGD) algorithms [Ruder, 2016]. These algorithms have, by design, fast convergence to a local optimum situated in a region of attraction and commonly no explicit strategy for exploration. Overviews of associated algorithms were presented by Lewis et al. [2000] and Kolda et al. [2003]. Common algorithms include the quasi-Newton *Broyden-Fletcher–Goldfarb-Shanno* algorithm [Shanno, 1970], *conjugate gradients* [Fletcher, 1976], the direct search algorithm *Nelder-Mead* [Nelder and Mead, 1965], and stochastic hill climbers, e.g., the *(1+1)-Evolution Strategy* [Rechenberg, 1973; Schwefel, 1977].

Famous SGD algorithms are *adaptive moment estimation* (ADAM) [Kingma and Ba, 2014] and the *adaptive gradient algorithm* (AdaGrad) [Duchi et al., 2011]. They are frequently applied in machine learning, particularly for optimizing neural network weights with up to millions of parameters.

As this class defines fundamental search strategies, hill-climbers are often part of sophisticated algorithms as a fast-converging local optimizer. Hill-climbers do not utilize individual operators for the initialization of the single starting point. Thus, it is typically selected at random in the valid search space or based on prior knowledge.

The variation of the last observed selected candidate generates new candidates, commonly in the current solution's vicinity. For example, the stochastic hill climber utilizes random variation with a small step size compared to the range of the complete search interval. Gradient-based methods directly compute or approximate the gradients of the objective function to find the best direction and strength for the variation. Algorithms such as Nelder-Mead create new candidates by computing a search direction using simplexes.

The most common selection methods are elitist strategies, which evaluate the new candidate, compare it to the old solution, and keep the one with the best fitness as a new solution. Always selecting the best is known as a greedy search strategy, as it tries to improve as fast as possible. This greedy strategy leads to the outlined *hill-climbing* search, which performs a trajectory of small, fitness-improving steps, which forms in the ideal case a direct line to the nearest optimum. In general, these algorithms search locally for an optimum and do not exploit or use global function information.

The most critical control parameter is the variation step size, which directly influences the speed of convergence. As a result of this, state of the art is to use an adaptive variation step size that changes online during the search, often based on previous successful steps, for example, as defined in the famous *1/5 success rule* [Rechenberg, 1973].

2.4.2 Trajectory Class: "The Sightseer"

Intuitive Description 2 (The Sightseer)
The intuitive idea of this class is a single hiker looking for interesting places. During the search, the sightseer takes into account that multiple places of interest exist. Thus, it explores the search space or systematically visits areas to gather information about multiple locations and utilizes it to find the most desired ones.

Trajectory class algorithms still focus on exploitation but are supported by defined exploration methods. This class encompasses algorithms that utilize information from consecutive function evaluations. These algorithms are the connecting link between the hill-climbing and population class. While trajectory algorithms are a step towards population-based algorithms and allow the sampling of several solutions in one iteration, they use the principle of initializing and maintaining a single solution. This solution is the basis for variation in each iteration. Again, this variation forms a trajectory in the search space over consecutive iterations, similar to the hill-climbing class. Thus these methods are known as *trajectory methods* [Boussaïd et al., 2013]. While the initialization and generation of the trajectory class are similar to those of the hill-climbing class, the main differences can be found during the *selection*, as they utilize operators to guide the search process in a global landscape in specific directions. Two different strategies can be differentiated, which define two subclasses:

(i) The *exploring trajectory* class utilizes functions to calculate a probability of accepting a candidate as the (current) solution.

(ii) The *systematic trajectory* class utilizes a separation of the search space into smaller sub-spaces to guide the search into specific directions.

These different strategies are susceptible to the correct parametrization, which must be selected adequate to the underlying objective function.

2.4.2.1 Exploring Trajectory Algorithms

The exploring trajectory subclass encompasses algorithms that implement *selection* operators to balance exploration and exploitation to enable global optimization. The introduction of selection functions that allow the expansion of the search space and escape the *region of attraction* of a local optimum achieves exploration. *Simulated annealing* (SANN) Kirkpatrick et al. [1983], which is known to be a fundamental contribution to the field of metaheuristic search algorithms, exemplifies this class. The continuous version [Goffe et al., 1994; Siarry et al., 1997; Van Groenigen and Stein, 1998] of the SANN algorithm extends the iterated stochastic hill-climber. It includes a new element for the *selection*, the so-called acceptance function. It determines the probability of accepting an inferior candidate as a solution by utilizing a temperature (T) parameter, in analogy to metal annealing procedures. This dynamic *selection* allows escaping local optima steps by accepting movement in the opposite direction of improvement, which is the fundamental difference to a hill-climber and ultimately allows the global search. At the end of each iteration, a *cooling* operator adapts the temperature. This operator can be used to further balance the amount of exploration and exploitation during the search [Henderson et al., 2003]. A common approach is to start with a high temperature and steadily reduce T according to the number of iterations or to utilize an exponential decay of T. This steady reduction of T leads to a phase of active movement and thus exploring in the early iterations, while with decreasing T, the probability of accepting inferior candidates reduces. With approaching a T value of zero, the behavior becomes similar to an iterative hill-climber. Modern SANN implementations integrate *self-adaptive* cooling-schemes which use alternating phases of cooling and reheating [Locatelli, 2002]. These allow alternating phases of exploration and exploitation but require sophisticated control.

2.4.2.2 Systematic Trajectory Algorithms

This subclass encompasses algorithms, which base their search on a space partitioning utilizing the exposed knowledge of former iterations. They create sub-spaces that are excluded from *generation* and *selection*, or *attractive* sub-spaces, where the search is focused on. These search space partitions guide the search by pushing candidate generation to new promising or previous unexplored parts of the search space. An outstanding paradigm for this class is *Tabu Search* [Glover, 1989]. A so-called tabu list contains the last successful candidates and defines a sub-space of all evaluated solutions. In the continuous version, Siarry and Berthiau [1997]; Hu [1992]; Chelouah and Siarry [2000], small (hypersphere or hyperrectangle) regions around the candidates are utilized. The algorithm will consider these solutions or areas as forbidden for future searches, i.e., it selects no candidates situated in these regions as solutions. This process shall ensure to move away from known solutions and prevents identical cycling of candidates and getting stuck in local optima. The definition of the tabu list parameters can control exploration and exploitation by, e.g., by the number of elements or size of areas. The areas of search can also be pre-defined, such as in *variable* *neighborhood search (VNS)* [Hansen and Mladenovic, 2003; Hansen et al., 2010c; Mladenović et al., 2008]. The search strategy of VNS is to perform sequential local searches in these sub-spaces to exploit their local optima. The idea behind this strategy is that by using an adequate set of sub-spaces, the chance of exploiting a local optimum, which is near the global optimum, increases.

2.4.3 Population Class: "The Team"

Intuitive Description 3 (The Team)
The intuitive idea of this class is a group of individuals, which team *up to achieve their mutual goal together. They split up to explore different locations and share their knowledge with other members of the team.*

Population class algorithms utilize distributed exploration and exploitation. The idea of initializing, variation, and selection of several contemporary candidate solutions defines this class. The algorithms are commonly metaheuristics whose search concepts follow processes found in nature. Moreover, it includes algorithms building upon the population-based concept by utilizing models of the underlying candidate distributions. Due to utilizing a population, the *generation* and *selection* strategies of these algorithms differ significantly from the hill-climber and trajectory class. We subdivide this class into the regular population and model-based population algorithms, which particularly differ in how they generate new candidates during the search:

(i) The *regular population* (Section 2.4.3.1) generate and maintain several candidates with specific population-based operators.

(ii) The *model-based population* (Section 2.4.3.2) generate and adapt models to store and process information.

2.4.3.1 Regular Population Algorithms

Well-known examples of this class are *particle swarm optimization (PSO)* [Kennedy and Eberhart, 1995; Shi and Eberhart, 1998] and different *evolutionary algorithms (EA).* We regard EAs as state of the art in population-based optimization, as their search concepts dominate this field. Nearly all other population-based algorithms use similar concepts and are frequently associated with EAs. Fleming and Purshouse [2002] go as far to state:

> *In general, any iterative, population-based approach that uses selection and random variation to generate new solutions can be regarded as an EA.*

EAs follow the idea of evolution, reproduction, and the natural selection concept of *survival of the fittest.* In general, the field of EC goes back to four distinct developments, *evolution strategies* (ES) [Rechenberg, 1973; Schwefel, 1977], *evolutionary programming* [Fogel et al., 1966], *genetic algorithms* [Holland, 1992], and *genetic programming* [Koza, 1992]. The naming of the methods and operators matches with their counterparts from biology: candidates are *individuals* who can be *selected* to take the role of *parents*, mate and *recombine* to give birth to *offspring.* The population of individuals is *evolved* (varied, evaluated, and selected) over several iterations, so-called generations, to improve the solutions. Different overview articles shed light on the vast field of evolutionary algorithms [Bäck et al., 1997; Eiben and Smith, 2015b].

EAs *generate* new solutions typically by variation of a selected subset of the entire population. Typically, competition-based strategies, which also often include probabilistic elements, select the subsets. Either random variation of this subpopulation or *recombination* by crossover, which is the outstanding concept of EAs, generates new candidates. Recombination partly swaps the variables of two or more candidates, aggregated or combined to create new candidate solutions.

The population-based *selection* strategies allow picking solutions with inferior fitness for the variation process, which allows exploration of the search space. Several selection strategies exist. For instance, in *roulette* wheel selection, the chance of being selected is proportional to the ranking while all chances sum up to one. A spin of the roulette wheel chooses each candidate, where the individual with the highest fitness also has the highest chance of being selected. Alternatively, in *tournament selection*, different small subsets of the population are randomly drawn for several tournaments. Within these tournaments, the candidates with the best fitness are selected based on comparisons to their competitors. This competition-based selection also allows inferior candidates to win their small tournaments and participate in the variation.

EAs usually have several parameters, such as the selection intensity (i.e., the percent of truncation), variation step size, or recombination probability. Parameter settings, in particular adaptive and self-adaptive control for evolutionary algorithms is discussed in Angeline [1995]; Eiben et al. [1999]; Lobo et al. [2007]; Doerr et al. [2020]; Papa and Doerr [2020].

2.4.3.2 Model-based Population Algorithms

The model-based population class encompasses algorithms that explicitly use mathematical or statistical models of the underlying candidates. These algorithms generally belong to the broad field of EAs (Section 2.4.3.1) and use similar terminology and also operators.

Estimation of distribution algorithms (EDAs) are a well-known example for this class [Larrañaga and Lozano, 2001; Hauschild and Pelikan, 2011]. Compared to a regular population-based approach, a distribution model of selected promising candidates is learned in each iteration, which is then utilized to sample new candidates. The sampling distribution will improve and likely converge to generate only optimal or near-optimal solutions over the iterations. EDAs utilize models from univariate, over bivariate, to multivariate distributions, e.g., modeled by Bayesian networks or Gaussian distributions with typical parameters, such as mean, variance, and covariance of the modeled population. The search principle of EDAs was first defined for discrete domains and later successfully extended for continuous domains [Hauschild and Pelikan, 2011]. Popular examples for EDAs are *population-based incremental learning* (PBIL) [Baluja, 1994; Gallagher et al., 1999], the *estimation of Gaussian networks algorithm* (EGNA)[Larranaga et al., 1999], the *extended compact genetic algorithm* (eCGA) [Harik et al., 1999], and the *iterated density estimation evolutionary algorithm* (IDEA) [Bosman and Thierens, 2000]. The surrogate class distinction is that the underlying learned distribution models are directly utilized to sample new candidates instead of substituting the objective function.

A well-known and successful model-based algorithm is the *covariance matrix adaption - evolution strategy* (CMA-ES) [Hansen et al., 2003]. While it also utilizes a distribution model, its central idea extends the EDA approach by learning a multivariate Gaussian distribution model of candidate steps, i.e., their changes over iterations, instead of current locations [Hansen, 2006]. Moreover, instead of creating a new distribution model of selected candidates in each iteration, the model is kept and updated. This principle of updating the model is similar to applying evolutionary variation operators, such as recombination or mutation, to the candidates in a regular population-based algorithm. However, in the CMA-ES, the variation operators' target is the distribution model and not individual candidates.

Again, this class has several *control* parameters, which are often designed to be adaptive or self-adaptive. For example, the CMA-ES utilizes a sophisticated step-size control and adapts the mutation parameters during each iteration following the history of prior successful iterations, the so-called *evolution paths*. These evolution paths are exponentially smoothed sums for each distribution parameter over the consecutive prior iterative steps.

2.4.4 Surrogate Class: "The Surveyor"

Intuitive Description 4 (The Surveyor)
The intuitive idea of the surveyor is a specialist who systematically measures a landscape by taking samples of the height to create a topological map. This map resembles the actual landscape with a given approximation accuracy and is typically exact at the sampled locations and models the remaining landscape by regression. It can then be examined and utilized to approximate the quality of an unknown point and further be updated if new information is acquired. Ultimately it can be used to guide an individual to the desired location.

Surrogate class algorithms utilize distributed exploration and exploitation by explicitly relying on landscape information and a landscape model. These algorithms differ from all other defined classes in their focus on acquiring, gathering, and utilizing information about the fitness landscape. They utilize evaluated, acquired information to approximate the landscape and also predict the fitness of new candidates. As illustrated in Section 2.2.3, the surrogates depict the *maps* of the fitness landscape of an objective function in an algorithmic framework. A surrogate algorithm utilizes them for an efficient indirect search instead of performing multiple, direct, or localized search steps. We divide this class into two subclasses:

– *Surrogate model-based* algorithms utilize a surrogate model for all variation and selection processes.

– *Surrogate-assisted* algorithms utilize surrogates partly for supporting spezific search operators.

The distinction between the two subclasses is motivated by the different use of the surrogate model. While a SMBO algorithm generates new candidates solely by optimizing/predicting the surrogate, surrogate-assisted algorithms use it to support spezific search elements, e.g., selection.

For both classes, the surrogate model is a core element for the optimization. A perfect surrogate provides an excellent fit to observations while ideally possessing superior interpolation and extrapolation abilities. However, many available surrogates have significantly differing characteristics, advantages, and disadvantages. Model selection is thus a complicated and challenging task. If no domain knowledge is available, such as in real black-box optimization, it is often inevitable to test different surrogates for their applicability.

Common models are: linear, quadratic or polynomial regression, Gaussian processes (also known as Kriging) [Sacks et al., 1989; Forrester et al., 2008], regression trees [Breiman et al., 1984], artificial neural networks and radial basis function networks [Haykin, 2004; Hornik et al., 1989] including deep learning networks [Collobert and Weston, 2008; Hinton et al., 2012, 2006] and symbolic regression models [Augusto and Barbosa, 2000; Flasch et al., 2010b; McKay et al., 1995], which are usually optimized by genetic programming [Koza, 1992].

Further, much effort in current studies is to research the benefits of model ensembles, which combine several distinct models [Goel et al., 2007; Müller and

Shoemaker, 2014; Friese et al., 2016]. The goal is to create a sophisticated predictor that surpasses the performance of a single model. The drawback of these ensemble methods are model complexity, computation times and challenging tuning concerning efficient model selection, evaluation, and combination.

2.4.4.1 Surrogate Model-Based Algorithms

SMBO algorithms explicitly utilize a global approximation surrogate in their optimization cycle by following the concept of *efficient global optimization* (EGO) [Jones et al., 1998] and Bayesian Optimization (BO) [Mockus, 1974, 1994, 2012]. They are either fixed algorithms designed around a specific model, such as *Kriging*, or algorithmic frameworks with a choice of possible surrogates and optimization methods *sequential parameter optimization* [Bartz-Beielstein et al., 2005a; Bartz-Beielstein, 2010]. Further well-known examples for continuous frameworks are the *surrogate management framework* (SMF) [Booker et al., 1999] and the *surrogate modeling toolbox* (SMT) [Bouhlel et al., 2019]. Versions for discrete search spaces are *mixed integer surrogate optimization* (MISO) [Müller, 2016] and *efficient global optimization for combinatorial problems* (CEGO) [Zaefferer et al., 2014b]. The basis for our descriptions of surrogate-based algorithms is mainly EGO, and it is to note that the terminology of BO differs partly from our utilized terminology. A general surrogate-based algorithm can be described as follows (cf. Section 2.2.3):

1. The initialization is done by sampling the objective function at k positions with $\mathbf{y}_i = f(\mathbf{x}_i), 1 \leq i \leq k$ to generate a set of observations $\mathcal{D}_t = \{(\mathbf{x}_i, \mathbf{y}_i), 1 \leq i \leq k\}$. The sampling design plan is commonly selected according to the surrogate.

2. Selecting a suitable surrogate. The selection of the correct surrogate type can be a computational demanding step in the optimization process, as often no prior information indicating the best type is available.

3. Constructing the surrogate $s(\mathbf{x})$ using the observations.

4. Utilizing the surrogate $s(\mathbf{x})$ to predict n new promising candidates $\{\mathbf{x}^*_{1:n}\}$, e.g., by optimization of the *infill* function with a suitable algorithm. For example, it is reasonable to use algorithms that require a large number of evaluations as the surrogate itself is (comparatively) very cheap to evaluate.

5. Evaluating the new candidates with the objective function $y^*_i = f(\mathbf{x}^*_i), 1 \leq i \leq n$.

6. If the stopping criterion is not met:
   Updating the surrogate with the new observations $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{(\mathbf{x}^*_i, y^*_i), 1 \leq i \leq n\}$, and repeating the optimization cycle (4.-6.)

For the initialization, the model building requires a suitable sampling of the search space. The initial sampling has a significant impact on the performance and should be carefully selected. Thus, the initialization commonly uses candidates following different information criteria and suitable experimental designs. For example, it is common to built linear regression models with factorial designs and preferably couple Gaussian process models with space-filling designs, such as *Latin hypercube sampling* [Montgomery, 2017; Sacks et al., 1989].

The generation has two aspects: the first is the choice of surrogate itself, as it is used to find a new candidate. The accuracy of a surrogate strongly relies on the selection of the correct model type to approximate the objective function. By selecting a particular surrogate, the user makes certain assumptions regarding the characteristics of the objective function, i.e., modality, continuity, and smoothness [Forrester and Keane, 2009]. Most surrogates are selected to provide continuous, low-modal, and smooth landscapes, rendering the optimization process computationally inexpensive and straightforward. The second aspect is the optimizer which variates the candidates for searching on the surrogate and the approximated fitness landscape. As the surrogates are often fast to evaluate, exhaustive *exact* search strategies, such as *branch and bound* in EGO Jones et al. [1998] or multi-start hill-climbers, are often utilized, but it is also common to use sophisticated population-based algorithms.

The surrogate prediction for the expected best solution is the basis of the selection of the next candidate solution. Instead of a simple mean fitness prediction, defining an infill criterion or acquisition function is common. Typical choices include the probability of improvement [Kushner, 1964], expected improvement [Jones et al., 1998] and confidence bounds [Cox and John, 1997]. Expected improvement is a common infill criterion because it is a balance of exploration and exploitation by utilizing both the predicted best mean value of the model and the model uncertainty. The optimization of this infill criterion then selects the candidate. Typically, in each iteration for evaluation and the model update, the algorithm selects only a single candidate. Multi-infill selection strategies are also possible. SMBO algorithms include a large number of control elements, starting with necessary components of such an algorithm, including the initialization strategy, the choice of surrogate, and optimizer. In particular the infill criteria have an enormous impact on the performance. Even for a fixed algorithm, the amount of (required) control is extensive. The most important are the model parameters of the surrogate.

2.4.4.2 Surrogate-assisted Algorithms

Surrogate-assisted algorithms utilize a search strategy similar to the population class, but employ a surrogate particular in the *selection* step to preselect candidate solutions based on their approximated fitness and *assist* the evolutionary search strategy [Ong et al., 2005; Jin, 2005; Emmerich et al., 2006; Lim et al., 2010; Loshchilov et al., 2012]. Only parts of the new candidates are preselected utilizing the surrogate, while another part follows a direct selection and evaluation process. The generation and selection of a new candidate are thus not based on optimizing the surrogate landscape, which is the main difference to the SMBO algorithms. The surrogate can be built on the archive of all solutions or locally on the current solution candidates. An overview of surrogate-assisted optimization is given by Jin [2011], including several examples for real-world applications, or by Haftka et al. [2016] and Rehbach et al. [2018], with focus on parallelization.

2.4.5 Hybrid Class: "The Chimera"

Intuitive Description 5 (The Chimera)
A chimera is an individual, which is a mixture, composition, or crossover of other individuals. It is an entirely new being formed from existing parts of various species and utilizes their strengths to be versatile.

We describe the explicit combination of algorithms or their components as the hybrid class. Hybrid algorithms utilize existing components and concepts, which have their origin in an algorithm from one of the other classes, in new combinations. They are mainly present in current research regarding the automatic composition or optimization of algorithms to solve a specific problem or a problem class. Hyperheuristic algorithms also belong to this class. Overviews of hybrid algorithms were presented by Talbi [2002]; Blum et al. [2011]; Burke et al. [2013]. There are two kinds of hybrid algorithms:

1. *Predetermined Hybrids* (Section 2.4.5.1) have a fixed algorithm design, which is composed of certain algorithms or their components.
2. *Automated Hybrids* (Section 2.4.5.2) use optimization or machine learning to search for an optimal algorithm design or composition.

The hybrid class contains many algorithms, and it can be challenging to distinguish a particular class. However, mainly the predetermined hybrids can be additionally described by their main components so that their origin remains clear, e.g., an evolutionary algorithm coupled with simulated annealing could be defined as *population-trajectory hybrid.* For automated hybrids, this definition is no longer possible, as they couple a large number of different components, and also, the algorithm structure is part of their search so that the final algorithm structure can differ for each problem.

2.4.5.1 Predetermined Hybrid Algorithms

The search strategies of this class improve or tackle algorithm weaknesses or amplify their strengths. The algorithms are often given distinctive roles of exploration and exploitation, as they are combinations of an explorative global search method paired with a local search algorithm. For example, population-based algorithms with remarkable exploration abilities pair with local algorithms with fast convergence. This approach gives some benefits, as the combined algorithms can be adapted or tuned to fulfill their distinct tasks. Also well known are *Memetic algorithms*, as defined by Moscato [1989], which are a class of search methods that combine population-based algorithms with local hill-climbers. An extensive overview of memetic algorithms is given by Molina et al. [2010]. They describe how different hybrid algorithms can be constructed by looking at suitable local search algorithms regarding their convergence abilities.

2.4.5.2 Automated Hybrid Algorithms

Automated hybrids are a special kind of algorithms, which do not use predetermined search strategies, but a pool of different algorithms or algorithm components, where the optimal strategy can be composed of [Lindauer et al., 2015; Bezerra et al., 2014]. Hyperheuristics belong to this class, particularly those that generate new heuristics [Burke et al., 2010; Martin and Tauritz, 2013]. Automated algorithm selection tries to find the most suitable algorithm for a specific problem based on machine learning and problem information, such as explorative landscape analysis [Kerschke and Trautmann, 2019]. Instead of selecting individual algorithms, it is tried to select different types of operators for, e.g., generation or selection, to automatically compose a new, best-performing algorithm for a particular problem [Richter and Tauritz, 2018]. Similar to our defined elements, search operators or components of algorithms are identified, extracted, and then again combined to a new search strategy.

Generation, variation, and selection focus in this class on algorithm components, instead of candidate solutions. The search strategies on this upper level are similar to the presented algorithms; hence, for example, evolutionary or Bayesian techniques are common [Guo, 2003; van Rijn et al., 2017].

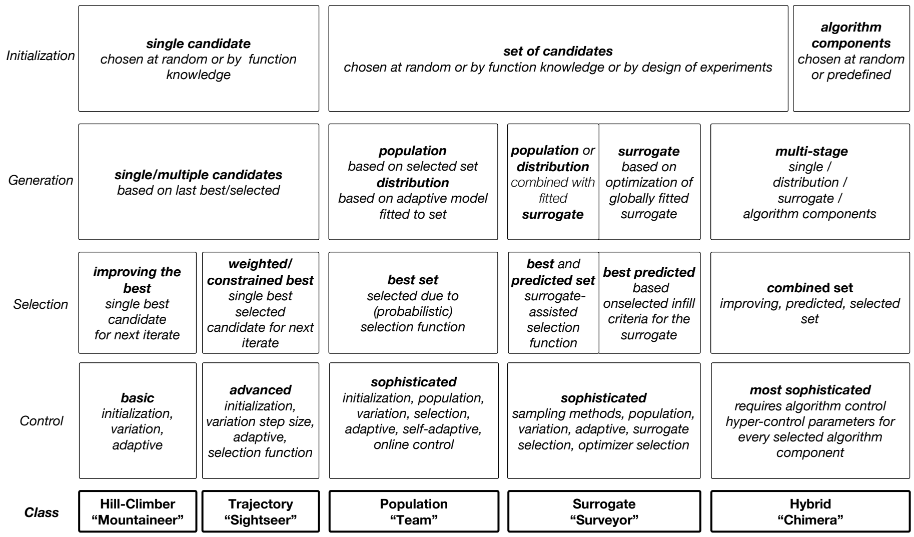

Initialization
single candidate chosen at random or by function knowledge
set of candidates chosen at random or by function knowledge or by design of experiments
algorithm components chosen at random or predefined
Generation
single/multiple candidates based on last best/selected
population based on selected set distribution based on adaptive model fitted to set
population or distribution combined with fitted surrogate
surrogate based on optimization of globally fitted surrogate
multi-stage single / distribution / surrogate / algorithm components
Selection
improving the best single best candidate for next iterate
weighted/ constrained best single best selected candidate for next iterate
best set selected due to (probabilistic) selection function
best and predicted set surrogate-assisted selection function
best predicted based onselected infill criteria for the surrogate
combined set improving, predicted, selected set
Control
basic initialization, variation, adaptive
advanced initialization, variation step size, adaptive, selection function
sophisticated initialization, population, variation, selection, adaptive, self-adaptive, online control
sophisticated sampling methods, population, variation, adaptive, surrogate selection, optimizer selection
most sophisticated requires algorithm control hyper-control parameters for every selected algorithm component
Class
Hill-Climber "Mountaineer"
Trajectory "Sightseer"
Population "Team"
Surrogate "Surveyor"
Hybrid "Chimera"

Figure 2.3: *Overview of defining algorithm features per search element and class. Overlapping features indicate the close connection between the individual classes.*

2.4.6 Taxonomy: Summary and Examples

Figure 2.3 illustrates an overview of all classes and connected features. It outlines the initialization, generation, selection, and control of the individual components for each class. The algorithm features are partly distinct and define their class, while others are shared. The figure shows the strong relationship between the algorithm classes; for example, the hill-climbing and trajectory class share similar characteristics. The algorithms of these classes are similar in their search strategy and built upon each other. The presented scheme is intended to cover most concepts. However, available algorithms can also have characteristics of different classes and do not fit the presented scheme. For example, some components of the more complex classes can also be utilized in the less complex classes, e.g., self-adaptive control schemes also apply for hill-climbers but are typically found in the population class. Table 2.1 describes examples for each of the defined algorithm classes and outlines their specific features. Again, the table is not intended to present a complete overview; instead, for each class and subclass, at least one example is given to present the working scheme. Other algorithms can be easily classified utilizing the scheme presented in Figure 2.3.

2.5 Algorithm Selection Guidelines

The selection of the best-suited algorithm poses a common problem for practitioners, even if experienced. In general, it is nearly impossible to predict the performance of any algorithm for a new, unknown problem. We thus recommend first gathering all available information about the problem if confronted with a new optimization problem. The features of a problem can be an excellent guideline to select at least an adequate algorithm class, where the users' choice and experience can select a concrete implementation.

Our guideline is strongly connected to the idea of *exploratory landscape analysis* (ELA) [Mersmann et al., 2011], which aims to derive problem features with the final goal of relating those features to suitable algorithm classes. For example, these features include information about convexity, multi-modality, the global structure of the problem, problem separability, and variable scaling. ELA's final goal is to provide the best-suited algorithms to previous unknown optimization problems based on the derived landscape features. This goal requires rigorous experimentation and benchmarking to match algorithms or algorithm classes to the landscape features [Kerschke and Trautmann, 2019]. As this information is not yet available, we extracted a small, high-level set of these features for our guideline, considering mainly the multi-modality and unique function properties as being expensive to evaluate.

Moreover, our selection is based on the available resources, both in terms of available evaluations and computational resources. To help with the selection, we developed a small decision graph that builds upon these significant features. The provided guideline is experience-based and utilizes basic concepts; it is not intended to serve as an absolute policy; instead, as the first recommendation if a new problem is considered. The graph is outlined in Figure 2.4.

A *hill-climbing* algorithm is in the first place suitable for unimodal functions or to exploit local optima or for cases where gradient information can be derived

Table 2.1: *Summary of example algorithms for each class of the presented taxonomy*

name	class	specifics
1+1-Evolution Strategy [Rechenberg, 1973; Schwefel, 1977]	hill-climber	probabilistic, adaptive mutation rates
L-BFGS [Liu and Nocedal, 1989], **CG** [Fletcher, 1976]	hill-climber	approximating gradient
Variable Neighborhood Search [Hansen and Mladenovic, 2003]	trajectory (systematic)	separation of search space in individually searched subspaces
Tabu Search [Glover, 1989; Siarry and Berthiau, 1997]	trajectory (systematic)	Tabu-list of search restricted solutions/areas
Simulated Annealing [Kirkpatrick et al., 1983]	trajectory (exploring)	control variable: temperature to define exploration strength
Evolutionary Algorithms [Bäck, 1996; Eiben and Smith, 2015b,b]	population (regular)	different variation and selection strategies, general framework
Particle Swarm Optimization [Kennedy and Eberhart, 1995]	population (regular)	exploration and exploitation strategy based on behavior in swarms
Estimation of Distribution Algorithms [Larrañaga and Lozano, 2001; Hauschild and Pelikan, 2011]	population (model)	probabilistic distribution model of underlying population
Covariance Matrix Adaption - ES [Hansen et al., 2003]	population (model)	modeling search steps, adaptive or pre-defined parameters
Efficient Global Optimization [Jones et al., 1998]	surrogate (based)	kriging models, expected improvement
Bayesian Optimization [Mockus, 1974, 1994, 2012]	surrogate (based)	general framework
Surrogate-Assisted EAs [Ong et al., 2005; Lim et al., 2010]	surrogate (assisted)	general framework, assisting evolutionary algorithms with surrogates
Memetic Algorithms [Moscato, 1989]	hybrid (predetermined)	combining EAs with hill-climber class unimodal search algorithms
Hyperheuristics [Burke et al., 2003]	hybrid (automated)	automatic selection of entire heuristics or individual search operators

from the objective function. Gradient-based algorithms are incredibly successful in optimizing large-scale optimization problems, such as frequently found in AI. However, they always have a high risk of getting stuck in local optima. It can be applied for global optimization to multimodal landscapes if an adequate multi-start strategy is employed. These multi-start strategies typically demand a high number of function evaluations and are most reasonable to be used if objective functions are not expensive.

Exploring trajectory algorithms are suitable for searches in unimodal and multimodal problems. As they do not rely on stored information of former iterations during their search, they are also an excellent choice to handle dynamic objective functions[Carson and Maria, 1997; Corana et al., 1987; Faber et al., 2005]. However, the rather simplistic utilization of exploited global information renders them inefficient for challenging and expensive optimization problems. Moreover, the control parameters have a significant effect on the performance of these algorithms. We thus advise to tune them in an offline or online fashion.

The central concept of *systematic trajectory* algorithms is to use the information of evaluated solutions and to direct the search to former unknown regions to avoid early convergence to a non-global optimum. The strategic use of sub-spaces allows precise control of exploration and exploitation and mainly ensures a high level of exploration. They include a large number of parameters, such as the number or size of sub-spaces, which makes them very vulnerable to false setups and less good universal solvers. If correctly tuned, algorithms from this class are suitable and efficient for multimodal problems. Algorithms using a pre-defined separation of the search space, such as VNS, can utilize domain knowledge for the initial definition of the sub-spaces. This pre-defined separation renders them useful for problems where the region of the global optimum is roughly known but not particularly suitable for black-box problems.

Population-based algorithms are very flexible in their implementation and adaptable by tuning. They are robust and suitable to solve a large class of problems, including multimodal, multi-objective, dynamic, and black-box problems, even with noise or discontinuities in the fitness landscape [Jin and Branke, 2005; Marler and Arora, 2004]. Further, they have successfully been applied to a large number of different industrial problems [Fleming and Purshouse, 2002] but typically require a relatively large number ($\gg 100 \times dim$) of function evaluations to be successful. This inefficiency makes them not the first choice for expensive problems, where the number of evaluations is sharply limited. Different mechanisms and strategies for controlling the balance between exploration and exploitation cause flexibility and robustness. A good overview is presented in the survey by Črepinšek et al. [2013]. The detailed survey classifies the different available evolutionary approaches and presents an intensive discussion on which mechanisms influence exploration and exploitation. Theoretical aspects of evolutionary operators are discussed by Beyer [2013]. *Parameter tuning and control* influence the performance of EAs, e.g., by the setting of population size, mutation strength, and selection probability. An extensive overview of the different on- and offline tuning approaches for parameter control in EAs was published by Eiben et al. [1999]. Further common strategies of controlling exploration and exploitation and multimodal optimization are so-called niching strategies, which utilize sub-populations to maintain the diversity of the population, investigate several regions of the search space in parallel or conduct defined tasks of exploring and exploiting [Shir and Bäck, 2005; Filipiak and Lipinski, 2014].

One step further, *model-based* algorithms try to combine the benefits of statistical models and their capability of storing and processing information with population-based search operators. They are high-level metaheuristics intended to be flexible, robust, and applicable to a large class of problems, particularly those with unknown function properties. This generalizability makes them very successful in popular black-box benchmarks [Hansen et al., 2010b]. For example, the design of the CMA-ES seeks to make the algorithm performance robust and not dependent on the objective function or tuning. The various *control parameters* of the algorithm were pre-defined based on theoretical aspects and practical benchmarks.

SMBO algorithms were created to solve expensive problems with the help of a surrogate. Their focus on high evaluation efficiency renders them particularly suitable for problems where only a small number of function evaluations are possible, such as real-world optimization or physical simulations. The downside of surrogate modeling is that it can impose a high computational complexity in the surrogate fitting and prediction process. This cost is usually low compared to the resource cost of a real-world function or an expensive simulation, but it can get significant compared to a cheap (i.e., fast to evaluate, low cost) objective function. A SMBO approach is not the best choice for problems that need very fast optimizations or allow large numbers of functions evaluations ($\gg 100 \times n$). Moreover, solving high-dimensional problems or those with a large number of samples requires unique strategies or specialized models [Regis and Shoemaker, 2013]. In general, the selection of an adequate model, experimental design, and optimizer requires both domain knowledge and expertise. Forrester and Keane [2009] and Bartz-Beielstein and Zaefferer [2017] give overviews of SMBO, different surrogate models, and infill criteria and match surrogates to problem classes and give hints about their applicability. SMBO was successfully applied to different applications, including expensive optimization problems [Lizotte, 2008; Khan et al., 2002] and machine learning [Snoek et al., 2012; Swersky et al., 2013; Stork et al., 2019b; Gaier et al., 2018] Surrogate-assisted optimization is more flexible, as it combines the strength of population-based algorithms with the evaluation efficiency of the surrogate.

Hybrid algorithms apply to a large class of problems, dependent on the origin of their algorithms or components. The most recent algorithms from this hybrid class search for the best algorithm design and composition automatically, even at different problem stages or search stages. The method of automatic algorithm selection has shown to be able to outperform a single algorithm on a set of benchmark functions [van Rijn et al., 2017; Vermetten et al., 2019]. However, this procedure also requires either a large amount of function knowledge or a large evaluation budget and computation time. Their immense complexity includes the risk that the automatic composition does not lead to improved performance due to the problematic balancing and required tuning of the distinct algorithms. Further, their sophisticated search strategies with a large number of control parameters can make them difficult to tune. For the automatic algorithm selection, numerous operators influence the convergence behavior, and the search strategy itself becomes a black-box that is challenging to comprehend.

The guideline in Figure 2.4 can be utilized in the following way: If features of the objective function are known (e.g., it is unimodal as the sphere function), a suitable algorithm can be selected based on these features, e.g., a hill-climber. If no information about the function is known, e.g., it is a complete black-box, we recommend using algorithms with a high generalization ability, e.g., a CMA-ES. Another essential decision regards the number of possible function evaluations and the costs of the objective function. If the number of available function evaluations is low and the objective function is costly, SMBO algorithms are a robust choice. Recent algorithms based on automated algorithm selection aim to optimize this selection process by extracting problem features while optimizing and adapting the search strategy to them [van Rijn et al., 2017; Kerschke and Trautmann, 2019].

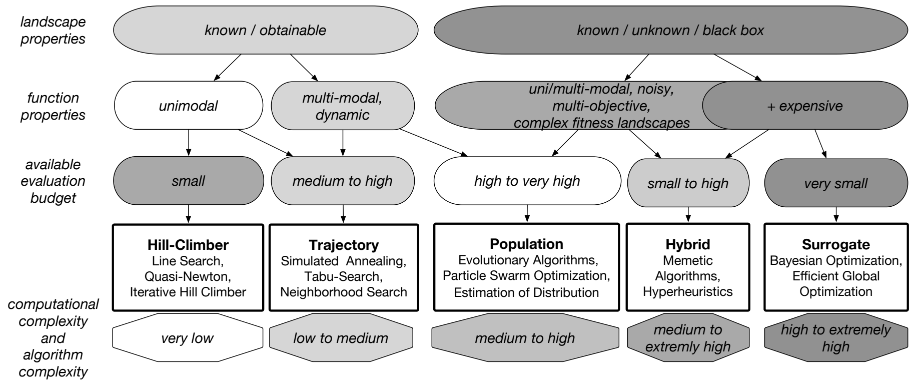

landscape properties
known / obtainable
known / unknown / black box
function properties
unimodal
multi-modal, dynamic
uni/multi-modal, noisy, multi-objective, complex fitness landscapes
+ expensive
available evaluation budget
small
medium to high
high to very high
small to high
very small
Hill-Climber
Line Search, Quasi-Newton, Iterative Hill Climber
Trajectory
Simulated Annealing, Tabu-Search, Neighborhood Search
Population
Evolutionary Algorithms, Particle Swarm Optimization, Estimation of Distribution
Hybrid
Memetic Algorithms, Hyperheuristics
Surrogate
Bayesian Optimization, Efficient Global Optimization
computational complexity and algorithm complexity
very low
low to medium
medium to high
medium to extremly high
high to extremely high

Figure 2.4: *Algorithm selection guideline. The figure connects landscape and function properties, as well as the available budget to a suitable algorithm class and outlines their computational complexity.*

2.6 Conclusion and Outlook

We presented a comprehensive overview of global optimization algorithms by creating a new taxonomy in this work. We set a particular focus on covering a broad range of optimization algorithms, including surrogates, metaheuristics, and algorithm combinations, because existing taxonomies do not cover these well. Based on a generalized algorithm scheme, we defined four characteristic elements of optimization algorithms, i.e., how they initialize, generate, and select solutions, how these solutions are evaluated, and finally, how these algorithms can be parametrized and controlled. With these elements, we created a generalized view on optimization algorithms by identifying their specific components. These components were then used to divide algorithms in the *hill-climber*, *trajectory*, *population*, *surrogate* and *hybrid* class and to identify similarities and differences in their search strategies.

We can conclude that most algorithms and algorithm classes have a close connection and share similar components, operators, and a large part of their search strategies. Current research for the automated design of algorithms builds upon this fact. It generalizes algorithms by breaking them down into their components and again combining these components into algorithms. This design process can be fully automatic, selecting components based on known features of the problem. Recent research aims in this direction [Lindauer et al., 2015; Kerschke and Trautmann, 2019; Bezerra et al., 2014; van Rijn et al., 2017]. These automatic methods can benefit from classifications of algorithms to build their design spaces of suitable algorithms classes. Our classes and identified elements can be utilized to create a design scheme for each class. For example, employ an algorithm generator for hill-climbers or population-based algorithms. Thus, our taxonomy is particularly useful for these automatic methods, as it includes a broad set of current developments in the presented classification scheme. Based on the human behavior in pathfinding, our set of accompanying analogies mainly helps novices comprehend the algorithm search strategies' fundamentals. We further outlined an algorithm selection guideline with best practices for more advanced practitioners, which can support them if they face a new problem and choose a suitable optimization algorithm class.

Our taxonomy has several limitations, which are part of future work: The first is that our current taxonomy cannot cover all available algorithms, particularly those that handle specific problem characteristics, e.g., multi-objective, noisy, or dynamic objective functions. The second limitation concerns the missing benchmarking evidence for our algorithms selection guideline: There are excellent structured collections of (test) problems, e.g., the *black-box optimization benchmark* (BBOB)[Hansen et al., 2010a]. Here we offer a system for algorithms providing a structured way of positioning algorithms. Combining the two facilitates systematic empirical studies on problem type and algorithm type combination. In other words, systematic benchmarking: To enable fair comparisons, each algorithm from a selected algorithm class has to be compared to several algorithms not belonging to this class. This implies a large amount of work on a long term done by the research communities, going far beyond this chapter's scope.

Exciting challenges for future algorithm design arise from problems in several categories. The first is *industry 4.0*. Digitalization in manufacturing and engineering, particularly the rapid development of communicating sensors and machines in engineering, requires new designs. Suitable optimization algorithms need to be directly included in the production cycle, adapting to generate robust solutions in challenging dynamic environments in an online manner, robustly improving themselves over a long-time period in the field. The second category is machine learning techniques, particularly training models with a massive amount of parameters, such as deep learning networks. They require algorithm designs that combine sampling and computation time efficiency, which is a challenging optimization problem. Further, the field of structural optimization or hyper-parameter optimization of neural networks is advancing. In this area, interesting research is dedicated to neuroevolution [Stanley et al., 2019] or population-based training [Jaderberg et al., 2017]. Moreover, surrogate-based algorithms are considered to optimize neural networks [Gaier et al., 2018; Stork et al., 2019b]. Finally, algorithms arising from new computing paradigms, such as quantum computing [Pittenger, 2012] or neuromorphic computing [Schuman et al., 2017], might completely change our current perspective on how optimization algorithms work.

3 Comparison of Evolutionary and Surrogate Model-Based Optimization of Neural Network Weights

Chapter 3 was published as:

Stork J, Zaefferer M, Fischbach A, Rehbach F, & Bartz-Beielstein T (2017) Surrogate-assisted learning of neural networks. *In Proceedings of the 27. Workshop Computational Intelligence* (pp. 195-210), KIT Scientific Publishing. DOI: 10.5445/KSP/1000074341

3.1 Introduction

Surrogate model-based optimization[1] has proven to be very successful if applied to expensive industrial problems. Using a data-driven surrogate model of an objective function during an optimization cycle has many benefits, such as being cheap to evaluate and further providing information about the objective landscape and the parameter space. In preliminary work, it was researched how SMBO can help to optimize the structure of an artificial neural network controller [Flasch et al., 2010a]. In this work, we will focus on how surrogates can help to improve the direct learning process of a transparent feed-forward ANN controller. As an initial case study, we will consider a manageable real-world control task: the *elevator supervisory group control* (ESGC) problem using a simplified simulation model [Bartz-Beielstein et al., 2005b]. We use this fast-to-evaluate simulation model as a benchmark to indicate the applicability and performance of SMBO to this kind of task. Complex ESGC simulators have huge computation costs, which allows only a few function evaluations. The results indicate that SMBO is capable of outperforming metaheuristic optimization methods for this low number of evaluations. Furthermore, we demonstrate that the surrogate is helpful for the significance analysis of the inputs and ANN weights.

3.2 Motivation

Recent advancements in control and robotics have shown that computational intelligence methods are becoming more and more significant. Robot control policies are no longer just trained by machine learning algorithms. Instead, robots learn how to solve a specific task by themselves, e.g., by methods of evolutionary robotics or reinforcement learning [Bongard, 2013]. In a real-world environment, evolutionary optimization of control policies can be costly, as the fitness of a particular robot action can only be evaluated after a sequence of time

[1]To follow the term definitions in Chapter 1 and harmonize, the original term "surrogate-assisted optimization" was changed to "surrogate model-based optimization"

steps, which can easily be in minutes or hours. Thus, these real-world processes pose a complex optimization problem, and standard methods are not suitable for the time requirements of these tasks. ANNs are a well-established type of controller in evolutionary robotics. Here, the coefficients and the topology of the network need to be optimized for optimal performance. More recent and sophisticated approaches for developing and learning controllers, such as *neuroevolution of augmenting topologies* [Stanley and Miikkulainen, 2002] were invented to handle these optimization processes, but they still need many evaluations to adapt the ANNs.

– We hypothesize that enhancing this learning process by employing SMBO, which has proven to be able to perform significantly well for expensive industrial optimization tasks [Ong et al., 2003; Queipo et al., 2005], should be beneficial.

– As a second hypothesis, we assume that surrogate models can help retrieve additional useful information about the objective function, e.g., the importance of certain inputs.

We want to test this hypothesis based on empirical experiments with a small real-world task simulator, implemented as a simple ANN and cheap and fast to evaluate. The results can be transferred to more sophisticated tasks, like a real-world process. The results should indicate the applicability and essential performance of SMBO methods in comparison to state-of-the-art EC algorithms. The variable importance information provided by the surrogate models is also analyzed concerning their usefulness. For instance, variable importance could be helpful to identify especially important or defective inputs sensors of a physical controller, e.g., in an evolutionary robotics task.

3.3 Elevator Supervisory Group Control

Today, elevator systems are present everywhere in urban areas. They need to be optimized to achieve the desired service quality in terms of waiting time for the customers, as well as in terms of energy efficiency. They are controlled by an elevator group controller, which assigns the elevator cars to particular floors and destinations based on the customer service calls. The ESGC problem, as introduced by [Bartz-Beielstein et al., 2005b] is a so-called destination call system, where the customer can choose their desired destination on the floor level outside the elevator cars. In the introduced problem instance, the controller is implemented as a sophisticated ANN, where the specific structure and weights depict a certain control strategy. The optimization of these weights imposes a set of challenges, which render this task highly complex:

- The topology of the fitness function is, to a high extent, non-linear as well as multi-modal.
- The traffic load is dynamic and stochastic, as customers do not arrive in a deterministic manner.
- Classic gradient-based optimization cannot be applied efficiently due to the complex objective functions.
- The simulator is computationally expensive, which limits the number of function evaluations.

As a consequence of the complexity of such simulators, Bartz-Beielstein et al. [2005b] utilizes a simplified validation model of an ESGC system, the *sequential ring* (S-Ring).

3.3.1 S-Ring Perceptron Simulator

The S-Ring was introduced to benchmark different ESGC algorithms independent of elevator-floor configurations. It uses a simplified ANN perceptron to control the elevators, where the connection weights can be modified and represent the variables of an optimization problem. Each weight setting will result in a specific control strategy tested on simulations of different traffic situations. The S-Ring has low computational costs, which allows us to use an ESGC instance as a benchmark for a large variety of algorithms. Using different traffic situations will lead to a fitness function that is subject to noise. The S-Ring optimization problem is defined by the number of elevators n, the number of floors m, the probability of an arriving customer per floor p and $(2 \times m) - 2$

states s. The target of the optimization is the ANN weight vector $\mathbf{w}$ with length s, which depicts the control policy. This objective function evaluates the average waiting time of all customers $\mathbf{c}_i, i = 1, ..., s$ during a simulated traffic situation with t steps. For a given set of n, m, p, the performance is only influenced by the weight vector $\mathbf{w}$ of the ANN controller. Thus, as further used during this paper, the simplified problem can be written as $\arg\min f(\mathbf{w})$. The parameters n, m, p were set as shown in Table 3.1. The table also displays the

Table 3.1: *S-Ring Configuration*

nFloors (nStates)	nElevators	probNewCustomer	nTimeSteps
6(10)	2	0.3	10000

number of time steps for a single simulation run, which was set relatively high to simulate an extended period. For each simulation run, the same period was used, resembling a specific fixed time frame, e.g., a particular day in a year. By choosing a fixed time frame, we removed the noise of the problem, which renders the problem simpler to optimize. Moreover, the problem was adapted by setting the desired customer service quality of the ground floor to a high priority, while the second floor was set to a lower priority. This should simulate a typical real-world hotel scenario, where it is wanted to immediately serve customers arriving in the lobby. The second floor displays an internal service area, which is of low priority for the quality of service.

3.4 Methods for Optimization of Neural Network Controllers

A standard method to optimize the ANN controller in a supervised learning process is gradient-based *back-propagation*. In back-propagation, the weights are optimized by utilizing a set of training data with labeled output data. The differences between the actual and desired outputs allow computing a loss function and further gradient information. In the case of the S-ring simulator, labeled training samples to compute gradient information are not (directly) available. We receive a fitness value only after evaluating the weights in a designated simulation run and can not utilize supervised learning by defined input-output pairs. The problem can be considered a reinforcement learning task but without per-state reward information. Instead, only the performance of a complete simulation is acquired at the end of each run. We thus rely on metaheuristic optimization and SMBO.

3.4.1 Metaheuristic Optimization

Metaheuristics are sophisticated heuristics, which are often inspired by nature. They utilize stochastic processes (randomization) and usually do not require any (direct) gradient information. Metaheuristics are known to be general solvers who apply to a large variety of *global* problems without needing a priori information. They are suitable for highly non-linear and multi-modal problems and so-called *black-box* problems, where no information about the topology of the objective function is known. No algorithm can deliver its best performance for every problem without adapting its control parameters; By parameter tuning [Bartz-Beielstein et al., 2005a; Eiben et al., 1999], we can exploit beneficial parameter settings is very time-demanding. To provide reliable results without putting much effort into algorithm tuning, we selected four different state-of-the-art R implementations of common metaheuristics from the range of *simulated annealing* methods and *evolutionary algorithms* for our comparison. Simulated annealing [Kirkpatrick et al., 1983] is inspired by annealing processes in metallurgy, where materials are heated and cooled to change their physical structure. Simulated annealing follows the base principle of a greedy stochastic algorithm but implements a control strategy that allows accepting solutions with lesser fitness. This allows to escape local optima and establishes a global search strategy. EAs [Bäck, 1996] are based on the principles of natural selection: in each generation, a population of individuals (e.g., solutions $\mathbf{w}$) is evolved by mutation, recombination, and selection steps. The selected R packages are *DEoptim*, *GA*, *GenSA* and *genoud*:

- **DEoptim** [Mullen et al., 2011] is an R implementation of the differential evolution algorithm [Storn and Price, 1997], which belongs to the class of EC algorithms. It is designed for global optimization.

- **GA** [Scrucca, 2013] is a package which implements a genetic algorithm that allows optimization of real and integer problems.

- **GenSA** provides a version of generalized simulated annealing [Xiang et al., 2013].

- **rgenoud** [Mebane Jr and Sekhon, 2011; Sekhon and Mebane, 1998] is an R package which provides and implementation of a so-called *hybrid* algorithm. This algorithm combines EAs with the derivative-based quasi-Newton method *Broyden-Fletcher-Goldfarb-Shanno* (BFGS).

DEoptim and GA were chosen due to personal preference, while the two latter (GenSA and genoud) were selected based on the survey on *continuous global optimization in R* by Mullen [2014] in which they performed best on a set of different optimization problems.

3.4.2 Surrogate-Model-Based Optimization

SMBO algorithms employ data-driven models to lighten the burden of expensive objective function evaluations. One framework for SMBO is sequential parameter optimization (SPO) [Bartz-Beielstein et al., 2005a]. SPO provides a flexible framework that employs methods from the design of experiments, optimization, and statistics. In essence, SPO starts by generating an initializing and evaluating a design, then builds a surrogate model. Then, the surrogate model is optimized to suggest a promising candidate solution, which is afterward evaluated by the expensive objective function. The steps model building, optimization, and evaluation are iterated until a selected evaluation budget is exhausted. Figure 3.1 shows the SMBO cycle for the underlying ESGC problem.

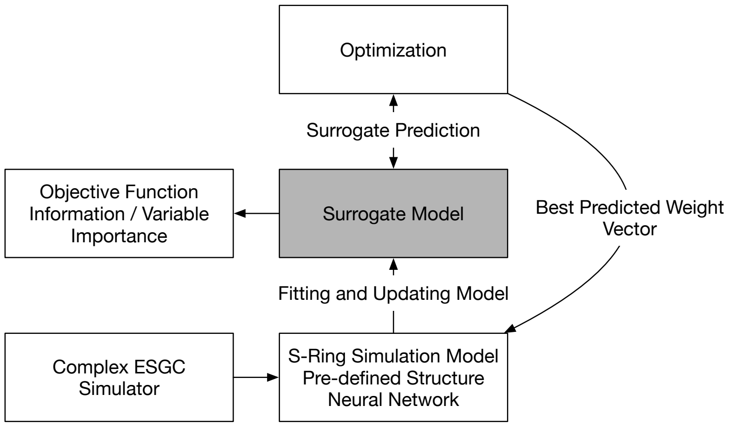

Optimization
Surrogate Prediction
Objective Function Information / Variable Importance
Surrogate Model
Best Predicted Weight Vector
Fitting and Updating Model
Complex ESGC Simulator
S-Ring Simulation Model Pre-defined Structure Neural Network

Figure 3.1: *SMBO Cycle. The S-Ring simulator approximates the ESGC Simulator. The fitness topology is fitted by the surrogate based on the initial design and sequential updates. Optimization of the surrogate computes the sequential weight vectors.*

For this study, we have chosen to investigate three different surrogate models within the SPO framework.

- **Second order model with stepwise regression:** Firstly, we build second-order linear regression models. The model is first built with all first-order effects, quadratic effects as well as second-order interactions. E.g., for two parameters x_1 and x_2 a model of the form $y(x) = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2$ is determined. This entire model is further refined by backward, stepwise variable selection based on the Akaike information criterion. The stepwise variable selection is skipped whenever the data size is insufficient. While the resulting models are comparatively simple, one advantage is the comparatively low computational effort.

- **Random Forest:** Secondly, we use a random forest [Breiman, 2001] model. Random Forests are ensembles of decision trees. We use the default settings of the randomForest R-package [Liaw et al., 2002]. Random forests can learn non-linear dependencies in the data, are typically numerically robust, fast to compute, and can handle discrete input variables.

– **Kriging:** Thirdly, a Kriging model (also known as Gaussian process regression) is employed. Kriging assumes that the observed data is the result of a stochastic process. We use an implementation from the R-package SPOT. The implementation is loosely based on Matlab code by Forrester et al. [2008]. The correlation of samples is modeled via an exponential correlation function $\mathrm{cor}(x, x') = \exp(-\sum_{i=1}^{n} \theta_i |x_i - x'_i|^{p_i})$. The vectors x and x' are samples, or candidate solutions of the optimization problem. The parameters $\theta_i > 0$ and $1 \leq p_i \leq 2$ are determined by maximum likelihood estimation. Forrester et al. [2008] provide a detailed and easy-to-follow description of Kriging and related methods. Kriging requires the most significant computational effort out of the three models yet produces the potentially most accurate model.

3.4.3 Variable importance

Most optimization algorithms deliver only the best-found parameter settings. Hence, these algorithms do not provide any information on what they have learned about the importance of the input variables during the optimization process. An advantage of the SMBO techniques is that the surrogate provides additional information beyond the best-found parameter setting. For example, the estimated model coefficients may serve as strong indicators for the importance of each input variable.

– **Linear Regression Models:** In linear regression models, the effect of each coefficient can be tested against the null hypothesis, resulting in a set of so-called p-values. However, a large p-value does not alone indicate a significant effect of the variable on the result. The values of the regression coefficients can be compared instead but must be standardized first to enable a valid comparison due to different scales. Larger (scale normalized) coefficient values account for a more considerable impact of the corresponding variable.

– **Random Forest:** Variable importance can be estimated by computing the mean decrease in model accuracy by excluding (or permutation) of a certain input variable. As a metric, commonly, the mean squared error (MSE) is utilized.

– **Kriging:** The kernel parameter θ_i determines how far the influence of each sample point spreads in dimension i. In detail, the larger the width parameter is, the faster are the potential changes in the predicted value. The smaller the width parameter is, the slower are the potential changes in the prediction. The descending order of the θ values gives an indicator of the variable importance.

3.5 Experiments

Our stated hypothesis is tested by performing empirical benchmarks with the S-Ring simulator. The performance of a *random search* algorithm is added as a baseline comparison. All four metaheuristic algorithms and the SMBO with the three different models are compared. The best-achieved performance (cumulated customer waiting time) after a fixed number of objective function evaluations is reported. The lower limit is set to 100 evaluations to simulate the optimization of a very expensive objective function. For the metaheuristics, the maximum number of evaluations is set to 1_{e+5} to see the convergence behavior. The maximum number of evaluations for SMBO is limited to 1000. SMBO utilizes ten percent of the available budget for the initial *latin hypercube sampling.* SMBO can become computationally expensive for large sample sizes due to model fitting, model optimization, and prediction. As all tested algorithms are stochastic, each experiment is repeated 20 times. For all tested algorithms, the parameter settings, besides the iterations and population size to set an exact number of evaluations, are not changed and use default settings. The experimental setup is summarized in Table 3.2.

Table 3.2: *Experimental Setup*

Algorithm	No. Evaluations	popSize	maxIter
RandomSearch	100	100	1
RandomSearch	1e+3, 1e+5	1e+3, 1e+5	1
Metaheuristics:			
GenSA	100, 200, 500	/	/
GenSA	1e+3, 1e+5	/	/
DEoptim	100, 200, 500	5,5,10	9,19,24
DEoptim	1e+3, 1e+5	10,100	49,499
GA	100, 200, 500	10,10,20	10,20,25
GA	1e+3, 1e+5	50,50	20,2e+3
genoud	100, 200, 500	10,10,20	10,20,25
genoud	1e+3, 1e+5	50,1e+3	20,100
Surrogates:			
Model	No. Evaluations	initDesign	Optimizer
SecondOrderLM	100, 200, 500, 1e+3	10,20,50,100	DEoptim
RandomForest	100, 200, 500, 1e+3	10,20,50,100	DEoptim
Kriging	100, 200, 500, 1e+3	10,20,50,100	DEoptim

3.6 Results and Discussion

3.6.1 Benchmark Comparison

Figure 3.2 shows the benchmark results for the different combinations of algorithms and the number of evaluations:

– **Metaheuristics:** For 100 evaluations, *GA* and *genoud* perform better than random search, while *DEoptim* and *GenSA* show inferior results. The methods significantly improve with a rising number of evaluations. For 1000 evaluations, most methods (except GenSA) perform better than the random search method, which is also the case for 1e+5 evaluations. The long-run results (1e+5) also indicate that *GenSA* and *DEoptim* seem to converge to a global optimum, while *genoud* and *GA* show significantly inferior results. A large number of evaluations seems to benefit *GenSA* the most. The performance of the metaheuristics is likely connected to their chosen default parameter settings.

– **SMBO:** For 100 evaluations, the SMBO outperforms all metaheuristics significantly. Particular *Kriging* performs on a level equivalent to this of 1000 random search evaluations and near 500 metaheuristic evaluations. This picture becomes even more apparent for 200 evaluations, where *Kriging* again improves and surpasses random search with 1000 evaluations. For 1000 evaluations, *Kriging* reaches the level of 1e+5 random search evaluations, again outperforming all metaheuristic algorithms on this level. The *second-order linear model* cannot show considerable improvements over the metaheuristics. *Random forest* shows poor results for large evaluation sizes, performing inferior to random search.

The good results of the *Kriging* surrogates can be explained by their strong ability to fit non-linear landscapes and their general good interpolation ability. While the *second-order linear model* can fit non-linear landscapes to a certain extent, they are not fit to build global surrogates of highly multi-modal landscapes. The poor performance of *random forest* is due to its inferior interpolation abilities given the continuous nature of the problem.

3.6.2 Model Variable Importance

To check the usefulness of the variable importance, we analyzed the importance values of the best models fitted with 1000 evaluations.

– The *second-order linear model* has a multiple R-squared of 0.869 and an adjusted R-squared of 0.8611. It contains many 60 model terms, including main effects, interactions, and quadratic effects, where the p-values indicate high importance. Due to this high number of terms, we regard a detailed importance analysis as less useful.

– The random forest model has a mean of squared residuals of 0.129 and explains 71.44 percent of the variance. The mean decrease in MSE is shown in Table 3.3 and compared to the results of *Kriging* (theta values).

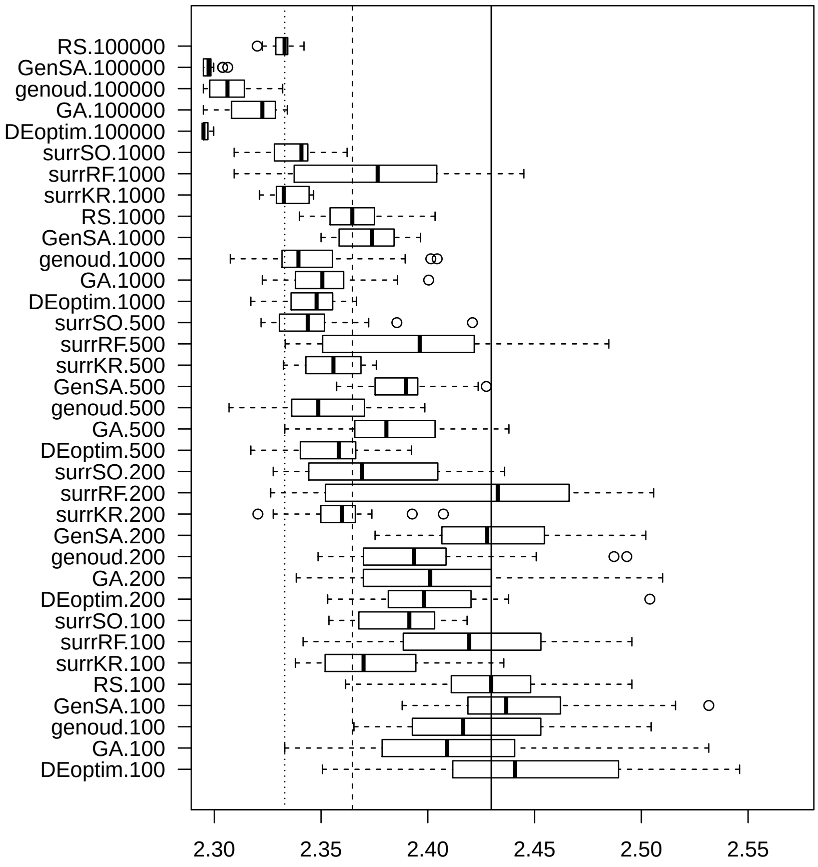


Figure 3.2: *S-Ring Simulator Benchmark Results. The algorithms with their respective number of evaluations are shown on the y-axis, the x-axis shows the achieved fitness (avg. customer waiting time). SurrSO uses the second-order model, SurrRF the random forest model, and SurrKR the Kriging model. The vertical lines represent the baseline, where the solid line is the median random search fitness for 100, the dashed line for 1000, and the pointed line for 100 000 evaluations.*

Table 3.3: *Variable importance.*

Weight	RF Mean MSE Decrease	Kriging theta values
1	117.31366	3.82746
2	9.83143	1e-04
3	59.79332	1.15356
4	17.86245	0.4643613
5	11.06656	0.2711343
6	14.05440	0.001178616
7	64.32869	2.221088
8	64.86968	1.164819
9	34.63966	1.8983
10	47.50249	3.45285

Table 3.3 indicates that at least the most crucial weight (No. 1) and the least essential weight (No. 2) are the same for both the *Kriging* and *random forest* model. The other weights also show some rank correlation. The variable importance comparison shows that the models can extract information beyond the best-found parameter setting. To validate the given results, we will need to use a designated experimental design, which will be part of future research.

3.6.3 Computation Time Comparison

An important aspect of every optimization technique is the total computation time. Table 3.4 shows approximated values for the metaheuristics and the SMBO with the respective models. As the problem itself has nearly no computation time, the indicated values are mainly caused by the optimization algorithms. As the values indicate, SMBO is rather expensive. The model fitting, updating, and optimization process are computationally demanding, particularly for a higher number of samples. This is especially visible for Kriging, which is sensitive to higher sample sizes.

Table 3.4: *Algorithm computation time. All values are approximated.*

Algorithm	No. Evaluations	Computation Time
S-Ring Problem C	1	< 0.001 seconds
S-Ring Problem R	1	< 1 second
Metaheuristics	100	0.1 second
Metaheuristics	1000	1 second
Metaheuristics	1_{e+5}	1-2 minutes
surrRF	100	1 minute
surrSO	100	4 minutes
surrKR	100	8 minutes
surrRF	1000	1 hour
surrSO	1000	4 hours
surrKR	1000	> 1 day

We also have to consider the*SPOT* R-framework implementation, which might be inferior to C or C++-based implementations. For instance, a (not optimized) re-implementation of the S-Ring simulator in R, which is implemented in C, is about 1000 times slower. We can assume that an optimized version of SMBO would be significantly faster. Furthermore, *SPO* performs only sequential optimization, while the metaheuristics can conduct evaluations in parallel.

3.7 Conclusion and Outlook

According to our hypotheses, the results show that SMBO is a beneficial approach for the underlying ANN control optimization task. The tested SMBO algorithm was capable of outperforming metaheuristic optimization methods in terms of sampling efficiency. Furthermore, the surrogate is helpful for the significance analysis of the inputs. We can thus assume that SMBO can provide a greater understanding of the learning process. The apparent downside of SMBO is the considerable computation time, which is more than 10000 times larger than these of the fastest metaheuristics. However, this vast downside becomes less significant in scenarios where the objective function evaluations become very expensive, e.g., in real-world optimization. The model fitting and optimization process could be conducted simultaneously with the real-time evaluations. Furthermore, parallel SMBO approaches could considerably improve the computation time. In this study, we used the default parameters for all given algorithms. An extended study to identify generally good settings for a large set of problems could further improve general performance.

4 Custom Distance Metrics for Surrogate Model-Based Optimization

Chapter 4 is based on two published papers:

Zaefferer M, Stork J, Friese M, Fischbach A, Naujoks B, & Bartz-Beielstein T (2014, July). Efficient global optimization for combinatorial problems. *In Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 871-878). ACM, New York.
and
Zaefferer M, Stork J, & Bartz-Beielstein T (2014, September). Distance measures for permutations in combinatorial efficient global optimization. *In International Conference on Parallel Problem Solving from Nature* (pp. 373-383). Springer, Cham.

4.1 Introduction

The optimization of costly objective functions, such as expensive simulations, is a frequently arising issue if real-world problems are considered. A well-established method to reduce the number of required evaluations is the employment of data-driven models in surrogate model-based optimization methods, such as efficient global optimization [Jones et al., 1998] or sequential parameter optimization [Bartz-Beielstein et al., 2005a]. While SMBO methods are frequently and successfully applied in continuous spaces, they are less prominent for combinatorial optimization problems. However, combinatorial problems are well present in several real-world settings, e.g., production process scheduling and machine sequencing [Voutchkov et al., 2005]. It is thus desirable to transfer successful SMBO approaches to combinatorial spaces.

A study by Moraglio and Kattan [2011a] provides a generalization of distance-based models to combinatorial spaces for radial basis function networks. Their core idea is to replace continuous distance measures, such as a Euclidean or Manhattan distance, with distance or similarity measures from combinatorial spaces, for example an edit distance or the Hamming distance.

We advance on these ideas by presenting research on
1) an extension of Kriging models and Kriging-based SMBO to allow custom distance measures, and
2) a performance comparison of different permutation distance measures in combinatorial optimization.

4.2 Surrogate Model-Based Optimization[1]

Generally speaking, a surrogate model $\widehat{\mathcal{M}}$ is a (coarse-grained or cheap) model that replaces a (fine-grained or expensive) model $\mathcal{M}$ with higher complexity. The reader may consider a computational fluid dynamics (CFD) model that replaces a real-world problem. A simplified analytical model can replace this CFD model itself. CFD models are considered surrogates in the former case,

[1]Based on [Zaefferer et al., 2014b] Section 2

whereas, in the latter, they are considered fine-grained models that are replaced by a surrogate. However, we concentrate our work on data-driven Kriging models. They replace the simulation model $\mathcal{M}$, which is given by a function f, see Algorithm 4.2.1. Here, it is assumed that function evaluations dominate the time consumption (or cost), i.e., most time is spent in lines two and six of Algorithm 4.2.1. Stopping criteria can be a given budget of function evaluations, a specified time limit, or a fitness value to be reached. The set A_p contains all underlying parameters of Algorithm 4.2.1, e.g., number of initial solutions, type, and parameterization of the search strategy (line 5) or the type of model.

Algorithm 4.2.1: Surrogate model-based optimization

```
  Input: Function $f$, stopping criteria, parameter set $A_p$
  Output: Best solution found $y^*$, final model $\widehat{\mathcal{M}}^*$
1 Create initial solutions (randomly or with design of experiment);
2 Evaluate solutions with $f$;
3 while Stopping criteria not reached do
4     Build/update $\widehat{\mathcal{M}}$;
5     Find best solution(s) predicted by $\widehat{\mathcal{M}}$;
6     Evaluate solution(s) with $f$;
7 end
```

Jones et al. [1998] introduced EGO, which uses the predicted mean and variance provided by a Kriging surrogate to compute the expected improvement (EI) of a candidate solution. Without loss of generality, we will consider the case of minimization. Following the notation in [Forrester et al., 2008], we consider an expensive function f and a related surrogate of f, denoted $\hat{f}$. Gaussian process models allow the determination of the mean squared error $\hat{s}^2(\mathbf{x})$ as described in [Sacks et al., 1989]. Let $y_{\min}$ denote the best found function evaluation so far, $\Phi(\cdot)$ and $\phi(\cdot)$ denote the cumulative distribution function and probability density function, respectively. If $\hat{s}(\mathbf{x}) > 0$, then the expected improvement can be determined as

$$\mathrm{EI}(\mathbf{x}) = (y_{\min} - \hat{f}(\mathbf{x}))\Phi\left(\frac{y_{\min} - \hat{f}(\mathbf{x})}{\hat{s}(\mathbf{x})}\right) + \hat{s}\phi\left(\frac{y_{\min} - \hat{f}(\mathbf{x})}{\hat{s}(\mathbf{x})}\right),$$

otherwise $\text{EI}(\mathbf{x}) = 0$. EI determines how much improvement can be expected from the candidate solution to be predicted. Thus, EGO uses EI instead of the mean prediction to determine a promising candidate solutions. Besides saving evaluations of the expensive function f, this approach also provides an infill criterion (i.e., EI) that balances exploitation versus exploration. Our SMBO variant for combinatorial optimization adapts this principle.

4.3 Kriging for Combinatorial Problems[2]

Li et al. [2008] proposed radial basis function network (RBFN) models for optimization in non-Euclidean spaces by replacing the employed distance measure. Their RBFN models were applied to mixed-integer problems using a mixed-integer evolution strategy. Another approach to a mixed problem is taken by Hutter [2009], who describes a Kriging model based on a weighted Hamming distance to model categorical variables for algorithm tuning. In a very similar way, Moraglio and Kattan [2011a] suggested a generalization of distance-based models from continuous to combinatorial spaces.

The core idea of combinatorial Kriging to employ distance measures, which are inherent to the combinatorial problem representation (e.g., edit distance). Such problem representations can be binary strings (e.g., binary knapsack, NK-Landscapes) permutations (e.g., assignment and scheduling problems), trees (e.g., symbolic regression), or any non-standard combinatorial representation.

The critical issue is to replace the Euclidean or per-variable distances by distance measures, which directly work for the inherent problem representation. Depending on the model type under consideration, other changes may become necessary. For instance, there is no guarantee that a given distance matrix will be invertible in the context of arbitrary distance measures, as required by RBFN. Therefore, Moraglio and Kattan [2011a] suggested replacing the matrix inversion with the pseudoinverse.

Kriging is a method for interpolation and regression based on Gaussian process modeling. The following notation is adopted from Forrester et al. [2008].

[2]Based on [Zaefferer et al., 2014b] Sections 2 and 3

Given a set of n solutions $\boldsymbol{X} = \{\mathbf{x}^{(i)}\}_{i=1\ldots n}$ in a k-dimensional continuous search space with observations $\mathbf{y} = \{y^{(i)}\}_{i=1\ldots n}$, Kriging is a method to find an expression for a predicted value at an unknown point by interpreting the observed responses $\mathbf{y}$ as if they are realizations of a stochastic process. The following set of random vectors $\boldsymbol{Y} = \{Y(\mathbf{x}^{(i)})\}_{i=1\ldots n}$ is used to define this stochastic process. The random variables $Y(\cdot)$ are correlated as follows [Forrester et al., 2008]:

$$\operatorname{cor}\left[Y(\mathbf{x}^{(i)}), Y(\mathbf{x}^{(l)})\right] = \exp\left(-\sum_{j=1}^{k} \theta_j |x_j^{(i)} - x_j^{(l)}|^{p_j}\right). \tag{4.1}$$

Equation (4.1) defines a non-Euclidean distance measure, which uses a weighted per-element distance. The weights θ_j and the shape parameter p_j have to be estimated. The matrix that collects correlations of all pairs $\{(i,l)\}$ is called the correlation matrix $\boldsymbol{\Psi}$. It is used in the Kriging predictor

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \boldsymbol{\psi}^T \boldsymbol{\Psi}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu}), \tag{4.2}$$

where $\hat{y}(\mathbf{x})$ is the predicted function value of a new sample $\mathbf{x}$, $\hat{\mu}$ is the maximum likelihood estimate of the mean and $\boldsymbol{\psi}$ is the vector of correlations between training samples $\boldsymbol{X}$ and the new sample $\mathbf{x}$. The error of the prediction can be estimated with

$$\hat{s}^2(\mathbf{x}) = \hat{\sigma}^2(1 - \boldsymbol{\psi}^T \boldsymbol{\Psi}^{-1} \boldsymbol{\psi}^T), \tag{4.3}$$

where $\hat{\sigma}^2$ is a model parameter to be estimated. The (usually small) contribution of error due to estimation of $\hat{\mu}$ is omitted.

The width parameter $\boldsymbol{\theta}$ determines how far the influence of each sample point $\mathbf{x}$ spreads. If the correlation structure differs in different directions of the search space, fitting different θ_j values for each direction of the search space is desirable. This is the so-called anisotropic case. Isotropic models are better suited for combinatorial search spaces because direction is a vague concept for combinatorial optimization problems. Therefore, Eq. (4.1) is transformed to become isotropic, i.e., with scalar θ and p, i.e.:

$$\operatorname{cor}\left[Y(\mathbf{x}^{(i)}), Y(\mathbf{x}^{(l)})\right] = \exp(-\theta \mathrm{d}(\mathbf{x}^{(i)}, \mathbf{x}^{(l)})^p), \tag{4.4}$$

where $\mathrm{d}(\cdot)$ can be any distance measure for the given problem representation. Now, the samples $\mathbf{x}$ are not restricted to continuous values and may consist of various types, e.g., binary strings, permutations, or trees.

Maximum likelihood estimation (MLE), which comprehends an optimization procedure, is used to determine the model parameters, i.e., θ, p, $\hat{\sigma}$ and $\hat{\mu}$. MLE requires a matrix inversion (also later in the prediction step, see (4.2)), usually performed directly or via Cholesky decomposition. A non degenerated or positive-semidefinite matrix is required for this inversion. We employ the more stable inversion via Cholesky decomposition, which requires a positive semi-definite correlation matrix.

4.4 Distance Measures for Permutation[3]

The choice of distance measure can also be understood as a (categorical) parameter of the model. Hence, we suggest performing MLE for each distance measure separately. Afterward, the distance measure with maximum likelihood is chosen for the model. This procedure repeats every time the model is built, i.e., in each iteration of a single SMBO run. In the experimental study, this will be referred to as *"All"*.

A wrong decision may occur, especially while data is still very sparse. Therefore, we expect the performance of choosing a distance measure with MLE to be equal to or worse than the best single measure. An exception would be the case where the underlying optimization problem has a dynamic behavior. Then, different measures may be preferable in different phases of the optimization run.

Distance measures for permutations were investigated in other domains, such as for landscape analysis [Schiavinotto and Stützle, 2007] or diversity preservation [Sevaux and Sörensen, 2005]. These previous studies illustrate that

[3]Based on [Zaefferer et al., 2014a] Section 3

Table 4.1: *Investigated distance measures. Second column lists runtime complexity. Third column lists median runtime of 1000 evaluations for permutations of length 30.*

Name	complexity	runtime [μs]	Abbrev.
Levenshtein	$O(n^2)$	7	Lev
Swap	$O(n^2)$	6	Swa.
Interchange	$O(n^2)$	14	Int.
Longest Common Subsequence	$O(n^2)$	8	LCSeq
Longest Common Substring	$O(n^2)$	8	LCStr
R	$O(n^2)$	5	R
Adjacency	$O(n^2)$	6	Adj.
Position	$O(n^2)$	6	Pos.
Position^2	$O(n^2)$	6	Posq.
Hamming	$O(n)$	2	Ham.
Euclidean	$O(n)$	6	Euc.
Manhattan	$O(n)$	4	Man.
Chebyshev	$O(n)$	3	Che.
Lee	$O(n)$	6	Lee

a large array of distance measures is available. In this study, we will analyze 14 different distance measures, as summarized in Table 4.1. The given runtime complexity refers to the employed implementations. More efficient variants may be available. All distance measures are scaled to yield values from $[0;1]$ to avoid scaling bias. In the following, we describe the basic features of these distance measures. Since the naming of measures in the literature varies, this clarification is useful to avoid confusion.

– **Levenshtein** and edit distance are sometimes used as synonyms. Levenshtein is only one example of an edit distance. It counts the minimum number of deletions, insertions, or substitutions required to transform one string (or here: permutation) into another.
– A **swap** operation is the transposition of two adjacent elements in a permutation. The swap distance is defined as the minimum number of swaps required to transform one permutation into another. It has also been called Precedence distance [Schiavinotto and Stützle, 2007], or Kendall's Tau [Kendall and Gibbons, 1990; Sevaux and Sörensen, 2005].
– An **interchange** operation is the transposition of two arbitrary elements.

Respectively, the interchange (also: Cayley) distance is the minimum number of interchanges required to transform one permutation to another [Schiavinotto and Stützle, 2007].

– The **longest common subsequence** distance counts the largest number of elements that follow each other in both permutations, with interruptions. We use the algorithm described in [Hirschberg, 1975].
– The **longest common-substring** distance counts the largest number of elements that follow each other in both permutations, without interruption, i.e., all elements are adjacent.
– The **R-distance** [Campos et al., 2005; Sevaux and Sörensen, 2005] counts the number of times that one element follows another in one permutation, but not in the other. It is identical with the uni-directional adjacency distance [Reeves, 1999].
– The (bi-directional) **adjacency** distance [Reeves, 1999; Schiavinotto and Stützle, 2007] counts the number of times two elements are neighbors in one, but not in the other permutation. Unlike R-distance (uni-directional), the order of the two elements does not matter.
– The **position** distance [Schiavinotto and Stützle, 2007] is identical with the deviation distance or Spearman's footrule [Sevaux and Sörensen, 2005]
– The **squared position** distance is Spearman's rank correlation coefficient [Sevaux and Sörensen, 2005].
– The **Hamming** distance or Exact Match distance counts the number of unequal elements in two permutations.
– The **Euclidean** distance is $\delta_{Euc.}(\pi, \pi') = \sqrt{\sum_{i=1}^{n}(\pi_i - \pi'_i)^2}$.
– The **Manhattan** distance is $\delta_{Man.}(\pi, \pi') = \sum_{i=1}^{n} |\pi_i - \pi'_i|$.
– The **Chebyshev** distance is $\delta_{Che.}(\pi, \pi') = \max\limits_{1 \leq i \leq n} (|\pi_i - \pi'_i|)$.
– The **Lee** distance [Lee, 1958] can be adapted to permutations with $\delta_{Lee}(\pi, \pi') = \sum_{i=1}^{n} min(|\pi_i - \pi'_i|, n - |\pi_i - \pi'_i|)$.

The reversal distance (number of reversals required to transform one permutation to another) was not used, even though it is especially promising for the traveling salesperson problem (TSP). Calculating the reversal distance for unsigned permutations is NP-hard [Caprara, 1997].

4.5 Experiments and Results[4]

For all further experiments, five permutation problem classes are used.

– Four instances of the quadratic assignment problem (QAP) [Burkard, 1984] from the QAPLIB [Burkard et al., 1997] are chosen (nug30, nug12, tho30 and kra32). In the QAP, n facilities have to be assigned to n locations. Assignment cost is minimized based on the flow between facilities and the distance between locations.
– Four instances of the flow-shop scheduling problem (FSP) [Taillard, 1990] are chosen (reC05, reC13, reC19, reC31 [Reeves, 1995]) from the OR-Library [Beasley, 1990]. Here, the finishing time of the last of n jobs sequenced on m machines is minimized.
– Three TSP instances are chosen from the TSPLIB [Reinelt, 1991] (bayg29, fri26, gr24). In the TSP, the cost or length of a route through several locations is minimized. Each location has to be visited once.
– Three instances of the asymmetric TSP (ATSP) are generated (atsp10, atsp20, atsp30). For each instance, a distance matrix is created randomly with a uniform distribution. The three instances are of sizes 10, 20, and 30. In contrast to TSP, the cost of traveling between two locations depends on the direction.
– Finally, four instances of the single-machine total weighted tardiness problem (WT) [Abdul-Razaq et al., 1990] are chosen, also from the OR-Library [Beasley, 1990] (the first four of length 40, i.e., wt40a, wt40b, wt40c, wt40d). Here, n 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 n given weights, is minimized. It depends on the given processing times and due dates of each job.

For QAP, TSP, ATSP, and WT, the length of the permutation n are given by the number in the instance name. For FSP, n is 20, 20, 30 and 50 for reC05, reC13, reC19 and reC31 respectively.

We use this benchmark set under the artificial assumption of costly target

[4]Based on [Zaefferer et al., 2014a] Section 4 and 5

function evaluation. While some of these problems have actual real-world relevance (e.g., based on real-world data), none may be considered expensive. This allows for a more in-depth study, providing first results, which of course should be validated with actually expensive problems in future studies.

We compare the optimization performance of SMBO, a model-free genetic algorithm (GA), random search (RS), and a simple 2-opt local search are employed with a strictly limited budget of 200 function evaluations. GA, RS, and 2-opt are baselines in this comparison.

The GA uses cycle crossover, and the mutation operator is an interchange of arbitrary elements. Furthermore, the algorithm employs a population size of ten, crossover rate 0.5, mutation rate $1/n$, tournament selection with size two, and probability 0.9. The SMBO algorithm will start with an initial set of ten solutions. Kriging parameter p is set to one, while the others are determined with MLE. Internally, SMBO will perform optimization of the (assumed to be cheap) surrogate model. Hence, the same GA is used with 10,000 model evaluations and population size of 20.

For an ideal comparison of actual competitors, most of the mentioned parameters would require tuning. The SMBO variants use identical settings, thus yielding a fair comparison among themselves.

Figure 4.1 shows the results of the optimization experiments. The SMBO variants are referred to by the name of the employed distance measure. Results of 2-opt are not shown for the sake of brevity. 2-opt usually ranks worse than GA and only outperforms GA for the three TSP instances. Still, it can not compete with the model-based approaches. Chebyshev distance (Che.) and RS are consistently outperformed by the GA and not included in the plot.

Three main groups with similar structures can be identified: first, the QAP instances, second, the TSP and ATSP instances, and third, the WT and FSP instances. Members of each group have a similar pattern, although the best performing method may not be identical for all members. Overall, the model-free GA is consistently outperformed by at least five SMBO variants. Choosing the wrong distance measure may, however, lead to perform worse than the model-free GA. Choosing a distance measure with MLE (*All*) never ranks worse

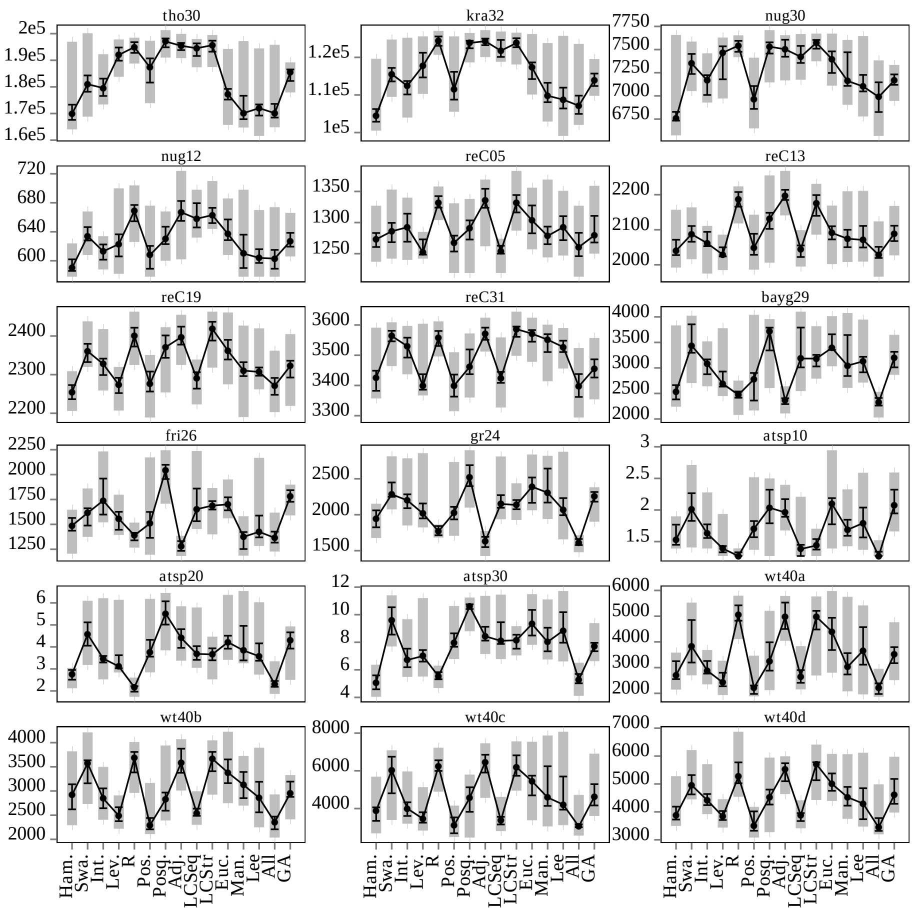


Figure 4.1: *Optimization performance: Dots are median, black bars are interquartile range, thick grey bars are range from minimum to maximum. Smaller values are better.*

than third best, making it the most robust method in this testbed. *All* ranks first place in 7 of 18 instances. It seems that the underlying problem does profit from a dynamic choice of distance measure. The single best distance measure is Hamming distance, yielding best results in 6 of the 18 test problems but receiving lower ranks for other instances. For each problem class, the single best distance measures are Ham for QAP, Lev for FSP, Adj for TSP, R for ATSP, and Pos for WT. While the scheduling problems instead reflect the importance of relative order, TSP or ATSP are more concerned with the adjacency of neighboring cities. Hence, it makes sense that a bi-directional adjacency measure is used for TSP, while uni-directional adjacency (R-Distance) is used for the ATSP instances. In ATSP, direction matters, whereas in TSP, it does not.

4.6 Conclusion and Outlook

We demonstrated that SMBO could be successfully applied to combinatorial optimization problems, and that this Kriging-based approach was able to outperform a model-free GA. However, finding the global optimum solution of the EI landscape requires a surrogate-optimizer that can search multi-modal landscapes. Stochastic, population-based methods like GA are most suitable for this purpose and are also employed in our SMBO approach.

Furthermore, the permutation distance measures were revealed to have a substantial impact on the results. It was shown that each permutation problem class or instance might require a different distance measure. The excellent and robust performance of choosing a distance measure with MLE makes for a promising result. Here, the only issue is to carefully avoid numerical problems, i.e., to use matrix inversion via Cholesky decomposition. Should the increased computational effort necessitate a smaller set of distance measures, Hamming distance should always be included due to good performance and lowest cost.

5
Comparison of Distance Metrics for Surrogate Model-Based Optimization in Genetic Programming

Chapter 5 was published as:

Zaefferer M, Stork J, Flasch O, & Bartz-Beielstein T (2018, September). Linear combination of distance measures for surrogate models in genetic programming. *In International Conference on Parallel Problem Solving from Nature* (pp. 220-231). Springer, Cham.

Surrogate models are a well-established approach to reduce the number of expensive function evaluations in continuous optimization. In the context of genetic programming, surrogate modeling still poses a challenge due to the complex genotype-phenotype-fitness relationships. We investigate how different genotypic and phenotypic distance measures can be used to learn Kriging models as surrogates. We compare the measures and suggest using their linear combination in a kernel.

We test the resulting model in an optimization framework, using symbolic regression problem instances as a benchmark. Our experiments show that the model provides valuable information. Firstly, the model enables an improved optimization performance compared to a model-free algorithm. Furthermore, the model provides information on the contribution of different distance measures. The data indicates that a phenotypic distance measure is vital during the early stages of an optimization run when less data is available. In contrast, genotypic measures, such as the tree edit distance, contribute more during the later stages.

5.1 Introduction

Genetic programming (GP) automatically evolves computer programs that aim to solve a task. This idea goes back to fundamental work by Koza [1992] and follows the principles of evolutionary computation. Computer programs are individuals subject to an evolutionary process, which improves them based on their fitness, i.e., their ability to solve a problem. Examples for GP tasks are symbolic regression (SR), classification, and production scheduling [Flasch, 2015; Nguyen et al., 2017].

Expensive fitness functions pose a challenge to evolutionary algorithms, including GP. This occurs, e.g., when the fitness function requires laboratory experiments or extensive simulations. Frequently, surrogate model-based optimization is used to deal with expensive evaluations [Bartz-Beielstein and Zaefferer, 2017]. Most SMBO research focuses on problems with continuous variables, where many competitive regression models are available. In the context of GP, the use of surrogates is not well researched. This might seem surprising, as the computational bottleneck of most GP applications is the evaluation of fitness cases. Unfortunately, surrogate modeling of GP tasks, such as SR, is intricate because it subsumes modeling a complex genotype-phenotype-fitness mapping. Recent work in deep learning suggests that this mapping can be approximated, at least in specific domains of program synthesis [Parisotto et al., 2016]. In the last years, combinatorial search spaces were treated successfully with SMBO by using distance-based models [Moraglio and Kattan, 2011a; Zaefferer et al., 2014b]. However, there is no generic way to choose an adequate distance measure. For complex tree-shaped structures, which occur in GP, it is challenging to select a suitable distance measure and find a feasible modeling approach. For that reason, we will focus on the following research questions regarding SMBO for GP and tree-shaped structures:

1. How do different distance measures compare to each other?
2. What impact do these distances have on the model?
3. How does SMBO based on a linear combination of these distances compare to a model-free evolutionary algorithm and random search?

To answer these questions, we will utilize bi-level optimization problems based on different SR tasks as test functions. 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. They allow us to gain insights into the topics summarized by our research questions. We expect our result to be transferred to other problems with tree-shaped structures, such as program synthesis for general purpose or domain-specific languages.

5.2 Related Work

In the following, we will differentiate between two approaches, which will be further referred to as a) SMBO and b) SAEA.

a) Sequential SMBO generates new candidate solutions by performing a search procedure on the surrogate model, e.g., as described for the efficient global optimization algorithm by Jones et al. [1998].

b) Approaches utilize surrogates to assist an EA (SAEA), e.g., as described by Jin [2011]. For example, the surrogate is utilized to support the selection process of an EA by predicting the fitness of the proposed offspring.

Most studies on GP and surrogate modeling focus on SAEA. Kattan and Ong [2015] describe an SAEA approach with two distinct radial basis function network (RBFN) models (semantic and fitness). The conjunction of both models is used to evolve a subset of the population. They report the superiority of their approach over standard GP for three different tasks, including SR.

Hildebrandt and Branke [2015] present a phenotypic distance. They optimize job dispatching rules with an SAEA approach. Their surrogate model is a nearest neighbor regression model based on the phenotypic distance. They demonstrate that their model allows for faster evolution of reasonable solutions. This approach is also discussed and extended by Nguyen et al. [2014, 2016].

To the best of our knowledge, only Moraglio and Kattan [2011b] describe an SMBO approach to GP where a minimal number of function evaluations is allowed. They use an RBFN with appropriate distance measures. 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 [Jones et al., 1998]) and employ them in an SMBO framework with a severely limited number of 100 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. Although SR is not particularly expensive, we argue that it presents a complex and challenging test case to investigate whether our proposed models can learn such a complex search landscape.

5.3 A Test Case for SMBO-GP: Bi-level Symbolic Regression

In SR, a regression task is solved by evolving symbolic expressions. In essence, SR searches for a formula that best represents a given data set. Trees can represent the formulas. Each tree consists of nodes and leaves and the discrete labels on the nodes (mathematical operators, e.g., $+, -, *, /$) and leaves (variables and real-valued constants). Figure 5.1 shows the tree structure of the symbolic expression $\sqrt{c_1 - z_2} + (z_1 c_2)$. Our goal is to develop models that learn the relation between discrete tree structures and their fitness. For now, we are not interested in the influence of the real-valued constants. Hence, we suggest a bi-level problem definition.

5.3.1 Problem Definition

The upper level is 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

$$\min_x F(x, c) \qquad \text{subject to} \quad c \in \arg\min_c f(x, c),$$

where x is the tree structure representation, $c \in \mathbb{R}^d$ is the set of d_c constant values, and $f(x, c)$ is the lower level objective function. Note, that the number of constants d_c depends on x. In extreme cases, the tree x may not contain any constants ($d_c = 0$), which eliminates the lower level problem. The fitness will be determined as

$$f(x, c) = 1 - |\text{cor}(\hat{y}(x, c), y)|, \tag{5.1}$$

where $\hat{y}(x, c)$ denotes the output of the symbolic expression for the data set, y is the corresponding vector of true observations, and $\text{cor}(\cdot, \cdot)$ is the Pearson correlation coefficient. If $\hat{y}(x, c)$ becomes infeasible (e.g., due to a negative square root or division by zero), we assign a penalty value. To that end, we use the upper bound of our fitness function, $f_{\text{penalty}}(x, c) = 1$. An example of an upper-level candidate's evaluation is visualized in Figure 5.1. If not stated otherwise, fitness evaluations refer to evaluations of the upper level function F.

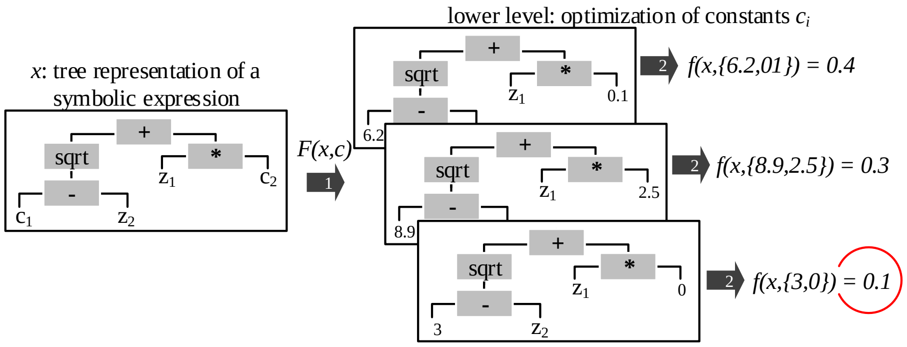


Figure 5.1: *Example for the upper level candidate $x = \sqrt{c_1 - z_2} + (z_1 c_2)$. To estimate its fitness $F(x, c)$, a lower level optimizer (step 1) estimates the fitness $f(x, c)$ for different constants (step 2) and returns the best to $F(x, c)$ (red circle).*

5.3.2 Surrogate Model-Based Optimization

The SMBO approach we employ for the upper-level optimization is loosely based on the EGO algorithm . Initially, the search space is randomly sampled. The resulting data is used to learn a suitable regression model. This surrogate model is subject to a search via an optimization algorithm (e.g., an EA), optimizing an infill criterion based on the model. An iteration ends with evaluating the actual (upper level) fitness of the new individual. Then, the surrogate model is updated with the new data, and the procedure iterates.

As in standard EGO, we utilize a Kriging regression model, which assumes that the observed data is derived from a Gaussian process [Forrester et al., 2008]. One reason for the popularity of Kriging in SMBO is that it allows estimating its uncertainty. The uncertainty estimate can be used to calculate the expected improvement infill criterion, which allows for balancing exploitation and exploration in an optimization process [Mockus, 1974; Jones et al., 1998].

Importantly, Kriging is based on correlation measures or kernels, which describe the similarity of samples. Exponential kernels, e.g., $\mathrm{k}(x, x') = \exp(-\theta||x - x'||_2)$, with the parameter θ determined by maximum likelihood estimation, are often used. It is straightforward to extend kernel-based models to combinatorial search spaces [Moraglio and Kattan, 2011a; Zaefferer et al., 2014b]. The core idea is to replace the distance measure, e.g., in the exponential kernel $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x'))$. The distance measure $\mathrm{d}(x, x')$ can be some adequate measure of distance between candidate solutions, such as an edit distance. Our study follows this idea. We will compare different distance measures and test how much they can contribute to Kriging models in an SMBO algorithm.

5.4 Kernels for Bi-level Symbolic Regression

We investigate four distance measures between trees or symbolic expressions that will be embedded into an exponential kernel.

5.4.1 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. Hildebrandt and Branke [2015] have suggested this idea for evolving dispatching rules via GP. They defined a phenotypic dissimilarity by comparing the outcome of a decision rule 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 $\hat{y}(x, \mathbf{1})$ with

$$\mathrm{d}_{\mathrm{PHD}}(x, x') = 1 - |\mathrm{cor}(\hat{y}(x, \mathbf{1}), \hat{y}(x', \mathbf{1}))|.$$

If either of the two expressions is infeasible (e.g., due to division by zero), the distance will be set to one. Setting all constants 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 a different symbolic expression behavior.

Instead of this definition (or in addition to it), we could also limit the number of training data samples used to evaluate the expression, which would be closer to the approach of Hildebrandt and Branke [2015]. For the test cases used in this study, the cost reduction would be negligible.

5.4.2 Tree Edit Distance

As an alternative to the PHD, we will also employ genotypic distances, i.e., distances between trees. 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 [2016b]. It is available in the APTED library version 0.1.1 [Pawlik and Augsten, 2016a]. The APTED implementation counts the following edit operations: node deletion, node insertion, and node relabeling. Alternatively to counts, costs can be defined for each operation, and the TED is then defined as the minimum cost sequence, e.g., to give different weights to different operations. We use equal weights in our study.

5.4.3 Structural Hamming Distance

The structural hamming distance (SHD) [Moraglio and Poli, 2005] has been used to express genotypic dissimilarity for model-based GP in several studies [Moraglio and Kattan, 2011b; Kattan and Ong, 2015; Hildebrandt and Branke, 2015]. Roughly speaking, it compares two trees by recursively checking each node that the two trees have in common. It uses the hamming distance (HD) to compare nodes, which has a value of one if two labels are different and zero otherwise. The original SHD (SHD1) is defined as

$$\mathrm{d}_{\mathrm{SHD1}}(x,x') = \begin{cases} 1, & \text{if } \mathrm{arity}(x_0) \neq \mathrm{arity}(x'_0) \\ \mathrm{HD}(x_0,x'_0), & \text{if } \mathrm{arity}(x_0) = \mathrm{arity}(x'_0) = 0 \\ \Delta(x,x'), & \text{if } \mathrm{arity}(x_0) = \mathrm{arity}(x'_0) = m, \end{cases}$$

with

$$\Delta(x,x') = \frac{1}{m+1}\left(\mathrm{HD}(x_0,x'_0) + \sum_{i=1}^{m} \mathrm{d}_{\mathrm{SHD1}}(x_i,x'_i)\right). \tag{5.2}$$

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 x, and arity(x_0) implies the number of subtrees linked to the corresponding node. We use a slight variation, which we refer to as SHD2. For the sake of simplicity, we define it for trees with a maximum arity of two. SHD1 and SHD2 are identical, except for the case $\mathrm{arity}(x_0) = \mathrm{arity}(x'_0) = m > 1$. Then, Eq. (5.2) becomes

$$\begin{aligned}\Delta(x,x') =& \frac{1}{m+1}\Big(\mathrm{HD}(x_0,x'_0) + \\ & \min\left\{\mathrm{d}_{\mathrm{SHD2}}(x_1,x'_1) + \mathrm{d}_{\mathrm{SHD2}}(x_2,x'_2), \mathrm{d}_{\mathrm{SHD2}}(x_1,x'_2) + \mathrm{d}_{\mathrm{SHD2}}(x_2,x'_1)\right\}\Big).\end{aligned}$$

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' which yields the smaller distance. Potentially, this is more accurate since it does not depend on the (arbitrary) initial alignment of the two trees. However, SHD2 requires additional computational effort, even more so for larger arities.

The reason for using this modified variant lies in the nature of our SMBO algorithm. SAEAs yield datasets where some individuals will have common ancestors (or are ancestors of each other), and hence, are inherently more likely to be aligned with each other. Contrarily, SMBO generates new trees via a randomly initialized search that avoids direct ancestor relationships among individuals. This implies that two trees are more likely to have different alignments. Then, SHD2 is a potentially more accurate (but costly) measure.

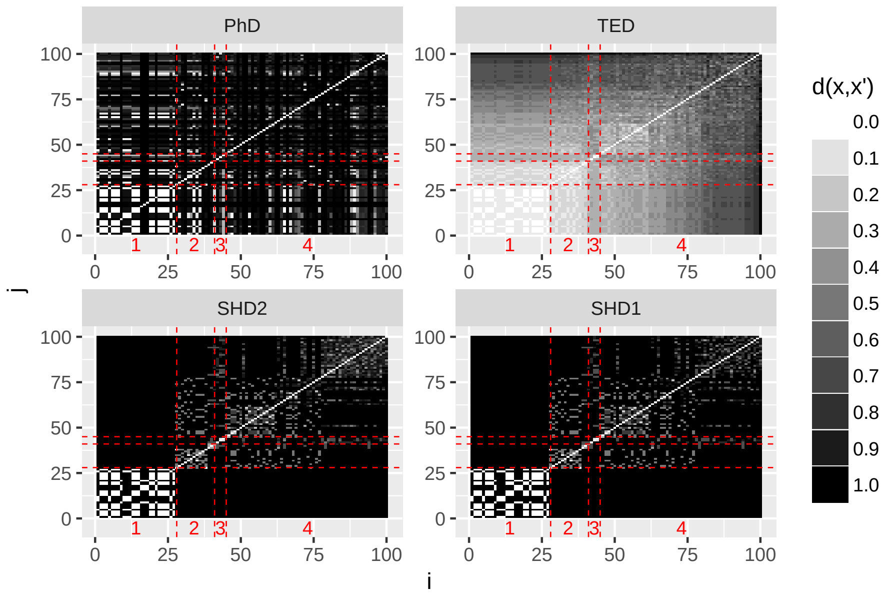


Figure 5.2: *Image plot of the four different tree distance measures. Each image cell is an element of a distance matrix. The trees are sorted by their complexity (tree depth and the number of nodes). Trees in the lower-left corner are less complex than those in the upper right. The tree depth is annotated in red at the bottom of each plot.*

5.4.4 Comparison and Linear Combination of Distances

To compare the four different distance measures, we first calculated the distance matrices for 100 randomly generated trees (symbolic expressions). We used the same random tree-generation method as in Section 5.5. We computed the Pearson correlation between the different distance matrices. For this sample, the SHD variants yielded a strong correlation of 0.99, which indicates that they reflect very similar information. For the remaining samples, the correlation was 0.51 (PHD, SHD2), 0.29 (PHD, TED), and 0.37 (TED, SHD2). That is, the largest diversity was observed between PHD and TED.

Figure 5.2 visualizes the corresponding distance matrices. It shows that the SHD does have problems with differentiating between trees of different complexity. 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 a complexity that has not been observed so far. TED and PHD tend to see larger distances for more complex trees. This is obvious for TED, as complex trees require more operations to be transformed into each other. For PHD it is clear that complex trees can produce more diverse phenotypic behavior.

With regards to the computational effort, we note that TED is by far the most expensive measure. The PHD follows it, 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 others. This is not surprising, as determining the minimal number of edit operations requires solving an optimization problem.

The PHD measure seems most promising in terms of generalizability. Most GP problems involve some phenotypic behavior that may be measured/compared. SHD and TED are limited to problems with tree structures and discrete labels.

The diversity of the different distances suggests that it is promising to combine them. We propose a linear combination of the PHD, TED, and SHD2. 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. Also, its increased computational cost disappears compared to the more considerable costs of the TED. The linear combination in the kernel is

$$\mathrm{k}(x, x') = \exp\left\{-\beta_1 \mathrm{d}_{\mathrm{SHD2}}(x, x') - \beta_2 \mathrm{d}_{\mathrm{PHD}}(x, x') - \beta_3 \mathrm{d}_{\mathrm{TED}}(x, x')\right\}. \quad (5.3)$$

Each distance receives a weight $\beta_i \in \mathbb{R}^+$ that is determined by MLE. The linear combination allows for a potentially more accurate Kriging model. As we do not know a-priori which distance measure is appropriate for a specific problem (or whether they complement each other), the combination shifts this decision problem to the model. Furthermore, the weights provide insights into when and how much each distance contributes to the model.

5.5 Case Study

We performed a case study, testing the SMBO algorithm with six SR tasks.

Symbolic Regression Test Problems: We chose the Newton, sine-cosine, Kotanchek2D, and Salustowicz1D problems as used in [Flasch, 2015] and the sqr and sqr+log problem as used in [Kattan and Ong, 2015]. All problem configurations remained unchanged, i.e., operator set, data set size, and bounds for variables. We did not evaluate the derived symbolic expressions on an additional test set since our goal was to determine the ability of the SMBO algorithm to learn the connection between candidate solutions and fitness.

Lower level optimization of the constants: To optimize the lower level objective function, we decided to use the locally biased version of the dividing rectangles (DIRECT) algorithm [Gablonsky and Kelley, 2001] for a global search. DIRECT uses $1000 \times d_c$ evaluations of the objective function. The result of the DIRECT run is further refined with a Nelder-Mead local search [Nelder and Mead, 1965] (also $1000 \times d_c$ 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 and a model-free EA as baselines. All operators were taken from the `rgp` package [Flasch et al., 2014]. For creating new individuals, both baselines used `randfuncRampedHalfAndHalf`, parameterized with a maximum tree depth of 4 and a probability to generate constants of 0.2. Furthermore, the EA employed `crossoverexprFast` for recombination, which randomly exchanges subtrees. For mutation, `mutateSubtreeFast` was used. The parameters of the mutation operator are 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 values were not considered at the upper level, the respective bounds in the operator are both set to one. We employed a standard EA (based on `optimEA` in the `CEGO` package [Zaefferer, 2017]) that used the above described operators. The EA used truncation selection and a fixed number of children in each generation. The population size and number of children were tuned (see Section 5.5.1).

The SMBO algorithm also solved the upper-level problem. We used the Kriging model from the CEGO package, with the kernel given in eq. (5.3). The model was trained within $1,000$ likelihood evaluations (via DIRECT). The EA searched on the surrogate model with $10,000$ evaluations of the EI criterion in each iteration. The SMBO search was initialized with 20 random trees.

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 time. Each algorithm run was repeated 20 times.

5.5.1 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 GP algorithm's population size μ and number of children λ produced in each iteration were tuned. All combinations of $\mu = \{5, 10, 15, 20\}$ and $\lambda = \{1, 2, 3, 4, 5\}$ were tested. The optimization performance was expected to be sensitive to these parameters due to the minimal fitness evaluation budget.

For the SMBO algorithm, we did not tune μ and λ. Due to the overall more significant 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 μ and related parameters. Since $10,000$ evaluations of the surrogate model were allowed, a (relative to the model-free EA) large $\mu = 200$ was given to the EA and correspondingly larger $\lambda = 10$.

We also performed preliminary experiments with the mean square error (MSE) instead of the correlation-based fitness measurement in Equation (5.1). The MSE-based experiments yielded relatively poor results with SMBO. This may be explained by the penalty for infeasible candidates. The penalty value is challenging 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

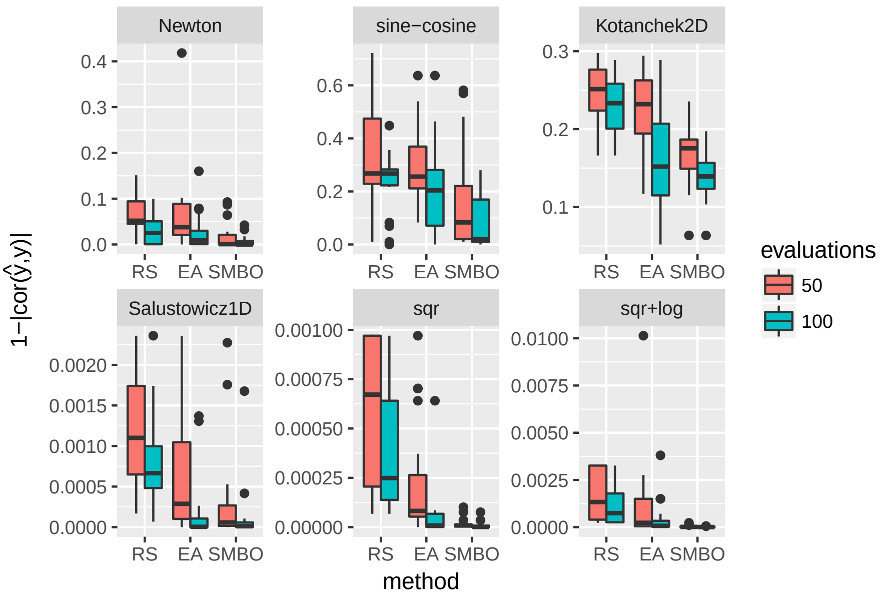


Figure 5.3: *Boxplot of best found values after 50 and 100 evaluations respectively.*

fitness landscape. While our preliminary experiments were not very detailed, they can be counted as additional tuning efforts since they influenced the choice of the correlation measure used in the phenotypic distance.

5.5.2 Analysis and Discussion

Boxplots of the best observed fitness after 50 and 100 evaluations of the objective function F are shown in Figure 5.3. We report results of the tuned, model-free EA that achieved the best mean rank on all problems ($\mu = 15$, $\lambda = 1$). The minimal λ makes sense, as it allows to perform a large number of iterations despite the small budget. We tested for statistical significance of the observed differences via the non-parametric Kruskal-Wallis rank sum test and Conover posthoc test for each problem and number of evaluations, with a significance level of 0.05. SMBO was significantly better than its two competitors in most cases, except for Salustowicz1D and Kotanchek2D after 100 evaluations, where

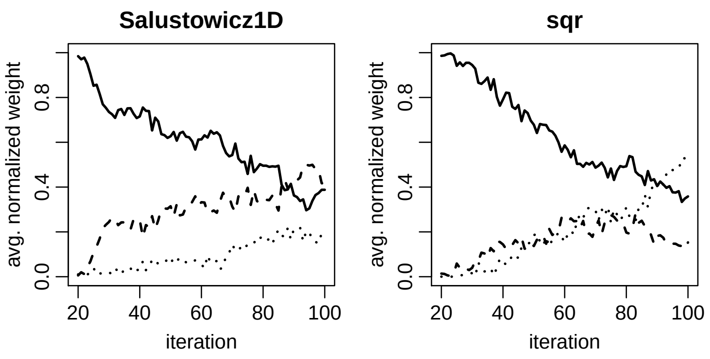


Figure 5.4: *Average normalized weights for the different kernels/distances. Solid line: PHD, dashed line: TED, dotted line: SHD2.*

no evidence for significant differences to the model-free EA is found. The EA was significantly better than the plain RS, except for Newton and sine-cosine (50 and 100 evaluations) as well as Kotanchek2D (50 evaluations).

To determine which distance measures contributed to these results, the weights of the linear combination are shown in Figure 5.4. The weights are normalized so that they sum up to one. We show results for two problems since they are similar in the other four cases. Usually, the PHD received the largest weights in the beginning, whereas the importance of the TED increased throughout the run, sometimes overtaking the PHD. SHD usually does not contribute as much, except for the sqr problem instance. Here, SHD overtakes both other distances at the end of the run. The generally more significant importance of the PHD compared to SHD is in agreement with previous results by Hildebrandt and Branke [2015], where a similar distance achieved better results than SHD.

We confirmed these results by additional optimization experiments for every single distance (i.e., without a linear combination). Runs with PHD tended to suggest reasonable candidate solutions early, whereas TED and SHD performed better later on. The linear combination performed at least as well as the best of the single-distance models.

5.6 Conclusion and Outlook

We investigated whether three distance measures can be employed in an SMBO algorithm based on a Kriging model. We tested the algorithm with SR tasks. Concerning the research questions stated in Section 5.1, our results can be summarized as follows:

1. The distance measures PHD, SHD, and TED are quite diverse. The SHD differentiates poorly between trees with different complexities. Especially the TED seems to be much more fine-grained, but it requires the most computational effort. On the other hand, the PHD is comparatively cheap to evaluate and independent of the genotype.
2. Interestingly, the PHD seemed to contribute most, followed by the TED. This was especially true for small data sets at the beginning of an optimization run. Later on, TED and, to a lesser extent, SHD gained importance.
3. A Kriging model based on a linear combination of the three distances seems beneficial for SMBO. The SMBO algorithm outperformed a model-free algorithm and random search. All algorithms used no more than 100 fitness evaluations.

In future work, we would like to determine how well these results apply to other problem classes. Furthermore, alternatives to the linear combination of distances should be investigated.

6 Comparison of Genotypic and Phenotypic Distance Metrics for Modeling Neural Networks

Chapter 6 was published as:

Hagg A, Zaefferer M, Stork J, & Gaier A (2019, July). Prediction of neural network performance by phenotypic modeling. *In Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1576-1582). ACM, New York.

Surrogate models are used to reduce the burden of expensive-to-evaluate objective functions in optimization. By creating models which map genomes to objective values, these models can estimate the performance of unknown inputs and so be used in place of expensive objective functions. Evolutionary techniques such as genetic programming or neuroevolution commonly alter the structure of the genome itself. A lack of consistency in the genotype is a fatal blow to data-driven modeling techniques: interpolation between points is impossible without a common input space. However, while the dimensionality of genotypes may differ across individuals, in many domains, such as controllers or classifiers, the dimensionality of the input and output remains constant. In this work, we leverage this insight to embed differing neural networks into the same input space. To judge the difference between the behavior of two neural networks, we give them both the same input sequence and examine the difference in output. This difference, the phenotypic distance, can then be used to situate these networks into a common input space, allowing us to produce surrogate models that can predict neural networks' performance regardless of topology. In a robotic navigation task, we show that models trained using this phenotypic embedding perform as well or better as those trained on the weight values of a fixed topology neural network. We establish such phenotypic surrogate models as a promising and flexible approach that enables surrogate modeling even for representations that undergo structural changes.

6.1 Introduction

Optimization of real-world engineering problems is a demanding task. Frequently, expensive simulations are needed to determine the quality of a solution. For example, to determine whether a car model produces low wind resistance, a numerical simulation of the airflow needs to be performed, which can take many hours or even days. In robotics control, we need to run physics-enabled simulations or run real-world experiments. Iterative optimization requires many of these evaluations to reach a satisfactory solution.

One of the most helpful techniques is to replace most evaluations with the predictions of a surrogate model [Jin, 2011; Jin et al., 2019]. The surrogate model is an efficient computational model trained with examples from the real objective function but takes orders of magnitude less time to predict the objective function's value for a specific candidate solution. Commonly used models such as Gaussian processes (also known as Kriging) or support vector machines [Rasmussen, 2004; Jin, 2011] are based on the similarity of candidate solutions. Similarity-based surrogate models have been used in such varied domains as: shape optimization in fluid dynamics [Ong et al., 2003; Daniels et al., 2018], the discovery of new drugs [De Grave et al., 2008], the placement of hospital trauma centers [Wang et al., 2016], and even to the optimization of other machine learning methods [Snoek et al., 2012; Stork et al., 2017]. To produce a prediction, these models interpolate based on the distance of a candidate solution to known examples. They assume that the objective function is smooth: the closer a candidate is to a known example, the closer its function value will be to that example.

A prerequisite for similarity-based surrogate models is that a distance metric is defined to encode a solution. Surrogate models are, therefore, usually applied to solution representations that encode a fixed number of parameters. Recently, more complex encodings have been developed that do not have a constant input space. Prime examples of such encodings are compositional pattern producing networks (CPPN) [Stanley, 2006], that encode complex shapes or behaviors indirectly, neuroevolution [Stanley and Miikkulainen, 2002], in which the topology of neural networks can be evolved, or genetic programming [Koza, 1992], which evolves the topology of graphs or trees representing computer code or mathematical equations. The non-uniform input space of these encodings frustrates typical ways of measuring distance as the dimensionality and even the meaning of these dimensions varies from one individual to the next (Figure 6.1).

A second problem arises when the quality of a solution depends on interaction with its environment. This behavior might vary significantly even if the parameterization of the encoding is changed only a tiny amount. If we trained a similarity-based model to predict the quality of such an encoding, a parameterization that is close to a training example would be assigned a similar

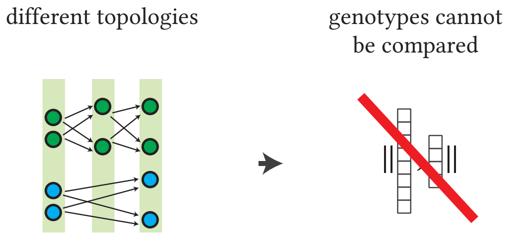


Figure 6.1: *Two networks with different topologies cannot be compared based on their genotypes.*

fitness, although its actual fitness might be very different.

To enable SMBO of these kinds of encodings, we investigate the idea of measuring distances not of the encoding, the genotype, but instead of the expression of the encoding, the phenotype. The phenotype may include morphological as well as behavioral aspects [Dawkins, 1982], and so can give us more information about how similar two individual solutions are than the genotype alone [Stork et al., 2019a]. Our main insights are that (1) regardless of a network's internal composition, the size of the output in relation to the input is constant, and (2) the relation between input and output describes the behavior and thus is a valuable proxy for the similarity between networks. To measure the difference in the behavior of two networks, we can give them the same input sequence and measure the difference in the output sequence using a standard metric like Euclidean distance. Using randomly selected but fixed input sequences, we do not have to run an actual simulation to get the output sequence. Instead, we sample the input/output relation and use the ad hoc difference in the output sequences of two individuals to measure their distance. This distance measure can now be used to build a similarity-based surrogate model.

In this chapter, we evaluate whether we can model the phenotype of ANNs using a phenotypic distance metric and whether the models are competitive to those using a genotypic distance metric based purely on the neural network's weights. We qualify the results with a more in-depth analysis of the complexity of the phenotypic modeling problem, which shows that the intrinsic dimensionality of the phenotypic data is much lower than that of the genotypic data.

6.2 Related Work

Similar to phenotypic distances, semantic distances are used in genetic programming. These semantic distances can be defined as a distance of the outputs of GP individuals, determined with the same measure that is used in the fitness function [Moraglio et al., 2012] . Semantic distances are applicable where the fitness function can be computed as a distance between the optimal target vector and the candidate outputs, such as in supervised classification or symbolic regression. In these cases, the semantic distance has a fitness distance correlation of exactly one and can be utilized to construct specific mutation and crossover operators, rendering the problem uni-modal.

Phenotypic distances have also been employed in a surrogate modeling context for GP. Hildebrandt and Branke [2015] suggested a phenotypic distance for dynamic job shop scheduling problems. Their definition of phenotypic distance compares individuals according to the results of evolved dispatching rules on a small set of test situations. Unlike semantic distances, their phenotypic distance is not identical to the measure used in the actual fitness function. This is necessary in the context of surrogate modeling for expensive fitness functions. If the fitness function is expensive to compute, it would also be expensive to use the same evaluation to compute a distance between candidates. Such an approach would render the surrogate model itself expensive.

Zaefferer et al. [2018] compare different genotypic and phenotypic distances for surrogate models in symbolic regression. Here, the underlying measure is not identical to that used in the fitness function. Specifically, the fitness function considers fixed coefficients in the symbolic expression. These coefficients are otherwise optimized during an actual fitness evaluation, which may become costly. In both of these cases, the phenotypic distance was reported to yield better results than genotypic distances [Hildebrandt and Branke, 2015; Zaefferer et al., 2018].

Doncieux and Mouret [2010] discuss the use of behavioral similarity in evolutionary robotics to employ a diversity measure for a multi-objective optimization approach. They compare different distances based on the states, outputs, and trajectories given concrete robot tasks. They outline that using these behavioral distances as the second objective in multi-objective optimization can enhance overall performance.

A first approach utilizing a surrogate model for evolving neural networks given complex control tasks was discussed by Gaier et al. [2018]. An evolutionary algorithm was combined with a surrogate model based on a hereditary distance defined in the context of neuroevolution of augmenting topologies (NEAT) as *compatibility distance*. The approach is able to improve the evaluation efficiency significantly. Stork et al. [2019a] also investigated surrogate models for neuroevolution. They examined simple classification tasks and compared a phenotypic distance measure to genotypic distances in surrogate-based cartesian genetic programming. The use of a phenotypic distance was shown to be very promising in terms of evaluation efficiency.

In this work, we build on the ideas about using the output of network representations and investigate whether sampling the phenotypes allows us to measure distances between networks. We evaluate whether we can use this distance metric to model the behavior of neural networks in a robot control task and predict their fitness.

6.3 Methods for Modeling Neural Networks

6.3.1 Kriging

To perform interpolation or regression on a given data set, Kriging models assume that the underlying data is sampled from a Gaussian process. For an in-depth introduction to Kriging and its application in model-based optimization, we refer to Forrester et al. [Forrester et al., 2008]. We give only a rough overview, focusing on the issues relevant to this work.

Here, the training data of the model is denoted as a set of n solutions $\boldsymbol{X} = \{\mathbf{x}^{(i)}\}_{i=1\ldots n}$ in a k-dimensional search space. The corresponding n observations are denoted with $\mathbf{y} = \{y^{(i)}\}_{i=1\ldots n}$. For an unknown point in our search space, $\mathbf{x}^*$, Kriging intends to estimate the unknown function value $\hat{y}(\mathbf{x}^*)$. In its core, the model assumes that the observations at each location $\mathbf{x}$ are correlated via a kernel function. We consider kernel functions of the following type:

$$\mathrm{k}(\mathbf{x},\mathbf{x}') = \exp\left(\, -\theta \mathrm{d}(\mathbf{x},\mathbf{x}')\right). \tag{6.1}$$

This essentially expresses the correlation of two samples $\mathbf{x}$ a $\mathbf{x}$', based on their distance $\mathrm{d}(\mathbf{x},\mathbf{x}')$, and a kernel parameter $\theta \in \mathbb{R}^+$. Kernel parameters are usually determined by maximum likelihood estimation; that is, they are chosen such that the data has the maximum likelihood under the resulting model. MLE usually involves a numerical optimization procedure [Forrester et al., 2008]. The distance measure $\mathrm{d}(\mathbf{x},\mathbf{x}')$ can potentially be any measure, though not all ensure that the kernel is positive semi-definite, a common requirement [Forrester et al., 2008]. In this chapter, we use the Manhattan distance, which is less affected by issues related to high-dimensional data [Aggarwal et al., 2001], defined as:

$$\mathrm{d}_{\mathrm{Man}}(\mathbf{x},\mathbf{x}') = \sum |x_i - x_i'| \tag{6.2}$$

Rather than a single parameter θ, a different θ can be used for each dimension i of the input samples, enabling the model to estimate the influence of each dimension on the observed values. However, in the interest of simplicity and computational efficiency, we opt for an isotropic kernel with a single θ.

Once the pairwise correlations between all training samples are collected in a matrix $\boldsymbol{K}$, the Kriging predictor can be specified with

$$\hat{y}(\mathbf{x}^*) = \hat{\mu} + \boldsymbol{k}^{\mathrm{T}}\boldsymbol{K}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu}), \tag{6.3}$$

where $\hat{\mu}$ is another model parameter (estimated by MLE), $\boldsymbol{k}$ is the vector of correlations between training samples $\boldsymbol{X}$ and the new sample $\mathbf{x}^*$, and $\mathbf{1}$ is a vector of ones. The error or uncertainty of the prediction can be estimated with

$$\hat{s}^2(\mathbf{x}) = \hat{\sigma}^2(1 - \boldsymbol{k}^T\boldsymbol{K}^{-1}\boldsymbol{k}^T), \tag{6.4}$$

where $\hat{\sigma}^2$ is a further model parameter to be estimated by MLE.

6.3.2 Genotypic vs Phenotypic Distance

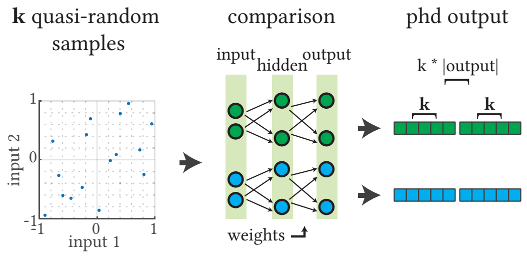


Figure 6.2: *Sampling the phenotype to compare two individual networks.*

The networks that we investigate in this work are results of optimization runs with fixed network topologies. This allows us to evaluate and compare the efficiency of models based on both genotypic and phenotypic distance measures. To define a genotypic distance we consider the vector of weights of the neural networks. Let $\mathbf{w_x} = \{w_1, w_2, ..., w_j\}$ be the weight vector of length j associated with a solution $\mathbf{x}$, then we can calculate the genotypic distance by the related weights of two samples: $d(\mathbf{w}, \mathbf{w}')$.

The disadvantage of genotypic distance measures is their lack of applicability when changing topologies are considered. If, in these cases, no clear concept to compare genotypic changes exists (as applied in [Gaier et al., 2018]), the genotypic distance comparison is difficult, misleading, and even destructive [Stork et al., 2019a; Doncieux and Mouret, 2010]. The ability to compare non-uniform topologies makes phenotypic distances a valuable technique, especially in cases when typical distances are not a viable option.

The phenotype displays the behavior of a neural network given a certain set of inputs. For example, in the case of neural networks used as controllers for robots, the phenotype can be defined as the control commands that are issued in response to different sensor inputs. We define a phenotypic distance as follows: Let $\mathbf{s} = \{s_1, s_2, ..., s_k\}$ be the vector of inputs with length k, then $\mathbf{o_x} = \{o_1, o_2, ..., o_{k\times z}\}$ is the associated processed output vector, or phenotype, for a neural network $\mathbf{x}$ with length $k \times z$, where z is the number of neural network output neurons. The phenotypic distance is employed by calculating the difference in the outputs of two samples: $d(\mathbf{o}, \mathbf{o}')$. Figure 6.2 illustrates the sampling of phenotypes and Figure 6.3 shows a comparison of both distances.

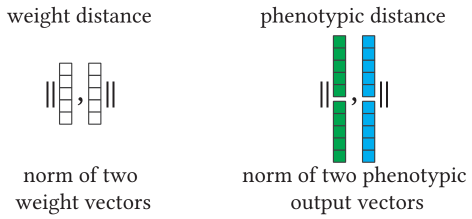

weight distance
phenotypic distance
norm of two weight vectors
norm of two phenotypic output vectors

Figure 6.3: *Weight models are based on weight vectors for fixed-topology networks. Phenotypic distance models are based on fixed-length sampled phenotypic output vectors for any-topology networks. We use the L1 norm (Manhattan distance) for interpolative modeling.*

The phenotypic distance is always task sensitive, i.e., a comparison of two samples $\mathbf{x}$ and $\mathbf{x}'$ requires the definition of an adequate input vector $\mathbf{s}$. In the context of model-based optimization, this input vector needs to fulfill two requirements:

a) the input should be representative for the underlying task, i.e., in the case of robot control, it should follow the given sensor ranges or depict a trajectory of states present in the task.

b) the dimensionality of the phenotype needs to be considered; the length of the input vector for generating the phenotypes might significantly affect the modeling performance as well as the computation time for querying the networks.

Given a carefully selected input vector, the phenotypic distance should be able to provide a clear impression of how the behaviors of two candidate networks compare to each other. A possible disadvantage of our definition of a phenotypic distance is that depending on the underlying task, the real behavior cannot be defined by the output of the neural network controller alone. For example, a robot is further influenced by the structure of the environment and its own body. Two robots with different controllers and phenotypes, one that uses four legs for movement and the other that uses three legs, might behave the same if the 4th leg is disabled due to damage [Doncieux and Mouret, 2010]. However, a representative set of samples of the input/output relationship should be descriptive enough to capture the behavioral differences and so allow the construction of surrogate models.

6.4 Experimental Setup

The goal of our experiments is two-fold. Firstly, to determine whether we can learn reasonable surrogate models based on a diverse set of phenotypic vectors. Secondly, to compare these results to a genotypic model. Our experiments are constructed as follows, we:

1. Run model-free optimization algorithms that optimize the weights of fixed topology neural networks for robot control;
2. Archive a selection of several hundred diverse neural networks from the results of these runs;
3. Train different genotypic and phenotypic surrogate models on a subset of these networks;
4. Test the performance of the surrogate models by predicting the performance of the remainder of the networks.

Problem Setup: Maze and Robot We design robot controllers for the multi-modal maze problem depicted in Figure 6.4. The environment consists of multiple rings and openings (Figure 6.4a). The robot begins in the center of the maze. Here we are not interested in the typical case of finding the best solution to escape the maze. Instead, we seek to establish to what degree we can sample the behavior (or phenotype) of neural network controllers and then derive fitness from those behaviors. This problem is much more fundamental and challenging than predicting fitness alone. To produce this data set of as many different high-performing behaviors as possible, we build up an archive out of robot controllers that reach every point in the maze in the shortest path possible (b). To force diversity of ending positions, a grid-like diversity measure is defined (c). At the end of the optimization, every niche should contain a robot that could reach it using a short path. This way, we can evaluate the distance measures over a diverse set of optimal behaviors.

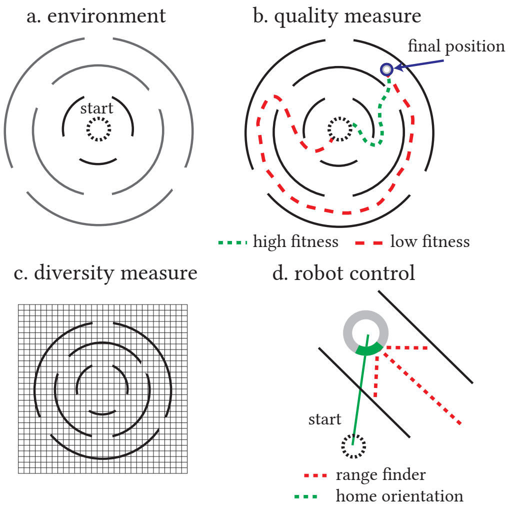


Figure 6.4: *Evaluation takes place in a maze environment (a) with a robot starting in the center. The distance of the path of a robot to its final position defines its quality (b), whereby a diversity measure allows us to train robots to reach all cells in the map (c). Robots can sense the orientation quadrant of the start position and uses three range finders to perceive the distance to the nearest wall (d).*

Simple feedforward controllers (see Figure 6.2) consisting of either 2 or 5 hidden neurons are sought that traverse the maze. Evaluation is performed using the simulation that was created in [Mouret and Doncieux, 2012]. The robot is equipped with three laser sensors that are able to detect the distance to the nearest walls and are set at 45-degree angles around the front (d). In addition, each robot has a home beacon that detects the direction of the robot's start position.

Data Generation We generate data sets to test the quality of our surrogate models. To that end, we record the data of model-free optimization experiments. Here, optimization is performed with a quality diversity (QD) algorithm. These algorithms are not only used to find reasonable solutions but also are intended to find as many diverse and high-performing solutions as possible. We choose MAP-Elites [Mouret and Clune, 2015], which builds up an archive of high-performing elites, one within each niche. Here, niches are defined as cells in the grid shown in Figure 6.4. Parents are selected from the archive at random and their genes mutated with 5% probability to form the next generation of controllers. These child controllers are tested and assigned the cell which corresponds to their end position. If the child arrived at that cell with a shorter path than the current occupant, it replaces the current occupant.

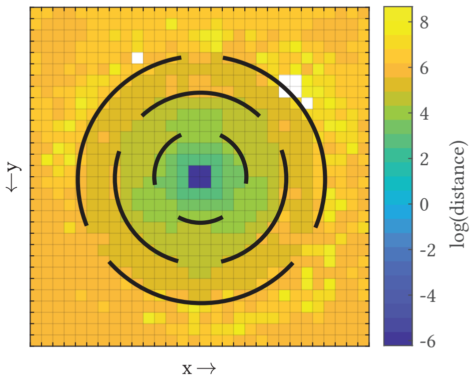


Figure 6.5: *Distance map generated by MAP-Elites (lower distance equals higher fitness). Each niche in the map contains a robot controller that is optimized towards reaching that niche in the shortest path possible.*

Figure 6.5 shows an example distance map after 5000 generations, with almost every niche filled with a high-performing controller. The distance values grow the further they are from the center, which is to be expected. Several controllers end up driving around the maze in circles, which explains the high distance values in some niches.

Data Used for Modeling Data generation was performed either with networks with 2 or 5 hidden neurons. We performed 20 replications; that is, we received data from 20 different QD runs for each experiment configuration, each with a different random number generator seed. This leads to 40 data sets (20 for each number of hidden neurons).

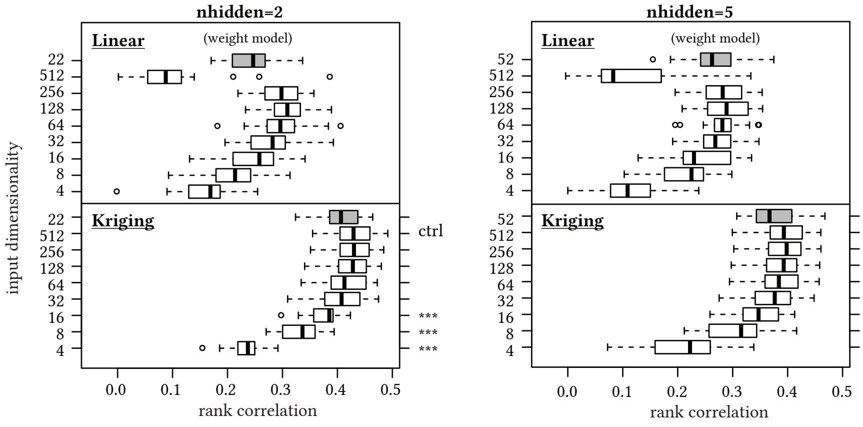


Figure 6.6: *The model quality in terms of correlation (x-axis), for linear and Kriging models and different input spaces, and different numbers of hidden neurons (nhidden). Here, the numbers at the start of each y-axis label denote the dimensionality of the input vector for the corresponding model. Gray fill color indicates a model based on the weights or genotype. The white fill color indicates phenotypic models. The y-axis labels on the right-hand side indicate p-values from a statistical test that compares each of the Kriging models against the model marked with ctrl (*: $p < 0.05$, **: $p < 0.01$, ***: $p < 0.001$).*

Each data set consists of roughly 900 neural network controllers. We received nine different data subsets for each of those controllers: one with the weights and eight with phenotypes of different sizes $(4, 8, \ldots, 512)$. Note that the phenotypes are derived from the two outputs of the networks; if the network is fed with four input samples, we observed eight phenotype values. We can now describe each of the 900 controllers either by its weights or by phenotype vectors (of different lengths).

During modeling, these data sets are split as follows. Four hundred controllers are used to train a model, and the remainder is used to test the model quality.

Note that the observed values y will be log-scaled before modeling, as the data contains strong outliers which might deteriorate the models.

Quality Measures To judge the quality of our models, we use Kendall's rank correlation coefficient [Kendall and Gibbons, 1990]. In contrast to Pearson correlation, this measure only considers ordinal correlation, i.e., the ranks of two compared sets of samples. Kendall correlation is, therefore, an excellent measure to estimate the accuracy of a model when used in rank-based evolutionary optimization methods.

Kriging Model We generate the Kriging model with the R-package CEGO [Zaefferer, 2017] as follows. For MLE, the optimization of the likelihood is performed via the locally biased variant of the dividing rectangles (DIRECT) algorithm [Gablonsky and Kelley, 2001]. It is configured to stop after 2000 likelihood evaluations or when a relative decrease in function values between iterations drops below 10^{-16}. The nugget effect (regularization) of the model is turned on to potentially account for noise in the data or ill-conditioned kernel matrices. The model uses the Manhattan distance (see Section 6.3.1).

Comparison Baseline: Linear Model We include a linear regression model in our experiments as a comparison baseline for the Kriging model. Like the Kriging model, the linear model is trained with the weight or the phenotype data. Since the generated data is potentially very high dimensional, we need some form of variable selection to generate reasonable models. We decided on a forward selection approach via the Aikake information criterion (AIC) [Venables and Ripley, 2002], starting from a model that only consists of an intercept. The most complex linear model may include main effects for all variables, but no interactions or higher-order terms are considered.

6.5 Results and Discussion

Figure 6.6 shows the Kendall correlation achieved by each of our models. Firstly, it can be observed that the Kriging model outperforms the linear model in most cases, as expected. Secondly, the variants based on phenotypic data are able to perform at least as well as the weight models if the number of elements in the phenotype vector is at least 32 or more. The larger phenotype vectors do not seem to yield much further improvement.

We confirmed these observations by applying statistical tests for each number of hidden neurons. Firstly, we tested for the global presence of significant differences via the non-parametric Kruskal-Wallis rank-sum test [Kruskal and Wallis, 1952], which yielded p-values of less than 10^{-8} in both cases, indicating that differences are present. Afterward, we performed Conover's non-parametric many-to-one comparison test [Conover and Iman, 1979], comparing each of the Kriging models against a single model (control group). The chosen control group was the most complex model with phenotype data of dimensionality 512. The implementations of the employed tests were taken from the `stats` and the `PMCMRplus` `R` packages [R Core Team, 2018; Pohlert, 2018]: `kruskal.test` and `kwManyOneConoverTest`. The respective cases with indications for significant differences are marked on the right-hand side of each plot in Figure 6.6. The statistical test essentially confirms the visual evaluation. No evidence for differences is found between the control group and the model with the genotypic weight data. Only models with phenotypic data of dimensionality of 16 or less are deemed to be different from the control group.

Notably, the results suggest that we can use phenotypic surrogate models instead of those based on the genotype or weights. The phenotypic data is largely unaffected by the number of hidden neurons and, hence, the number of weights. Where standard models would struggle to compare the weights of differently structure networks, a phenotypic comparison would still be possible.

The baseline linear model shows some peculiar behavior. The model's performance drops off for models with phenotype vectors of more than 256 elements. This behavior can be primarily explained with a number of coefficients selected by the AIC forward selection procedure, as shown in Figure 6.7. Clearly, the selection procedure will not select more than n variables. The required number of variables seems to increase non-linearly with the dimensionality of the data.

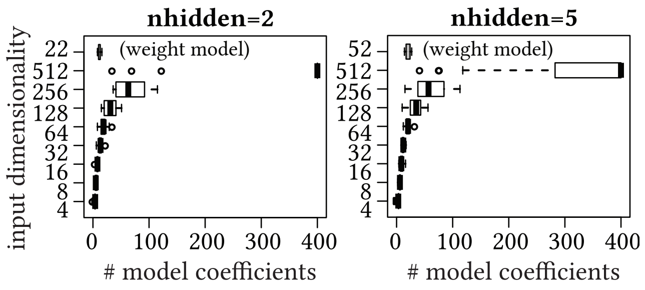


Figure 6.7: *The number of linear model coefficients selected via forward selection based on AIC. Gray fill color indicates a model based on the weights or genotype, the remainder are based on phenotype data.*

Notably, the Kriging model does not show such a performance drop, and in fact, performs exceptionally well even for the very high dimensional phenotype vectors. This may be counter-intuitive at first: Kriging is usually not suggested for high-dimensional data. We suggest two reasons for this: Firstly, we use an isotropic model which avoids the complex optimization of fitting numerous kernel parameters (θ). Secondly, there may be some correlation in the observed phenotypes. Increasing the number of samples used to generate the phenotype vector will increase the dimension yet also increase the density in the sampled space. In that sense, a new phenotype observation added to a large set of existing observations is likely to be quite similar to the existing observations. Essentially, we assume that the latent dimensionality of the data is much lower.

To verify this, we considered a principal component analysis (PCA) of the input data (that is, excluding the dependent variable). For each of our data sets, we performed a PCA on the weight data, as well as on the phenotype data. In each case, we recorded the number of principal components required to explain 90% of the variation in the data set. This number is shown in Figure 6.8. There are two interesting observations here. Firstly, the number of components levels off for the largest phenotype vectors. The median stays at 7 (nhidden=2) and 9 (nhidden=5), despite data sets with several hundreds of variables. It seems that this confirms our assumption that the additional columns due to higher-dimensional phenotype vectors describe a much lower-dimensional latent

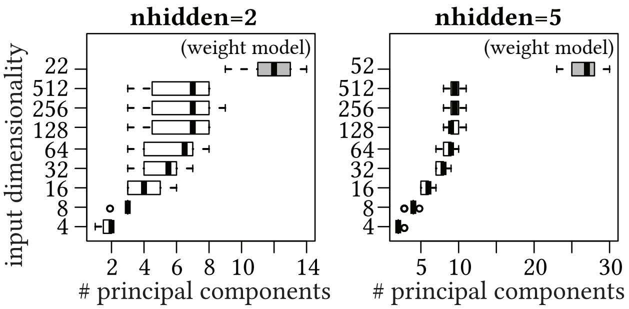


Figure 6.8: *For each data set, the number of principal components required to explain 90% of the variation in the data. This only concerns the respective input data of the surrogate models; the observed output (i.e., quality of the controller) is not considered here. Gray fill color indicates weight or genotype data, the remainder is based on phenotypic data.*

space. Secondly, we can see that the number of principal components for the weights is much larger. However, this does not coincide with better models based on the weight data.

6.6 Conclusion and Outlook

In this chapter, we evaluated the use of phenotypic data of neural networks as a basis for surrogate modeling. We have shown that models based on phenotypic data can perform at least as well as those based on genotypic (weight) data. This was true both for a baseline, linear model, and a non-linear Kriging model. Our analysis further indicates that even high dimensional phenotypes with several hundreds of observations can yield sound Kriging models. A principal component analysis reveals that these high dimensional data sets can be very well reproduced with only very few components. A much larger number of components is required for the genotype data. This success of a phenotypic model is promising since a model based on genotypes becomes infeasible if the compared networks have different structures or topologies.

In the context of evolutionary algorithms that can change the structure and size of the solution encoding, e.g., in surrogate-based neuroevolution. Measuring the behavior of neural networks without using actual simulations not only seems to be possible but also a practical way to compare networks.

Phenotypic distances can be used successfully as kernels to build surrogate models that predict the fitness of networks with varying sizes and topologies. Whereas previous approaches to construct surrogate models of neural networks with non-uniform structure rely on the peculiarities of the evolutionary algorithm [Gaier et al., 2018], our approach is independent of the optimization approach. In fact, a phenotypic distance approach to modeling is independent even of encoding: a neural network grown with NEAT, a fixed topology network optimized with particle swarm optimization, and a controller evolved with genetic programming could all share the same surrogate model.

In future work, we plan to take the obvious next step: to actually use the developed models as surrogates in an optimization framework. In addition, we want to investigate the generation of phenotype vectors in more detail. As the PCA showed, as well as the diminishing returns for models with more phenotype samples, a lower-dimensional data set may suffice to produce good models. Creating better, more condensed phenotype samples with less redundant information is hence of interest to reduce the load of distance calculations.

Being able to successfully model the performance of a robot controller by observing its behavior provides a computationally efficient and effective approach for surrogate modeling of varying-length representations. We show that modeling the behavior of networks avoids some complexities that are caused by genotypic comparisons. SMBO of non-uniform representations will allow a much more diverse set of solutions to be calculated with a limited number of real evaluations.

7
Comparison of Distance Metrics for Surrogate Model-Based Neuroevolution

Chapter 7 was published as:

Stork J, Zaefferer M, & Bartz-Beielstein T (2019, April). Improving neuroevolution efficiency by surrogate model-based optimization with phenotypic distance kernels. *In International Conference on the Applications of Evolutionary Computation (Part of EvoStar)* (pp. 504-519). Springer, Cham.

In neuroevolution, the topologies of artificial neural networks are optimized with evolutionary algorithms to solve tasks in data regression, data classification, or reinforcement learning. One downside of neuroevolution is the large number of necessary fitness evaluations, which might render it inefficient for tasks with expensive evaluations, such as real-time learning. For these expensive optimization tasks, surrogate model-based optimization is frequently applied as it features a good sampling efficiency. While a combination of both procedures appears as a valuable solution, the definition of adequate distance measures for the surrogate modeling process is complex. In this study, we will extend cartesian genetic programming of artificial neural networks using SMBO. We propose different distance measures and test our algorithm on a replicable benchmark task. The results indicate that we can significantly increase the sampling efficiency and that a phenotypic distance based on the behavior of the associated neural networks is most promising.

7.1 Introduction

Artificial neural networks are utilized in many different fields, such as data regression, data classification, or reinforcement learning [Basheer and Hajmeer, 2000]. In each of these tasks, the network topology is significant for the ANNs performance. Often, only parameters such as the edge weights, number of hidden layers, and number of elements per layer are considered during the optimization of an ANN. In neuroevolution, ANNs are generated by evolutionary algorithms. NE allows severe modifications of networks, such as individual connections between neurons and different neuron transfer functions, leading to a large search space of potential topologies. Two example algorithms of this category are neuroevolution of augmenting topologies (NEAT) [Stanley and Miikkulainen, 2002] and cartesian genetic programming of artificial neural networks (CGPANN) [Miller and Thomson, 2000; Turner and Miller, 2013]. Both NEAT and CGPANN may require numerous (>1000) fitness evaluations to find adequate topologies in the large search space. This makes them inefficient for applications with expensive function evaluations such as simulations or real-world experiments, e.g., when an ANN is the controller of a robot that operates in a complex real-time environment. SMBO is frequently used for data efficient optimization of real-world processes, as it has a high sampling efficiency [Koziel and Leifsson, 2013]. Surrogates models are commonly employed in continuous optimization, where Euclidean spaces and distance metrics exist. The more complex discrete search spaces are less often investigated [Zaefferer et al., 2014b]. In this work, we want to extend CGPANN to employ surrogate model-based neuroevolution (SMB-NE) to reduce the load of fitness evaluations. For this task, we defined several distances based on the CGPANN genotypes and a distance that measures, instead of the structure, the difference in behavior of the ANNs. Our approach allows the definition of a phenotypic distance that is indifferent to the size or topology of an ANN. It only requires a representative input set to model the input to output correlation.

Our main research questions are:

1. Is SMB-NE able to outperform CGP neuroevolution in terms of sampling efficiency without loss of accuracy?
2. How can we create a representative input set for the phenotypic distance in SMB-NE?

For all experiments, we utilize data-mining classification tasks as a controllable and cheap to evaluate benchmark for the learning efficiency of SMB-NE using few fitness evaluations. This chapter is structured as follows: Section 7.2 discusses related work. Section 7.3 illustrates the utilized methods and the SMB-NE algorithm. Section 7.4 introduces the different distance measures for SMB-NE. In Section 7.5 the experimental setup and the results are shown and further discussed. Finally, in Section 7.6 we conclude the chapter and give an outlook to future work.

7.2 Related Work

This work is an extension of an unpublished study presented at an informal workshop [Stork et al., 2018]. A recent study by Zaefferer et al. [2018] investigated the use of SMBO in genetic programming with genotypic and phenotypic distance measures. In contrast to this study, they performed tests comparing an EA with SMBO for a bi-level symbolic regression optimization problem. They concluded, that the SMBO approach is able to outperform the EA in terms of sampling efficiency. Moreover, the phenotypic distance performed best and was very fast to compute. A first surrogate model for ANN optimization was applied to NEAT by Gaier et al. [2018]. They use a surrogate-assisted optimization approach by combining an EA with a surrogate distance-based model, employing a genotypic compatibility distance that is part of NEAT. With this approach, they are also able to increase the sampling efficiency. To our best knowledge, nobody else used a SMBO approach for the task of neuroevolution. Phenotypic distances are also used in other applications, for example, the optimization of job dispatching rules. For this tasks, Hildebrandt and Branke [2015] utilize a surrogate-assisted EA, which is able to increase the evolution towards good solutions. The use of surrogates including genotypic distances is more often discussed, e.g., for optimization of fixed neural network topologies in reinforcement learning [Stork et al., 2017].

7.3 Data Efficient Neuroevolution

7.3.1 Neuroevolution by Cartesian Genetic Programming

For NE with CGP, we use the C library `CGP` by A. Turner[1], which also allows the application of NE [Turner and Miller, 2015]. The `CGP` library was modified with function interfaces to the statistical programming language R, distance measures described below, and additional fitness functions. The genotype of a CGPANN individual consists of a fixed number of nodes. Each node has a number of connection genes based on the pre-defined arity with adjacent weight genes and a single categorical function gene. Nodes are only connected to preceding nodes. Duplicate connections to nodes are possible and if present, adjacent weights will be added to form a single connection in the resulting ANN. Moreover, each node has a Boolean activity gene, which signals if it is used

[1]http://www.cgplibrary.co.uk - accessed: 2018-01-12

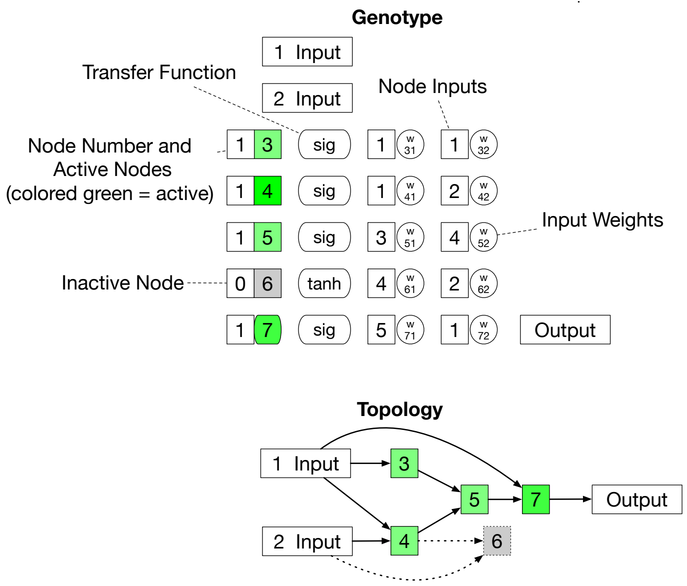


Figure 7.1: *A CGPANN genotype with two inputs, eight nodes, an arity of three and different transfer functions. Each node has a transfer function, a boolean activity gene and several inputs with adjacent weights. Green nodes are active and part of the encoded ANN.*

in the active ANN topology. Figure 7.1 displays an example for a CGPANN genotype and the encoded ANN with two inputs, eight nodes and an arity of three. In CGP NE, the network topology and weights are optimized using an evolutionary approach, utilizing a (1+4)-evolution strategy, i.e., one parent and four new individuals in each generation. In contrast to the standard selection in ES, the rank-based selection process favors the offspring, and thus the new solution, over a parent with the same fitness. Different mutation operators are available, the default is probabilistic random mutation. Inactive genes will not influence the fitness of an ANN.

7.3.2 Kriging

Our SMBO algorithm is based on a Kriging regression model, which assumes that the observations are samples from a Gaussian process [Forrester et al., 2008].

Kriging is a kernel-based model, i.e., the model uses a kernel, or correlation function, to measure the similarity of two samples. One typical kernel for real-valued samples is the exponential kernel, i.e., $\mathrm{k}(x, x') = \exp(-\theta||x - x'||_2)$. Here, the kernel parameter θ expresses how quickly the correlation decays to zero, with increasing Euclidean distance $||x - x'||_2$ between the samples. The parameter is determined by maximum likelihood estimation, optimized by using numerical optimization algorithms [Forrester et al., 2008]. It is straightforward to extend kernel-based models to combinatorial search spaces [Moraglio and Kattan, 2011a; Zaefferer et al., 2014b]. Essentially, the Euclidean distance is replaced by a corresponding distance that applies to the respective search space, with the exponential kernel $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x'))$. The design of appropriate distances $\mathrm{d}(x, x')$ for neural networks is in the focus of this paper.

Kriging is frequently employed in SMBO algorithms, because in addition to relatively accurate predictions, it also provides an estimate of the uncertainty of each prediction. The predicted value and the uncertainty estimate are used to compute infill criteria. These infill criteria are supposed to express how desirable it is to evaluate a new candidate solution x. To that end, the uncertainty estimate is integrated to push away from known, well-explored areas, instead preferring solutions that have large uncertainties, yet promising predicted values. One of the most frequently used criteria is expected improvement [Mockus, 1974; Jones et al., 1998]. We employ the Kriging implementation of the R package `CEGO` [Zaefferer, 2017; Zaefferer et al., 2014b]. It uses distance-based kernels to model data from structured, combinatorial search spaces.

7.3.3 Surrogate Model-based Neuroevolution (SMB-NE)

We extend CGPANN with an SMBO approach, leading to the surrogate model-based neuroevolution strategy outlined below. The strategy is intended to perform a data efficient search by predicting the fitness of candidate solutions with the help of a Kriging surrogate. The algorithm is outlined in 7.3.1. The SMB-NE process starts by creating a random initial set of individuals, in our case ANNs, and evaluates them with the objective function.

Algorithm 7.3.1: Surrogate Model-based Neuroevolution

```
1  begin
       // phase 1: initialization
2      $t = 1$
3      initialize $k$ neural networks $(x_i)$ at random
4      evaluate their fitness on the objective function
5      build Kriging surrogate model utilizing $s_t(x_i)$ and distance measure
        $d(x_i, x_j)$;
       // phase 2: optimization
6      while not termination-condition do
7          if $t > 1$ then
8              rebuild surrogate model $s_t$ with a set $\mathcal{M}_{\sqcup t} \in \mathcal{D}_t$ of observations
9          end
10         optimize EI with a (1+4)-ES to discover improved $x_t$
11         evaluate network $x_t$ fitness $y_t$ on the objective function
12         add evaluated networks to archive $\mathcal{D}_{t+1} = \{\mathcal{D}_t, (x_t, y_t)\}$
13         $t = t + 1$
14     end
15 end
```

The resulting data is used to learn a Kriging model. For learning the model, we define different genotypic and phenotypic distance measures for ANNs in Section 7.4. With the model, we are able to estimate the EI of an individual. In each following iteration, the model is utilized to suggest new promising ANNs by optimizing the EI criterion with the (1+4)-ES algorithm of CGP. The (1+4)-ES is used to introduce the ability to directly compare CGPANN to SMB-NE, without an additional optimization algorithm with a different operator set, or parametrization influencing the results. In each iteration, the single most promising individual is evaluated and added to the archive $\mathcal{D}_t$. As the archive grows during the optimization process, the computational effort of creating the Kriging model rises with $O(m^3)$, where m is the surrogate model sample size. To keep the computational effort on a feasible level, a subset $\mathcal{M}_t \in \mathcal{D}_t$ of size m is used for the modeling process. This modeling set $\mathcal{M}_t$ is formed by selecting $\frac{m}{5}$ of the best and $\frac{4*m}{5}$ randomly drawn individuals out of the archive in each iteration. We chose the fractions in the strategy to ensure a balance between exploration and exploitation. If the size of the archive $\mathcal{D}_t$ is smaller or equal to the size of $\mathcal{M}_t$, all individuals are selected. The influence of the size of the modeling set $\mathcal{M}_t$ is investigated in Section 7.5.2.

7.4 Proposed Kernels and Distances

In the following, we always use an exponential kernel $k(x, x') = \exp(-\theta d(x, x'))$. Here, x and x' represent ANNs. They consist of the weights x_w, input labels x_i, activity labels x_a, and transfer function labels x_f, i.e., $x = \{x_w, x_i, x_a, x_f\}$. All distances $d(x, x')$ are scaled to $[0, 1]$. In addition, we also considered employing the graph edit distance, but decided against it due to its complexity. Computing the graph edit distance is NP-hard [Zeng et al., 2009].

We illustrate the different distances by providing an example that compares two specific ANNs, with a similar structure as outlined in Figure 7.1. For the sake of simplicity and comparability, all weights are set to 1. The two ANNs only differ with respect to two nodes. The transfer function and connections of one active (node 3) and one inactive node (node 8) are changed.

7.4.1 Genotypic Distance (GD)

The genotypic distance is based solely on the genotype of a CGPANN individual. As the CGPANN genotypes have a fixed structure, the distance of two individuals can be calculated by a row-wise comparison of their nodes. By combining the distances of weights, inputs, activity of nodes, and transfer functions we obtain a distance $d(x, x') = ||x_w - x'_w||_2^2 + H(x_i, x'_i) + H(x_a, x'_a) + H(x_f, x'_f)$, where $H(a, b)$ denotes the Hamming distance, i.e., $H(x_f, x'_f)$ is 0 if the transfer function is identical, else $H(x_f, x'_f)$ is 1. The GD is further normalized by the total number of possible comparisons for the given genotypes. Figure 7.2 illustrates an example for calculating the GD distance. Although taking the non active nodes into account, the normalized GD is rather small. We further extended the GD by ordering the inputs of each ANN before the calculation to match the weight distances to the correct input if (and only if) two nodes have similar connections, but a different ordering of inputs in the genotype.

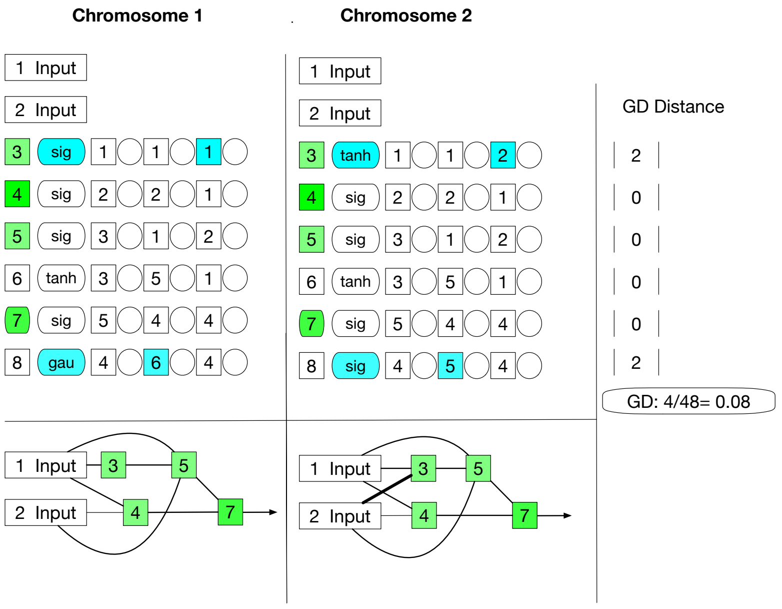


Figure 7.2: *Example calculation the GD distance for two distances. By introducing only small changes to the genotype, the normalized GD stays rather small.*

7.4.2 Genotypic ID Distance (GIDD)

The GIDD is intended to solve an important issue of the genotypic distance: different nodes and adjacent functions and weights in one row of a CGPANN genotype are not easily comparable, if their influence on the resulting ANN and also phenotype is considered. The idea behind the GIDD is to only compare those nodes, which are placed in the same position in the ANN and to solve the problem posed by competing conventions, i.e., that a certain ANN topology can be expressed by numerous genotypes. The distance is based on the active topology and creates IDs which are designed to be unique for an equal placement of a node in the ANN. Inactive nodes do not influence the GIDD. Thus, each active node in the ANN is given an ID based on the connections to prior nodes or inputs and the number of non-duplicate connections of this node. Then, the distance of nodes can be calculated by a pairwise comparison of all node IDs. If a certain node ID matches for both ANNs, the subgraph prior to this

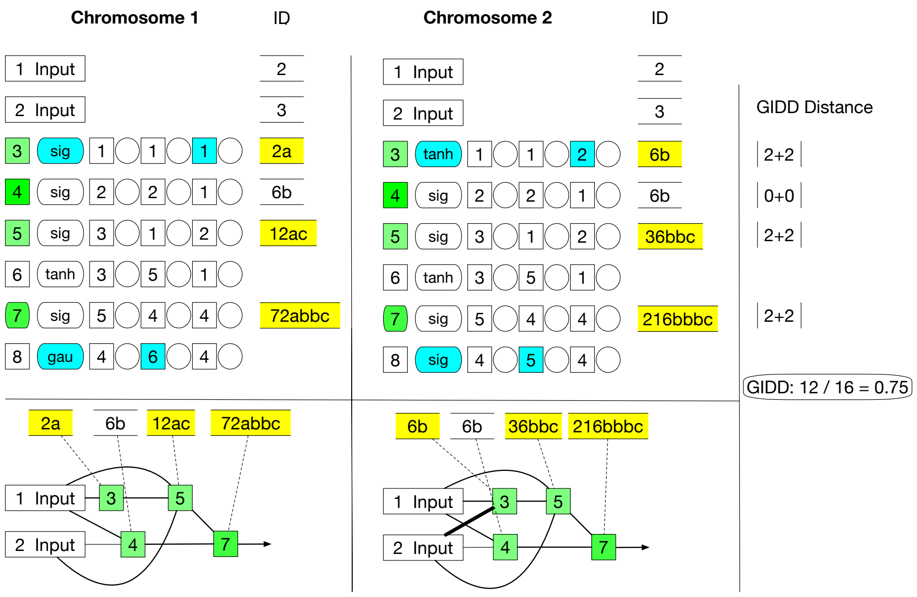


Figure 7.3: *Example calculation for GIDD. The IDs are based on multiplication of prior node or inputs IDs (set to prime numbers) and the number (a=1, b=2, c=3) for non-duplicate connections. The calculated distance is high because of the different node IDs, which are based on their relative position in the ANN.*

node is analyzed recursively for validation of the ANN up to the position of the matching nodes. If all IDs in the subgraph are identical, we assume that the corresponding nodes have an equal position in the ANN topology. For all nodes that are matched in this way, the Euclidean distance of the weights (x_w) and Hamming distance of the transfer functions (x_f) is computed. A node pair can only be used once for this comparison, as node IDs may be present several times in each individual. If all node IDs of both individuals x and x' are equal, the GIDD is simply the distance $d(x,x') = ||x_w - x'_w||_2^2 + H(x_f, x'_f)$ between all weights and transfer functions. If nodes do not match, for each node not present in the compared ANN, a fixed distance is assigned (in our example 2). Again, the GIDD is normalized by the maximum distance of two individuals. Figure 7.3 illustrates the calculation in an example. Contrary to the GD, the GIDD reacts strongly to the introduced changes, as they have a large influence on the node relations, which results in different node IDs and puts them to maximum distance.

7.4.3 Phenotypic Distance (PHD)

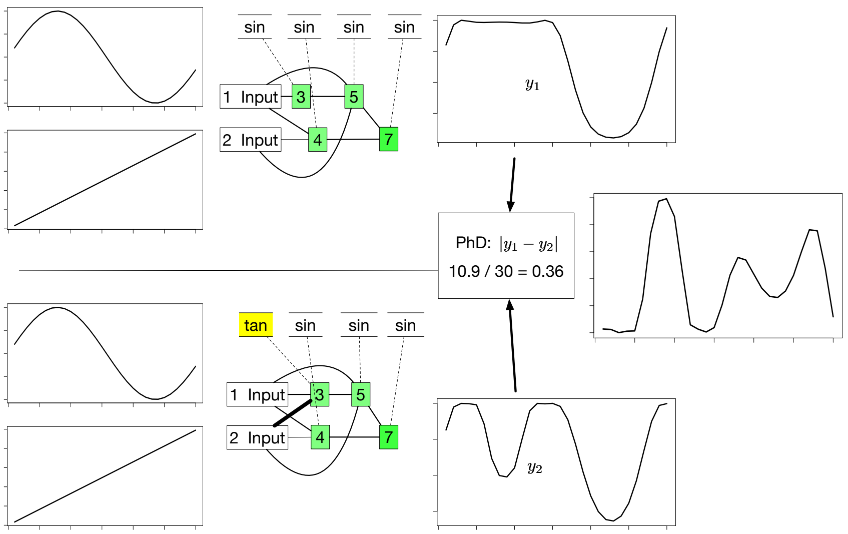


Figure 7.4: *PHD example with continuous inputs and trigonometric transfer functions. Two input samples, from a sine and linear function of length 30 are fed to two ANNs, which differ in their transfer functions and a single connection. The phenotypic distance is the normalized absolute difference of their output signals.*

The phenotypic distance does not utilize any genotypic information of the ANNs to compute the distance. Instead, it utilizes solely their behavior. In our definition, the phenotype of a neural network is how it reacts to a certain input set, i.e., which outputs are produced. This output is then compared to resolve in a distance measure, which is indifferent to changes to the underlying genotype, including transfer functions, weights or connections, which result in the same behavior. More importantly, it is insensitive to the size of the genotype. The PHD distance utilizes the L1 norm distance to account for large dimensions of the input and output vector and is further normalized by the input set length. While it is indifferent to the network topology in terms of encoding, size, number of connections etc., it is very sensitive to the choice of the input set. Thus, these input sets have to be carefully selected to be representative for the underlying task and/or environment. In Section 7.5.2 we examine the influence of the input set for the task of classification. An example calculation for the distance of small ANNs utilizing PHD is given in Figure 7.4. In the example, two continuous input samples, from a sine and linear function of length 30 are fed to two ANNs, where they differ in their transfer functions and a single connection. As this example is intended for understanding the change in the output signals, simple trigonometric functions have been used, which are commonly utilized in genetic programming. The small example shows that minor changes in the network topology can result in a large difference in the phenotype. The disadvantage of the PHD is, that it utilizes the ANN outputs that have to be computed for each new candidate. Thus, in the SMB-NE model search process, the required ANN evaluations might take a considerable amount of computation time, particular for complex ANNs. This renders the PHD particular interesting if on assumption regarding the objective function is met: that is, that the function calls to evaluate the fitness are, compared to the surrogate model search steps, significantly more expensive.

7.4.4 Mixed Distance (MD)

Similar to linear combination distance in [Zaefferer et al., 2018], we utilize a mixed distance of GD, GIDD, and PHD, where each distance receives a weight $\beta_i \in \mathbb{R}^+$ determined by MLE. As the performance of each distance is unknown a-priori, the idea behind the MD is that allows an automatic selection of the most adequate distance measure in each optimization step.

7.5 Experiments

To assess the ability of SMB-NE to improve the efficiency of optimizing the topologies of ANNs, we decided to perform experiments with classification tasks. Classification is well understood, easily replicable, and does not introduce complex problems with the selection of environments or tasks as in general learning (e.g. reinforcement learning). We limited the experiments to a small budget of function evaluations, which provides a realistic scenario for problems with expensive fitness evaluations, such as real-time learning. The experiments are twofold:

– First, we estimate the ability of SMB-NE to learn an elementary data set comparing the introduced distance measures GD, GIDD, PHD and MD.

– Second, we further research how SMB-NE using the PHD reacts to different inputs sets and surrogate model sizes.

7.5.1 Comparison of Distance Measures for SMB-NE

Experimental Setup: For the first set of experiments, the well-known IRIS data set is used as an elementary and fast to compute benchmark problem. IRIS has $n = 150$ samples, 4 variables, and 3 evenly distributed ($n = 50$ for each) classes of different flower types (Iris setosa, Iris virginica and Iris versicolor). The focus of this benchmark is the capability of SMB-NE to learn the best network topology to classify the data set with only 250 fitness evaluations. The fitness function is the adjusted classification accuracy: $\text{acc} = \sum_{i=1}^{n} a_i$, where $a_i = 1$ if the predicted class is true, otherwise, a_i is the predicted probability for the true class. ANN optimized with random search and the inbuilt (1+4)-ES of CGPANN with different mutation rates are being used as baselines. For SMB-NE, all above described distance measures are compared, while for PHD four different input sample sets are tested. As a baseline, we used the complete IRIS data set and additional factorial designs (FD) with small (15) and a large (60) sizes as well as latin hypercube sampling (LHS) with 150 samples. Both the FD and LHS are based on the IRIS variable boundaries. Further, the output of the PHD is adapted by computing the *softmax*, which yields the class probabilities. In this experiment, for the (1+4)-ES of CGPANN and SMB-NE pure probabilistic random mutation is used with a strength of 5%. In SMB-NE, the (1+4)-ES is used in each consecutive iteration alternating between exploitation (local search=L) and exploration (global search=G). Parameters are listed in Table 7.1. The ANNs in this benchmark were kept rather small with a maximum of 40 nodes and 200 connections. This was intended to keep the search space small, but sufficient for the IRIS data set. Algorithm parameters were not tuned and all experiments were replicated 30 times.

Table 7.1: *Parameter setup, where evaluations denote the initial candidates plus the budget for consecutive evaluations. 40 nodes were used to keep the search space small, but sufficient for the IRIS problem given this small budget. Evaluations of CGP-ANN are due to the underlying (1+4)-ES, where in each iteration 4 new candidates are proposed. SMB-NE proposes only a single one.*

arity	**nodes**	**weight range**	**function set**
5	40	[-1,1]	tanh, soft, step, sigmoid, gauss
method	**mutation rate**	**evaluations**	**surrogate evaluations**
Random		250	
CGPANN	5%/15%	$1 + 4 \cdot 63$	
SMB-NE	L:5% G:15%	50+200	L:10+400 G:1000+400

Results: Figure 7.5 visualizes the results. Firstly, the results show that the standard (1+4)-ES of CGPANN performs better than random search, even with the small number of evaluations. A low mutation rate, which depicts a rather local search, seems to be beneficial in this case. SMB-NE utilizing the GD and GIDD distance measures performs only slightly better than the basic CGPANN. With the use of PHD, a significant performance increase is observed, while the PHD with the complete input set performs the best and the LHS the worst. The mixed distance, which also utilizes PHD with complete input sets, cannot benefit from the linear combination, but is able to deliver a close performance to the sole use of PHD. Most runs seem to end up at a local optimum around an accuracy of 66%, which can be explained by the fact, that at this point two out of three classes of the IRIS data set are predicted correctly.
Discussion: An important insight of this experiment is that in comparison to the PHD, the genotypic distances (GD, GIDD) show a poor performance, even for the small genotype size. This fact might be explained by the small correlation between changes to the genotype and the resulting phenotype: small changes to the genes can have a massive impact on the behavior. Moreover, the GD has the

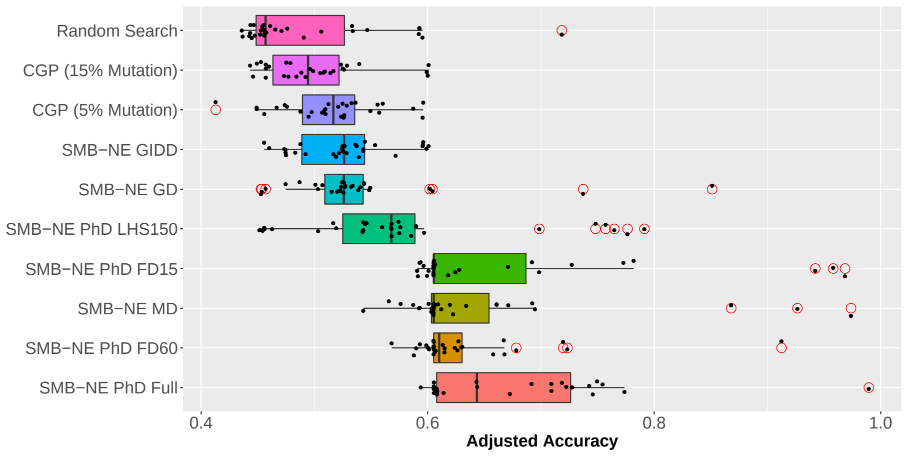


Figure 7.5: *Results after 250 fitness evaluations with 30 replications, comparing random search, original CGPANN ((1+4)-ES) and different SMB-NE variants. The numbers behind the FD and LHS variant depict the length of the ANN input samples to calculate the PHD. The results were ranked (top down) by median values. Red circles depict outliers.*

problem that the calculated distances do not consider that aligned row-paired nodes in the genotype can have different, not similar placements in the ANN topology. This results in a misleading distance calculation of weights, inputs and transfer function for these nodes. However, even the more complex GIDD, which may be able to avoid this issue, does not seem to provide significantly better results. Further given the fact that the GIDD is computationally very expensive to calculate and has also shown numerical problems for large genotypes (as it utilizes several recursions over the complete ANN), it is further considered as not suitable for the task of SMB-NE. In contrast, the PHD distances show very promising results by directly exploiting the ANN behavior. Importantly, the small factorial input sample, which is very fast to compute, shows a good performance. Given the poor performance of GD and GIDD, the MD is able to automatically select the PHD distance measure and deliver equal performance. For the second set of experiments, we thus focus on exploiting features of SMB-NE utilizing the PHD distance measure.

7.5.2 Influence of the Input Set and Surrogate Model Size using SMB-NE with PHD

Table 7.2: *Algorithm parameter setup for the second set of experiments. The size of the genotypes and the number of evaluations was significantly increased.*

arity	**nodes**	**weight range**	**function set**
25	100	[-1,1]	tanh, soft, step, sig, gauss
method	**mutation**	**evaluations**	**surrogate evaluations**
CGPANN	single active	$1 + 4 \cdot 137$ (x10, x100)	
SMB-NE	single active	50+500	L:10+400 G:1000+400

Experimental Setup: To assess the performance of PHD, we discarded the GD, GIDD, and MD distance and introduced two more complex data sets, the glass and the cancer data set[2]. Both data sets were preprocessed by normalization and subsampling. Glass has 9 variables (material composition), 6 unevenly distributed classes of different glass types and 214 samples, while cancer has 9 variables, 2 classes (cancer yes/no) and 699 samples. Two benchmarks were conducted. The first benchmark investigated the influence of the input sample set used for generating the PHD distance measure. For the benchmark, different input samples were created by design of experiments (DOE) methods. All DOE sets are based on the known variable ranges and are not subsamples of the original data sets. The lengths of the samples are identical for both data sets, as they have the same number of variables. We compared the following:

1. Small and large factorial designs, including main, interaction and quadratic effects, with 55/157 samples each.
2. Small and large Latin Hypercube samples, with 55/110 samples each.
3. The complete datasets as baseline input set, with 214/699 samples each.
4. As algorithm baseline, CGPANN with an increasing number of evaluations (5.5×100, $5.5 \times 1,000$, and $5.5 \times 10,000$).

The motivation of this benchmark is as follows: in real-world optimization tasks, often *a priori* information of the the task and/or environment is sparse, as

[2]Available in the UCI machine learning repository: https://archive.ics.uci.edu/ml/index.php

the underlying problem is a black-box problem. Thus, initially no data set is available to serve as an input set and the user has to rely on design of experiment methods to create input data for the PHD. Further, several changes were made to the algorithm setup to account for the more complex classification problems:

– The genotype size was significantly enlarged to 100 nodes with 25 arity, resulting in a maximum of 2500 weights/connections.

– For all compared algorithms, the mutation operator is changed to Goldmans single mutation, which mutates exactly one active node in the genotype (and an arbitrary number of inactive nodes).

– The number of total function evaluations was raised to 550, while fixing the surrogate model set $\mathcal{M}_t$ size to a value of 100. Each sample hereby consists of 80% random and 20% of the most fit individuals from the solution archive $\mathcal{D}_t$.

Table 7.2 shows the adjusted algorithm parameter setup. In the second, connected benchmark, the influence of the surrogate model size, which is used in each iteration of SMB-NE, to the overall optimization performance is analyzed. The glass data set with the complete input set and cancer with the factorial input set are compared for different model set $\mathcal{M}_t$ sizes
Results: Figure 7.6 shows the results of the benchmarks with SMB-NE using PHD for different input sets created by experimental design techniques. For glass, the SMB-NE with the complete input set performed best and on one level with CGPANN with 100 times more real function evaluations. This indicates that SMB-NE is clearly able to improve the sampling efficiency, if the (best possible) input data set is selected. All DOE input sets performed on an equal level similar to CGPANN with 10 times the evaluations. They are thus also able to significantly increase the sampling efficiency, while utilizing an input set, which requires (nearly) no a priori information. The results for cancer firstly indicate that the underlying problem seems to include strong local optima, as we can identify certain clusters of different levels of the adjusted accuracy (around 0.8, 0.86, and 0.95).

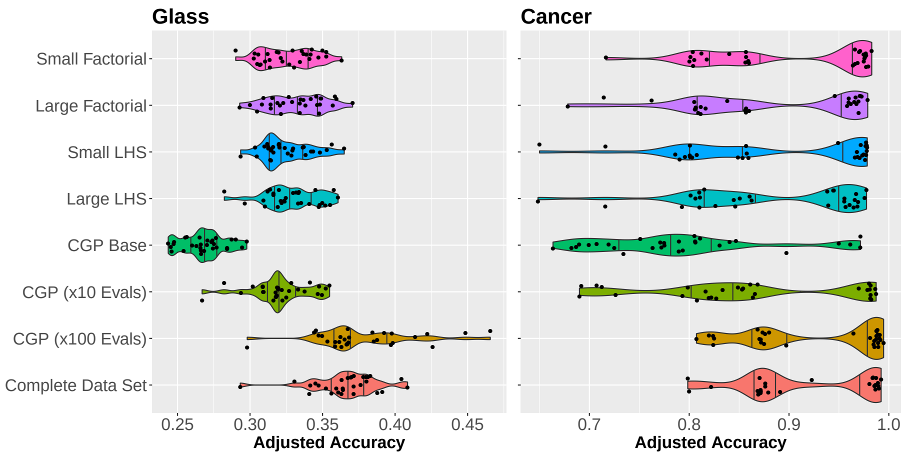


Figure 7.6: *Violin plot of the benchmarks with SMB-NE using PHD for different input sets created by experimental design techniques. The results are compared to the complete dataset and CGPANN using a different number of total function evaluations*

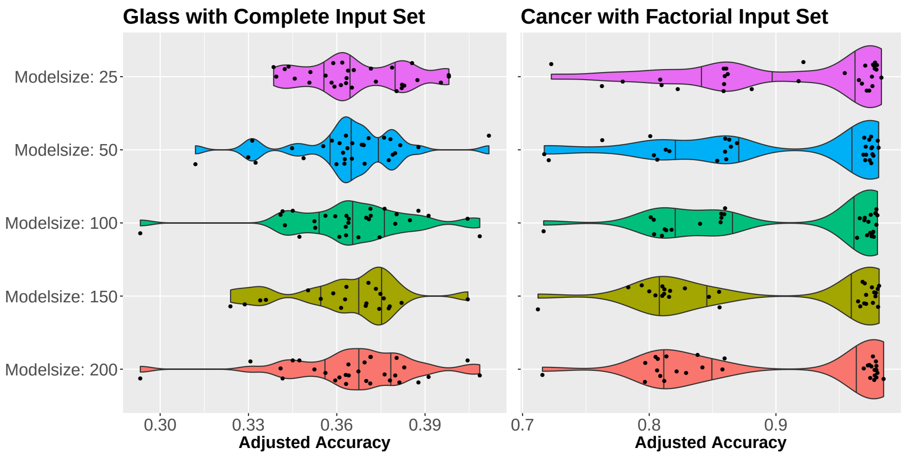


Figure 7.7: *Results with different surrogate model sizes during sequential optimization run for SMB-NE using PHD. Compared datasets are glass with complete input set for PHD, cancer with factorial input set for PHD.*

If we only consider only the best cluster, again we can identify that again the complete set performs best, together with CGPANN x100, while the DOE sets are again on one level with CGPANN x10. Figure 7.7 shows the results of the benchmark with different surrogate model sizes during sequential optimization runs. In contrast to our expectations, the benchmarks show that the model sample size does not seem to have a significant impact on the performance. Even the small sample sizes perform on the same level.

Discussion: The second set of experiments significantly shows two features of SMB-NE using the PHD. As expected, the choice of input set has a strong influence on the algorithm performance. The different designs and input sample sizes show a similar performance, which is an unexpected finding, as we would have anticipated that more samples and a larger design would lead to a more precise representational distance for the complete dataset and thus an improved performance. As the complete dataset shows the best performance, a representatively measured dataset of the original task seems to be the best choice, if initially available or producible by experiments. For the surrogate model size, we anticipated that the large model, which has more information, would be beneficial to the search process, but the smaller, more local models also worked well. Thus, it can be held on a feasible level which is fast to compute.

7.6 Conclusion and Outlook

In this chapter, we proposed a new surrogate-based approach for the task of neuroevolution. Further, we investigated the influence of different distance measures for constructing the surrogate during the optimization process. We have shown that SMBO is a valuable extension for CGPANN, which is able to improve the NE of ANNs in case of small evaluation budgets. Regarding research question 1, we can conclude that SMB-NE with phenotypic distance kernels shows significantly better results than basic CGPANN. Utilizing the PHD with a perfect input set, we were able to reach a sampling efficiency which is on one level with CGPANN with 100 times more function evaluations. Further, the comparison indicates that SMB-NE utilizing genotypic distances does not provide significant performance increases. This fact can be explained by the low correlation of changes in the genotype to the resulting ANN behavior. Regarding research question 2, the experiments have shown that the choice of input sets for computing the PHD has a significant influence on the algorithm performance. Input sets created by DOE methods show a reduced performance in comparison to a perfect input set. Finally, we can conclude that the PHD, which is insensitive to the ANN size, is very promising. A similar study could be conducted with NEAT instead of CGP and it would be interesting to compare the phenotypic distance to the inbuilt compatibility distance in terms of the evaluation performance. Former considerations include how we can employ the PHD in complex tasks, such as evolving robot controllers, which pose additional challenges how to select a representative input vector to compute and compare the behaviors of the ANNs.

8
Surrogate Model-Based Optimization for Behavioral Neuroevolution in Reinforcement Learning

Chapter 8 was published as:

Stork J, Zaefferer M, Bartz-Beielstein T, & Eiben AE (2019, July). Surrogate models for enhancing the efficiency of neuroevolution in reinforcement learning. *In Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 934-942). ACM, New York.

In the last years, RL received much attention. One method to solve RL tasks is NE, where neural networks are optimized by evolutionary algorithms. A disadvantage of NE is that it can require numerous function evaluations while not fully utilizing the available information from each fitness evaluation. This is especially problematic when fitness evaluations become expensive. To reduce the cost of fitness evaluations, surrogate models can be employed to replace the fitness function partially. The difficulty of surrogate modeling for NE is the complex search space and how to compare different networks. To that end, recent studies showed that a kernel-based approach, particularly with phenotypic distance measures, works well Gaier et al. [2018]. These kernels compare different networks via their behavior (phenotype) rather than their topology or encoding (genotype). In this chapter, we discuss the use of surrogate model-based neuroevolution using a behavioral distance for reinforcement learning. In detail, we investigate a) the potential of SMB-NE with respect to sampling efficiency and b) how to select adequate input sets for the behavioral distance measure in a RL problem. The results indicate that we are able to considerably increase the sampling efficiency using dynamic input sets.[1]

8.1 Introduction

Neuroevolution is a technique concerned with the construction of artificial neural networks via evolutionary optimization algorithms. One important application of NE is to generate control policies in RL, where it is a considerable challenge to evolve competitive ANN policies with evolutionary methods. The mapping from the genotypical representation to its phenotype and behavior, and finally to the fitness measurement (i.e., its ability to solve a learning task) can become extremely complex. Evolutionary algorithms will need to spend a significant amount of fitness function evaluations to find well-performing networks. This may become an issue if fitness evaluations are expensive and dominate the optimization process's overall time or resource consumption.

[1]For clarity, the term "phenotypic distance", used in the published version, was changed to "behavioral distance".

Surrogate model-based optimization is one way to deal with this issue [Jin, 2011]. Here, data-driven models partially replace the expensive fitness function. Except for a few recent studies [Gaier et al., 2018; Stork et al., 2017, 2018, 2019a], SMBO has found no application in the context of NE.

Following these recent developments, we intend to design surrogate models that allow us to learn a cheap yet accurate representation of the genotype-phenotype-fitness mapping. In that context, we also focus on kernel-based Kriging models. The approach of kernel-based modeling with Kriging for complex, combinatorial structures is discussed in more detail by Zaefferer [2018]. For graphs, such as ANNs, this can become a difficult task. Specific graphs, such as trees, may allow computing kernels based on measures like the tree edit distance [Pawlik and Augsten, 2015]. Such distances on the genotype can be plugged into the kernel function (i.e., replacing the Euclidean distance) and used to model the genotype-fitness mapping [Zaefferer et al., 2018]. However, the same is not as simple for graphs like neural networks, as the computation of edit distances is NP-hard in the general case. At best, approximate distances can be used, such as the compatibility distance employed by Gaier et al. [2018]. Stork et al. [2017] discuss the use of surrogates of fixed ANN topologies in control tasks using genotypic distances.

As an alternative to genotypic distances, it is often possible to compute the distance on some form of behavior or phenotype. This idea was first discussed by Hildebrandt and Branke [2015] in the context of genetic programming for dynamic job shop scheduling problems. It was later also tested for symbolic regression by Zaefferer et al. [2018]. The key idea of this approach is that complex structures can be compared by observing their output (phenotype) rather than their structure (genotype).

In terms of NE, different distance measures were recently tested for classification problems by Stork et al. [2018, 2019a]. They concluded that phenotypic distances for ANNs are promising if the correct input signal is chosen. A reasonably different model has been used for an RL problem in the context of NE by Koppejan and Whiteson [2011]. Their goal was not to replace the fitness function (as is usually done in SMBO). Instead, they intended to reduce the sample cost involved in testing a controller within different instances of a specific problem class. Instead of a purely data-driven model, they employ a model that is mainly based on understanding the physical system under consideration (a hovering helicopter). In terms of the classification by Bartz-Beielstein and Zaefferer [2017], this can be seen as a customized modeling strategy. A transfer to other problem domains is not straightforward. Most approaches considered in our study can be seen as similarity-based strategies or mapping strategies.

In contrast to the related work, we focus on applying surrogate model-based neuroevolution for RL and the unique needs that arise from solving such tasks, particularly considering the computation of a behavioral distance. In summary, we investigate the following questions:

- Q-I How can we learn the mapping from a neural network to its performance in terms of solving RL tasks?
- Q-II How does a model-based NE compare to model-free NE on RL problems?
- Q-III How should behavioral distances be configured to generate well-performing surrogate models?

We describe the corresponding models and algorithms for NE and RL in Section 8.2. SMB-NE in the context of RL is described in Section 8.3. Our experimental setup is described in Section 8.4 and the results are discussed in Section 8.5. Section 8.6 concludes the work.

8.2 Methods for Model-Based Search

The application of model-based search for RL problems introduces a set of challenges. In general, we want to distinguish between three possible scenarios:

- S-1 *New task:* We want to solve a new task, i.e., no prior experiments were conducted, and no data is available. Here, no information from prior runs can be used to accelerate the current run, and initial experiments have to be conducted to gather information.

S-2 *Same task, different instance:* Data and optimized controllers from former experiments are available, and a different problem instance of the same environment (e.g., different start parameters) has to be solved. For many cases, the ANN controller trained for prior runs may be reused if it is not overfitted and provides a robust solution performance. If not, the existing ANN controller can be subject to further optimization, where a fast convergence to a good solution is anticipated.

S-3 *Same task, different environment:* Data from former experiments is available, but a different yet similar environment needs to be solved. For example, changes to the environment, such as a different maze (in a maze solving problem) or the different physical shape of a robot or appliance, could be considered. In this case, a prior optimized ANN controller could provide a good starting solution. An available data model of the optimization run could still provide valid information.

In this work, we will focus on the first scenario (S-1), whereas for (S-2), different instances will be tested, and a robust controller is part of the benchmark target. (S-3) will be part of future work.

Our model-based approach for solving RL tasks is a combination of existing algorithms: cartesian genetic programming for neural networks (CGP-ANN), SMBO, and specific Kriging models utilizing behavioral distance (BD) kernels. We describe these algorithms in the following.

8.2.1 Cartesian Genetic Programming

In this work, the ANNs are encoded as in the CGP-ANN algorithm [Khan et al., 2010; Miller and Thomson, 2000; Turner and Miller, 2013]. Each individual consists of a fixed number of nodes, as visualized in Figure 8.1. Besides the input nodes, which represent the data inputs, each node has a single transfer function, a fixed number of inputs, and associated weights based on their arity. Nodes are always connected to proceeding nodes, and multiple connections to the same node are possible. Only those nodes which are directly or indirectly connected to an output are evaluated during a run, while all other remain

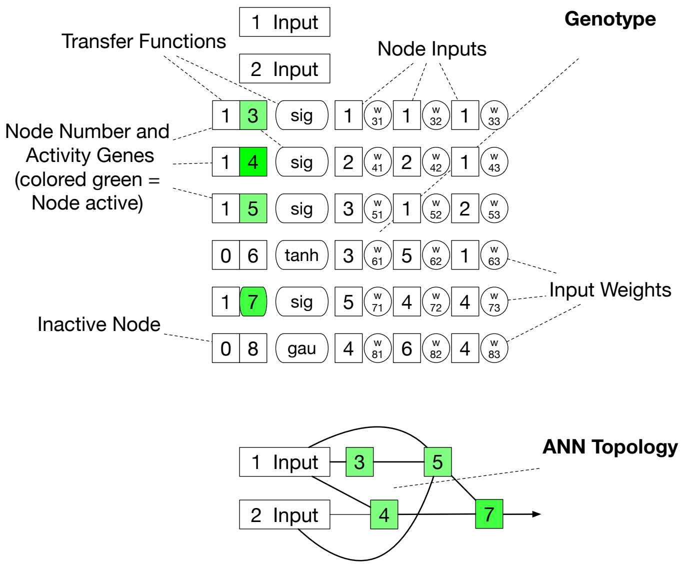


Figure 8.1: *A CGPANN genotype with two inputs, eight nodes, an arity of three and different transfer functions. Each node has a transfer function, a boolean activity gene and several inputs with adjacent weights. Green nodes are active and part of the encoded ANN. In the related topology only active nodes are included and duplicate connections are aggregated. Taken from [Stork et al., 2019a].*

passive and do not influence the behavior of a network. Thus, very small active ANN topologies and also those only using specific inputs are possible, even if the genotype has numerous nodes. In CGP-ANN, the networks are optimized using mutation, following the concept of an evolutionary strategy. A typical choice is a (1+4)-ES, whereby the elitist is always the current best individual (e.g., if an individual achieves the same fitness as the elitist during evolution, the more recent is selected). The C library `CGP` by A. Turner[2], extended by interfaces to R, was used to perform the experiments.

[2]http://www.cgplibrary.co.uk - accessed: 2018-01-12

8.2.2 Kriging for modeling ANNs

In this study, we focus on an SMBO approach that employs Kriging (Gaussian process regression) [Forrester et al., 2008]. The main question in this context is how Kriging can model the complex dependencies between a neural network's topology and its fitness.

At its core, Kriging is based on kernels such as the exponential kernel $\mathrm{k}(x, x') = \exp(-\sum_{i=1}^{n} \theta_i (x_i - x'_i)^2)$. In this example, $x \in R^n$ is a vector of real values, and $\theta_i \in R^+$ is a non-negative parameter of the kernel. If x is not a real-valued vector but rather represents a candidate ANN, we need to change the kernel such that it compares networks rather than vectors. For instance, the weighted distance measure $-\sum_{i=1}^{n} \theta_i (x_i - x'_i)^2$ may be replaced with some distance between ANNs. To that end, previous work suggested evaluating distances that are based on observations of network behavior (or phenotypes) [Hildebrandt and Branke, 2015; Zaefferer et al., 2018; Stork et al., 2019a]. More details on the computation of phenotypic distances are given in Section 8.2.3.

One complication of using such phenotypic distances in Kriging is dimensionality. Specifically, Kriging is often suggested for problems with less than 20 variables (e.g., see Table 3.1 in [Forrester et al., 2008]). At the same time, the vectors of phenotypes we consider in the context of ANNs may quickly grow to lengths of 100 or more elements. Hence, the combination of Kriging and phenotypic distances may appear to be a poor choice.

We propose to tackle this in two manners, each related to two different aspects that affect problems with high-dimensionality in Kriging. One problem of high-dimensional data is the determination of the kernel parameters such as θ_i. These are usually determined by maximum likelihood estimation, using numerical optimization algorithms [Forrester et al., 2008]. In MLE, the parameters are chosen to maximize the likelihood determined by the Kriging model.

Clearly, the number of parameters increases with the dimensionality of the data. Optimizing many parameters by numerical optimization may pose a tough problem. A straightforward fix is to set all parameters θ_i to the same value, that is, to use $\mathrm{k}(x, x') = \exp(-\theta \sum_{i=1}^{n} (x_i - x'_i)^2)$, where only a single parameter θ has to be determined by MLE. That is, we choose an isotropic rather than anisotropic model.

A second problem is the behavior of distances in high-dimensional spaces. Roughly speaking, when measuring Euclidean distances of different data points in a high dimensional space, nearly all points will have the same distance [Aggarwal et al., 2001]. This is undesirable for any model that is based on such distances. Aggarwal et al. [2001] state that other distances are less affected by this issue, especially the Manhattan distance (based on the L_1 norm). For that reason, we finally propose to use an isotropic kernel based on the Manhattan distance to measure the distance between behavior vectors, i.e.,

$$\mathrm{k}_{\mathrm{BD}}(x, x') = \exp(-\theta \sum_{i=1}^{n} |x_i - x'_i|). \tag{8.1}$$

Figure 8.2 illustrates an example model for a one-dimensional case. Besides dimensionality, another essential aspect of kernels for Kriging is their definiteness. Usually, kernels are required to be positive semi-definite. The kernel k_{BD} from Eq. (8.1) is definite, as it is a special case of the positive semi-definite Gaussian kernel [Forrester et al., 2008].

8.2.3 Behavioral Distance Measure for ANN Topologies

Evolved ANN topologies do not have fixed structures regarding hidden layers, weights, connections, or functions. Measuring the distance between these complex structures is thus a challenging task. In detail, it is difficult to measure a distance directly on these structures due to several problems:

P-1 *Competing Conventions:* A famous problem that arises in the context of ANNs are competing conventions [Schaffer et al., 1992], i.e., different genotypes can result in the same topology, as well as the same phenotype.

P-2 *Incomparability:* Even ANNs with fixed genotypic structures (e.g., as produced by CGP) are often not directly comparable. When comparing

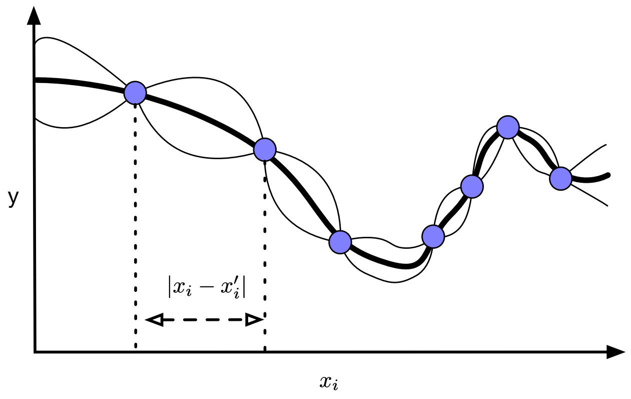


Figure 8.2: *Example for Kriging modeling: distances between ANNs (blue circles) in one dimension x_i of a behavior output. The bold line is the model prediction, while the thin lines display the uncertainty of the model.*

two genotypes, elements such as specific nodes maybe not be aligned in the same way, despite having the same effect on the outcome. In other words, it is not always straightforward to decide which pairs of nodes should be aligned with each other when comparing two different networks. This could be handled by complex and computationally expensive sorting and aligning processes, but this would pose an optimization problem in itself and render the comparison computationally expensive [Stork et al., 2019a]. This issue becomes more problematic with the increasing size of the ANNs.

P-3 *Lack of Smoothness:* In some cases, small changes in the genotype can have a significant effect on the final behavior. For instance, removing a single connection may change the fitness of the network dramatically. That means small distances in the search space may lead to large distances between fitness values. Essentially, this implies that the search space is not smooth under a genotypic distance. This presents a severe problem to every optimization or modeling algorithm.

P-4 *Distance Balancing:* Different types of changes in a network have different meanings and impacts. For example, changing a weight has a different impact than changing the transfer function of a node. It is thus difficult to provide a meaningful distance that correctly balances (or weights) such changes. For example, it is unclear whether two networks that only differ in a single weight are at the same distance as two networks that only differ in a single transfer function.

Due to these issues, genotypic distances do not seem to be ideal for the computation of high-performing surrogate models in SMB-NE. Thus, we follow the idea of comparing the behavior of ANNs and employ a behavioral distance for modeling the dependencies of ANNs [Hildebrandt and Branke, 2015; Zaefferer et al., 2018; Stork et al., 2019a]. In the context of ANNs, we consider the reaction (output) of an ANN to an input signal to be its behavior or phenotype. In detail, we compute the BD by first selecting a representative input vector for a given task. Then, these inputs are fed into the ANN, and we observe a vector of output values. In terms of RL, the input vectors are typically vectors of consecutive observed environment states. The observed outputs are then directly compared via the kernel described at the end of Section 8.2.2.

The clear advantage of the BD is that the output length and structure are not dependent on the genotype or topology of the compared ANNs. The computation of the input to output mapping can be performed independently of different structures (e.g., it does not matter whether the network has 10 or 1000 active connections or different transfer functions). Moreover, the observed outputs of the BD give a clear impression of how the networks react and explicitly account for granular changes in how ANNs differ in solving a task.

The potential disadvantages of the BD are the computation times for generating the output vectors. For complex, large ANNs topologies (genotype size does matter much less), they can sum up to a significant amount. This issue is not significant if we consider the task itself to be computationally expensive, especially for expensive simulators or even real-time experiments. In this study, we thus concentrate on enhancing the sampling efficiency and not the overall computation time, which is strongly related to solving a specific task.

8.2.4 Behavioral Distances: Input Vector Selection

The BD is, in contrast to genotypic distances, strongly task and environment-dependent. One critical aspect of employing phenotypic distances is thus that their design needs to be adapted to each specific application. For our purpose of modeling ANNs in RL, we first need to define how the input of the networks is chosen. These inputs can then be used to generate the output of the ANNs, which will be compared by the distance measure.

Choosing the input vectors involves multiple issues, such as distribution of the state data, size of the data set, and many more. In this paper, we focus on four different types of input vectors:

– **Precomputed set (Pre):** As a comparison baseline, we use the set of states that are observed by optimal or near-optimal solutions. A number of state vectors from previously successful runs are stored and used as a precomputed input set. This is an artificial baseline for our outlined scenario (S-1), where the knowledge about these states is initially unavailable. This could be seen as a best-case scenario.

– **Static initial set (Init):** We extract the observation vectors $s_{t=1}$ from the initial dataset $D_{t=1}$ to form the input vector $v_{t=1}$, which is not altered during the run. The possible downside of this approach is that the initial, randomly generated solutions typically have poor fitness. Inadequate solutions may see quite different states than successful, near-optimal solutions. They may cover only a tiny subset of all possible observable states. As these initial input vectors are thus not representative, the resulting distance may not help predict reasonable solutions.

– **Sampling set (LHS):** Furthermore, input vectors can also be determined by the design of experiment methods, such as Latin hypercube sampling [McKay et al., 1979]. The idea is to distribute the data in a space-filling manner between known bounds for the inputs. In contrast to the initial static approach, these inputs cover the whole state space; they are artificial and do not represent the observed states of actual runs.

- **Dynamic set (Dyn):** Finally, we may update v_t at each iteration if the SMBO algorithm finds a new best ANN. The observed states s_t of this new solution will replace the worst in the input vector v_t. This implies that the input vectors are changed dynamically over time and may approximate the baseline if the algorithm converges to the optimum. This also means that the employed distance measure changes in each iteration. Models trained in different iterations of the algorithm are hence not directly comparable.

Further considerations include the number of observations in the input vector and the maximum size of the overall input vector. Large size for v_t increases the computation time for generating the phenotypes of the ANNs, and further may introduce problems with too high dimensionality (Section 8.2.2). On the other hand, a larger input vector including the states of different runs may lead to a more representative behavior and thus distance. Another challenge is the over-time changing sizes of the dynamic input set.

8.3 Surrogate Model-based Neuroevolution for Reinforcement Learning

In this work, we employ SMB-NE, which was first introduced by Stork et al. [2018, 2019a] and tested by evolving neural networks for classification problems. The SMB-NE algorithm follows the principles of efficient global optimization [Jones et al., 1998], which was introduced in the context of expensive real-world optimization problems. In the case of SMB-NE for RL, we require additional steps due to the complex ANN structures. The complete procedure is outlined in Figure 8.3 and Algorithm 8.3.1 . The detailed steps of the algorithm are:

Initialization of model set: The algorithm starts by sampling an initial set with k candidates $D_t = \{(x_{1:k}, y_{1:k})\}$. The initial set is generated entirely at random, so the included genotypes can differ in their size, the number of genes, connections, functions, and weights. The active ANN topologies are then compiled with CGP to be evaluated with the underlying RL task.

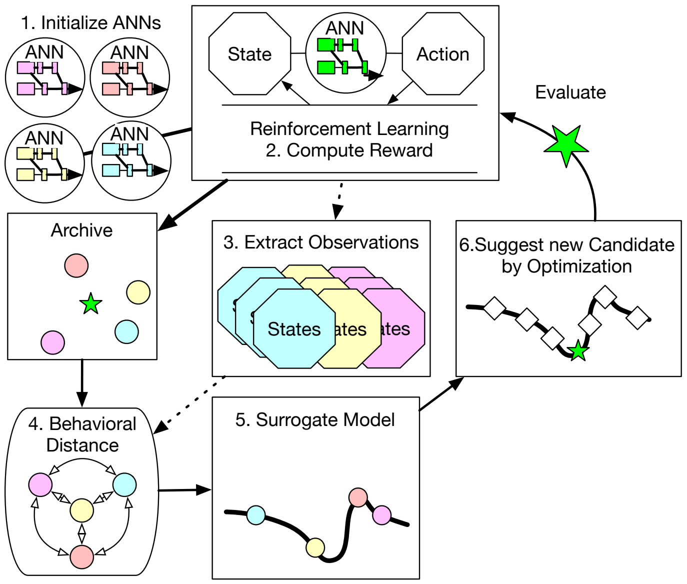


Figure 8.3: *SMB-NE Cycle for Reinforcement Learning*

Evaluation with RL task: Each ANN is evaluated over several time steps, which are defined by the RL task. In each time step, the ANN computes actions based on the currently observed state inputs. Each action is then given a reward. Some actions lead to a negative reward, such as a robot bumping in a wall or expending some resources. Positive rewards are given for accomplishing a specific goal. The fitness of a so-called episode is typically the sum of all rewards over the executed time steps. The fitness information is thus limited, as it includes only the final reward of an episode and does not reveal which single action was beneficial or not.

Extraction of observed states: If the input vector v_t is not given upfront or precomputed by DOE methods, it needs to be extracted from the RL experiment. For each experiment, all observed system states are stored in a vector s_t. The set of state observation vectors is sorted according to the determined fitness values for each experiment. From this set, the best num_{s_t} observation vectors (according to the fitness of the respective ANN) are selected.

They are combined in the order of their fitness to form a single input vector v_t with length $\text{len}_{v_t} = \text{num}_{s_t} * \text{len}_{s_t}$. If the observation vector length is beyond a specific size, a subset of each s_t is selected before combining them to v_t. This intends to keep the number of elements in the vector from becoming too large.

Kriging model construction: The Kriging model is constructed as described in Section 8.2.2. We utilize the R-package `CEGO` [Zaefferer, 2017] to train the Kriging model. At the start of the process, the model is trained with the initial set of solutions $D_{t=1}$. The state input vector v_t is used to compute the phenotype of each ANN, which is required to calculate the BD. A subset M_t is selected from D_t to build the model in the later modeling steps. This subset selection intends to avoid issues with growing data sizes, which may render the Kriging model too time-consuming to compute. This set M_t contains num_m of all archived solutions. It is typically set to $\text{num}_m > 100$, so for runs with fewer than 100 evaluations it has no effect. M_t is formed by combining a number (typically $\frac{1}{5} * \text{num}_m$) of the best-found solutions, with the rest being sampled at random from the archive (without replacement, thus duplicates are not possible). This process further influences the balance between exploration and exploitation, as, in each iteration, a different set of ANNs is considered for the model construction.

Surrogate Optimization: The sequential optimization steps are conducted by optimizing the expected improvement of the surrogate model to suggest new promising ANNs. The EI criterion delivers a balance between the predicted fitness and the uncertainty of a solution, leading to a balance between exploration and exploitation in den model-based search [Jones et al., 1998]. We utilize the same (1+4)-ES of CGP-ANN to generate new candidates for the model optimization. To predict the fitness of new candidates, their BD needs to be computed, which requires their ANN outputs, based on the selected input vector v_t. The identified candidate with the highest EI on the surrogate model is again evaluated with an RL run and added to the archive D_t.

Dynamic state vector update (optional): If the *dynamic* strategy for the input vector v_t is chosen, it is updated if the new candidate solution has a better fitness than the best-known solution. During this update, the observed states s_t of the related RL run replace the ones of the worst candidate solutions in the input vector v_t.

Cycle: If the stopping criterion is not met, the next iteration is started.

Algorithm 8.3.1: SMB-NE for Reinforcement Learning

```
1  begin
2      $t = 1$
3      initialize $k$ CGP genotypes $(x_i)$ at random
4      evaluate their fitness with the objective function to get initial solutions
         $D_t = \{(x_{1:k}, y_{1:k})\}$
5      extract state vectors to create BD input vector $v_t$
6      build Kriging surrogate model $m_t$ with $D_t$ using input vector $v_t$ to
         compute ANN behaviors for BD ;
7      while not termination-condition do
8          if $t > 1$ then
9              rebuild surrogate model $m_t$ with selected subset $M_t \subseteq D_t$
10         end
11         optimize EI estimated by $s_t$ with evolution strategy to discover
             promising $x_{t+1}$
12         evaluate network $x_{t+1}$ with the objective function
13         if $y_{t+1} < y_t$ then
14             update input vector $v_{t+1}$ with states of successful run (dynamic
                 input vector)
15         end
16         update archive $D_{t+1} = \{D_t, (x_{t+1}, y_{t+1})\}$
17         $t = t + 1$
18     end
19 end
```

8.4 Experiments

We chose two problems from the OpenAI Gym toolkit as a benchmark for our experiments because they are well-known in the community. Specifically, we chose the classic RL problems *CartPole-v1* and *MountainCar-v0*, displayed in Figure 8.4.. They are implemented in python, and the `reticulate` package in R was used to create an interface between SMB-NE and openAI Gym.

8.4.1 OpenAi Gym Benchmarks

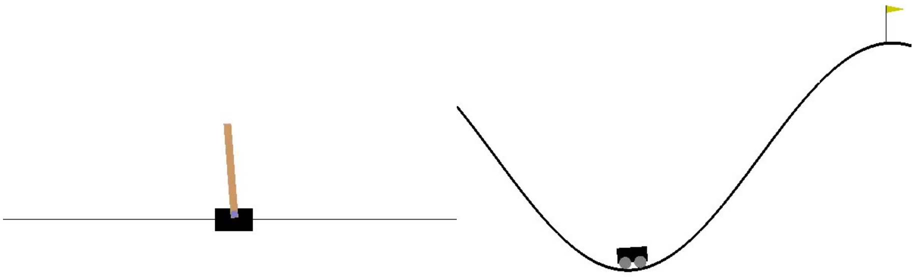


Figure 8.4: *OpenAI Gym CartPole-v1 and MountainCar-v0*

The *CartPole-v1* environment is a classic cart-pole balancing problem, where a pole is placed with an un-actuated joint to a cart moving on a frictionless track. It has four observations per state, the cart position, cart velocity, pole angle, and the pole velocity at its tip. Based on these observations, two discrete actions can be chosen by a controller, either pushing the car to the left or the right. The goal is to keep the pole balanced and the cart near the center of the track. Each episode of the environment is evaluated over 200 time steps and terminated if the pole or cart moves out of pre-defined ranges. For each time step, a reward of 1 is given, and the environment is considered solved if an average reward of 195 is achieved per episode over 100 trials.

In the *MountainCar-v0* environment, a car situated between two hills on a one-dimensional track has to be driven up a mountain, whereby the acceleration of the car is not strong enough to drive up directly. Thus, a swinging forward and backward behavior is needed to succeed. The observation space consists of only two variables, the current position, and velocity. The action space has three discrete options: drive left, do nothing and drive right. Again, an environment episode is run over 200 steps but terminated if the goal is reached. A negative reward -1 is given for each step, and the environment is considered solved if a reward larger than -110 is achieved over 100 trials.

While for CartPole the fitness function is set to direct negative reward (as we utilize minimization during optimization), the MountainCar fitness function is altered for the optimization to a mixture of achieved maximum height (maxHeight) and reward by

$$\text{fitness} : y(x) = -(\max \text{Height}_{episode} + \frac{\text{Reward}_{episode}}{100}) \tag{8.2}$$

This modification was chosen to compute a more granular fitness. Without this, most initial solutions get the worst reward. An initial data-set where nearly all solutions have the same poor fitness would be detrimental for training a surrogate model. The stopping criterion remains unchanged.

8.4.2 Parameter Tuning and Setup

Due to the considerable runtimes, we were not able to perform exhaustive tuning of the parameter space for CGP-ANN and SMB-NE but conducted some preliminary tests to acquire information about the algorithm parameter space and the significance of specific variables. For CGP-ANN, two mutation operators (*single active mutation* and *random mutation*), as well as different mutation strengths, were tested. The preliminary tests have shown that CGP-ANN with a *single active mutation*, wherein each iteration the genotype is mutated until at least a single active genome (and an arbitrary number of non-active genomes) is altered, was not able to deliver competitive performance. Moreover, the choice of the mutation strength in random mutation has a significant impact on the performance. Thus, we decided to conduct experiments with different mutation strengths of two, five, and ten percent, which were selected based on former experiments to show the influence of this parameter and conduct realistic comparisons. SMB-NE includes an even more extensive set of parameters, such as the choice of input vectors (for the BD), their number and dimensions, and an optimizer and its parameters during the surrogate model optimization. Thus, most of these parameters were set by the authors' experience and the small set of preliminary tests. We identified that the number of different observation vectors num_s for creating the input vector v_t might be a crucial tuning factor

Table 8.1: *Algorithm Parameter Setup for the Experiments*

Problem	**Weight Range**	**Nodes**	**Arity**
CPole/MCar	[-1,1]	200/100	20/10
CGP-RS		**Max Episodes**	
CPole/MCar		3000/5000	
CGP-ANN	**Mutation rate**	**Max Episodes**	
CPole	2/5/10	$20 + 750 \cdot 4$	
MCar	2/5/10	$20 + 1250 \cdot 4$	
SMB-NE	**BD Input Sets**	**Max Episodes**	**Surr Evals**
CPole	Pre, Init, LHS	20 + 3000	1000 per iter
CPole	Dyn: $\text{num}_s = 2/5/10$	20 + 3000	1000 per iter
MCar	Pre, Init, LHS	20 + 5000	1000 per iter
MCar	Dyn: $\text{num}_s = 2/5/10$	20 + 5000	1000 per iter
MCar	*Dyn***, $\text{num}_s = 5$	*20 + 5000*	*4000 per iter*

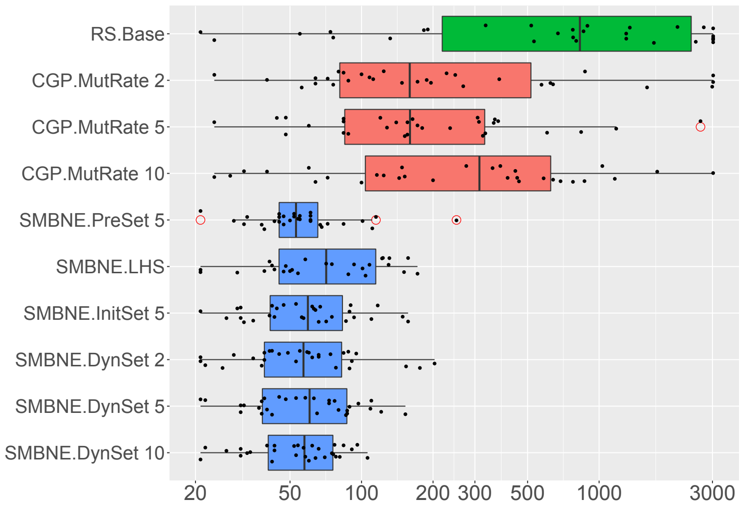
and thus added different variants to the experiments. Although we expect that the chosen parameters for both algorithms do not reflect the best possible options, they should still provide valuable insights on the performance level of both algorithms. Table 8.1 shows the parameter setup for the benchmarks. CGP-ANN genotypes for CartPole/MountainCar are set to an arity of 20 or 10, with 200 or 100 nodes, resulting in up to 4000 or 1000 connections between nodes. We perform all tests with a large set of activation functions: tanh, softsign, step, sigmoid, and gauss. Both the maximum size of the genome and the function were set by the authors' experience. This displays a typical scenario in NE, where we do not know upfront which size for the genotypes is best.

All inputs are, if possible, normalized to the [-1,+1] range, and the connection weight range was also set to [-1,+1]. The setup considers a maximum runtime of 3000 or 5000 episodes, whereby the run is stopped as soon as the stopping criterion (environment solved) is met. The size of the initial data set $D_{t=1}$ is set to 20 for all algorithms. CGP-ANN starts the normal evolution with the best-found solution of the initial set and computes four candidates per iteration. SMB-NE selects a single new candidate per iteration and utilizes 1000 search steps for the model optimization. All SMB-NE setups use the same mutation rate during the model search (5%). The tested setups include all variants introduced in Section 8.2.4 (*Pre, LHS, Init* and *Dyn*). For the dynamical approach, different numbers of observation vectors num_s to create the input vector v_t are tested. The input vectors generated by LHS are based on the (theoretical) bounds of the state observations for each environment and have 800 elements. All experiments are repeated 30 times with different random number generator seeds. Different algorithms/configurations are tested with the same set of seeds to be comparable. A CGP-ANN configuration that only generates random solutions is included in the experiments as a baseline (RS). For assessing the performance of an exhaustive model search, we test an additional variant (Dyn**) of SMB-NE with the dynamic set for MountainCar-v0, where we set the size of the surrogate model evaluations to 4000. Due to the computational effort, this variant is only repeated 20 times. The statistical significance of the observed differences is evaluated using the Kruskal-Wallis rank sum test [Kruskal and Wallis, 1952] and a posthoc test for multiple pairwise comparisons according to Conover and Iman [1979].

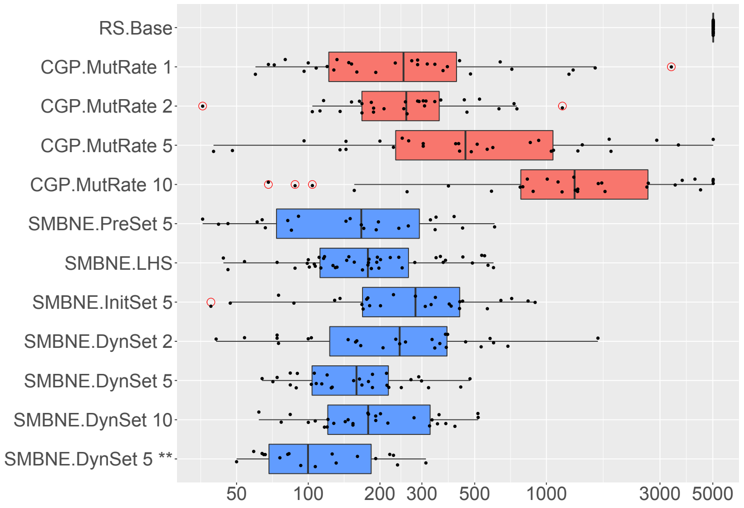
8.5 Results and Discussion

Figure 8.5 and Table 8.2 show the results of all conducted experiments with both benchmark problems. For easier comparison, the results of the box plots are log10 scaled and colored according to the type of algorithm. The numbers indicate either the utilized CGP-ANN mutation rate in percent or the number of utilized state observation vectors num_s for computing the BD in SMB-NE.

CartPole-v1: For CartPole-v1, all algorithms are able to find successful solutions but show a high variation in solution quality over the different random seeds. This variation relates to the different starting conditions for the RL environment and different initial data sets. On the one hand, the environment might rarely be solved by pure random chance during the initialization of the algorithms. On the other hand, even CGP-ANN sometimes fails to find solutions within the specified budget of 3020 total fitness function evaluations. In contrast, all SMB-NE variants can discover ANNs that solve these environments within the given budget. The statistical tests indicate an overall significance of the results (Kruskal-Wallis rank-sum test). However, no evidence for significant



CartPole-v1



MountainCar-v0

Figure 8.5: *Experimental results. The number of required function evaluations (episodes) to solve the environments is Log10 scaled. Algorithms have different colors and specific setups are attached to the algorithm names. The numbers indicate either the utilized mutation rate in percent (CGP-ANN) or the number of utilized state observation vectors num_s (SMB-NE). Red circles depict outliers.*

differences between any CGP variant and random search is discovered by the statistical test procedure (posthoc), while all SMB-NE variants are evaluated to be different from CGP and random search. There is insufficient evidence to indicate a significant difference between the tested input sets for SMB-NE, according to the respective posthoc test. The results show that the best tested SMB-NE variants can outperform the best tested CGP-ANN variant and require about 70% (median) or 80% (mean) fewer function evaluations (or environment episodes).

Table 8.2: *Result tables for both environments, reported mean and standard deviation, sorted by CartPole-v1 ranking*

Algorithm	Setup	Evaluations (Required Episodes) ± sd	
		CartPole-v1	MountainCar-v0
SMBNE	DynSet 5 **	not tested	*130.67 ± 76.90*
SMBNE	DynSet 10	**57.57 ± 22.79**	218.96 ± 129.70
SMBNE	PreSet 5	63.17 ± 41.88	198.70 ± 152.10
SMBNE	DynSet 5	64.83 ± 32.43	**179.40 ± 105.05**
SMBNE	InitSet 5	64.97 ± 34.43	327.22 ± 235.47
SMBNE	DynSet 2	66.67 ± 43.98	320.67 ± 240.68
SMBNE	LHS	80.41 ± 44.19	219.05 ± 152.43
CGP	MutRate 5	**328.00 ± 508.28**	916.13 ± 1146.07
CGP	MutRate 10	487.20 ± 626.35	1830.40 ± 1612.21
CGP	MutRate 2	541.73 ± 897.50	**320.67 ± 240.68**
CGP	MutRate 1	not tested	462.13 ± 667.98
RS	Base	**1271.07 ± 1139.16**	**5020.00 ± 0.00**

MountainCar-v0: As the results indicate, the MountainCar-v0 is more challenging to solve, and CGP with random search cannot discover a single valid solution. Overall, more evaluations are required to solve the task. The mutation rate in CGP-ANN has a noticeable influence on the performance. The tested configuration with a mutation rate of 2% performed best. Based on these results, additional runs with even smaller mutation rates were conducted (1% is reported) but showed no improvements. For the sake of brevity, they are not shown in the result plots. Again, the Kruskal-Wallis rank-sum test indicates that significant differences are present. The posthoc test shows that all algorithms performed statistically different from random s earch (except for CGP with 10%

mutation rate). Only the SMB-NE *DynSet*** variant, which features a more extensive surrogate model search, shows evidence for a significant difference to CGP-ANN. From the input sets, the *Init* set variant performed worst. Still, the best standard dynamic variant *DynSet 5* requires 45% fewer evaluations than CGP-ANN and the dynamic variant *DynSet*** performs even better (60-70% less required function evaluations).

Discussion The presented experimental results provide substantial insights on the performance potential of utilizing model-based search in NE. For both test cases, SMB-NE outperforms the basic (1+4)-ES integrated into CGP-ANN. First, we focus on the results of the model-free CGP-ANN with the (1+4)-ES. Particularly for MountainCar-v0, significant performance differences in the choice of mutation rate are visible. This shows the need for either exhaustive tuning of this parameter or the development of an adaptive strategy.

Secondly, no apparent statistically significant differences were observed for the different choices of how the state input vector for the BD distance is generated. Thus, we cannot support the different assumptions raised in the introduction of the input sets (Pre, Init, LHS, and Dyn). Given the current experimental design with large input vectors, we can state that SMB-NE is reasonably robust to the choice of the input vector. However, the MountainCar-v0 results show a slight preference towards the dynamic input sets. Thus we still assume that it is preferable if the computed distance is based on state vectors that were actually observed rather than artificially created or precomputed. Especially if a further dimension reduction of the BD is considered, the dynamic input vector thus seems the best choice.

In comparison to CGP-ANN, SMB-NE is, in general, able to produce more stable results, rendering it a promising choice for new tasks, as in the outlined scenario (S-1). For example, Figure 8.6 shows the convergence of the SMB-NE using a dynamic input vector with $\text{num}_s = 5$ in comparison to the CGP-ANN with a mutation rate of 5%. The mean reward over the 30 repeats of the current candidate is shown for each iteration. As can be observed, the model-based search shows a substantial increase in reward after the initial set and then a steady convergence, while CGP-ANN also improves steady but slower.

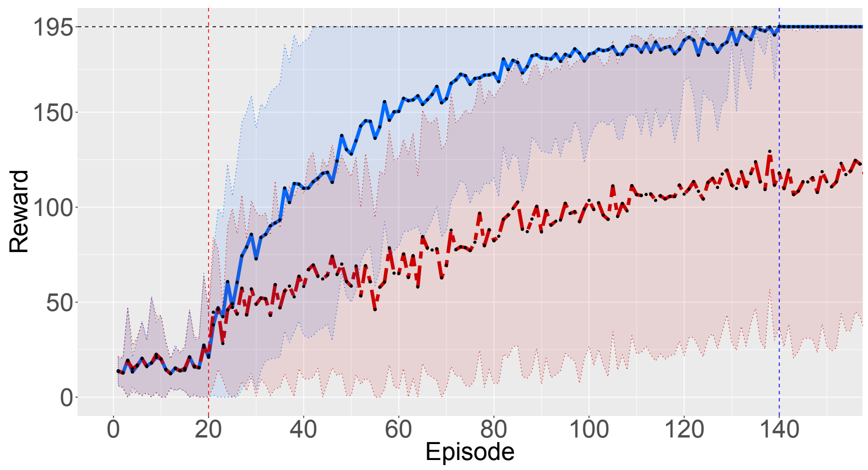


Figure 8.6: *Convergence plot of SMBNE.DynSet 10 (solid blue) and CGP.MutRate 5 (dashed red) on CartPole-v1, mean reward of current candidates (not best solution) aggregated over repeats with standard deviation (colored areas), the environment is solved by reaching a reward of 195 per episode.*

8.6 Conclusion and Outlook

In this work, we investigated how a surrogate model-based search can be utilized to enhance the efficiency of NE, given the complex task of evolving artificial neural networks for reinforcement learning. Our surrogate models are based on phenotypic distance measures, which utilize the observed differences in the outputs of an ANN. We discovered that our SMB-NE for RL could significantly outperform a model-free evolutionary strategy, which answers the initially raised research question Q-II. We proposed different approaches to generating state vectors for the ANN's input space regarding Q-I and Q-III. The current empirical results do not provide evidence for solid differences between the input sets generation methods. SMB-NE thus seems relatively robust towards this choice. Still, we regard the dynamic input sets as the most promising approach.

Of course, this work and the results raised further questions. The first is how to set the parameters of the SMB-NE algorithm optimally, particularly regarding the dimension of the input sets. Up to now, we do not know which length of the state vector is required to generate a well-performing model and, thus, reasonable optimization performance. The length of the state vector is also related to the computation costs, particularly for computing the BD measure. The computation costs are a potential drawback of SMB-NE, but the clear and robust improvements of the sampling efficiency render it notably attractive for tasks where the fitness evaluations themselves are costly. For instance, consider a robot controlled by an ANN. Testing that robot in a real environment may be very expensive, while computing only the outputs of the ANN are considerably cheaper. In ongoing work, we will furthermore attempt to generalize our results to more environments from the Gym toolkit as well as tests with real-world problems. We plan to investigate the underlying mechanics. In particular, the significance of specific algorithm parameters is of interest and may require more attention to algorithm tuning.

9 Analysis of the Behavioral Space in Context of Reinforcement Learning

Chapter 9 was published as:

Stork J, Zaefferer M, Bartz-Beielstein T, & Eiben AE (2020). Understanding the behavior of reinforcement learning agents. *In Bioinspired Optimization Methods and Their Applications: 9th International Conference*, (pp. 148-160). Springer Nature, Cham.

Reinforcement learning is the process of training agents to solve specific tasks, based on measures of reward. Understanding the behavior of an agent in its environment can be crucial. For instance, if users understand why specific agents fail at a task, they might be able to define better reward functions, to steer the agents' development in the right direction. Understandability also empowers decisions for agent deployment. If we know why the controller of an autonomous car fails or excels in specific traffic situations, we can make better decisions on whether/when to use them in practice. We aim to facilitate the understandability of RL. To that end, we investigate and observe the behavioral space: the set of actions of an agent observed for a set of input states. Consecutively, we develop distance or similarity measures in that space and analyze how agents compare in their behavior. Moreover, we investigate which states and actions are critical for a task, and determine the correlation between reward and behavior. We utilize two basic RL environments to investigate our measures. The results showcase the high potential of inspecting an agents' behavior and comparing their distance in behavior space.

9.1 Introduction

In reinforcement learning, agents are learning policies to solve a specific task. For example, we can consider a robot as an agent who has to navigate a particular environment and react to certain obstacles. At first, a user is interested in these robots' performance, which is commonly evaluated by their ability to solve the task and further based on a user-defined reward function. Besides this performance assessment, the trained robot's behavior, such as the action it takes for individual states, is the only observable part, as the internals of the policy remains indistinguishable by an external observer. Thus, users desire to analyze and compare the behavior to exploit how the robot reacts in certain situations or if it behaves as intended. Even a well-performing robot may have developed a specialized behavior not intended by the user, such as only driving backward. This chapter compares agents based on their behaviors, which span a new space, the behavior space. This chapter's primary motivation is to create a better understanding of this behavior space and develop valuable measures for the comparisons of agents without knowing the inner details of their policies. Moreover, these measures could allow us to identify how agents in a learning set differ, not concerning their reward, but with regard to their behavior. It is particularly interesting to identify situations (states) in which an agent behaves differently than expected. As this is a broad topic, we will start by tackling the following research questions:

Q-I How does an agent's behavior with good performance compare to similarly performing agents or inferior agents?

Q-II Which input states are important or problematic for the task?

Q-III Is there a correlation between an agent's reward and behavior, and how do changes in the behavior affect their reward?

Comparing agents in the behavior space has some prerequisites: Individual agents will not visit the entire state space for most RL environments and thus not learn the optimal action for these unobserved states. Unobserved states are, for instance, present in environments with continuous state spaces or exclusive paths. Nevertheless, as we investigate agents that map a policy from state to action space (i.e., artificial neural network policy controllers), we can compute an agent's behavior to any state, even if not observed or observable by the agent itself. This property allows us to compare two agents in the behavior space on a mutual state set and investigate differences. However, state sets are usually not initially known but based on processing the RL tasks and discovered during each agent's learning process. Thus, the individual state sets' contents are based on the state trajectory each agent follows, for example, an absolute path for a robot in a maze. Each agent in a learning set will likely have different trajectories, which renders it challenging to select input states to compute a proper behavior space to compare many agents.

Behavior spaces have previously been investigated in the RL literature. Most frequently, they were utilized to measure diversity and enforce explorative search strategies. For instance, Doncieux and Mouret [2010] used behavioral similarity measures to encourage the diversity of evolved agents in an evolutionary search. Ollion and Doncieux [2011] suggested to measure and enforce exploration in the behavioral space. Meyerson et al. [2016] investigated how behavior characterizations can be learned automatically for novelty search. Quality diversity algorithms also depend on effective behavior comparison [Pugh et al., 2016]. Similar directions have been investigated in the field of surrogate model-based optimization. Here, the term *phenotypic* space has been used, defining a space that encompasses behaviors and outcomes of individuals rather than their encoding (genotype). Distances in the phenotypic space are used to train surrogate models. For instance, this has been investigated in the context of tree-coded genetic programming [Hildebrandt and Branke, 2015; Nguyen et al., 2016; Zaefferer et al., 2018]. Similar work focused on graph-coded representations of neural networks. Here, phenotypic spaces and distance measures have been investigated for tasks like classification, reinforcement learning, or for evolving neural network controllers for robotic navigation [Stork et al., 2019b; Hagg et al., 2019]. Unlike these previous investigations, we aim to look at the behavior space not primarily to improve the performance of optimization or modeling algorithms. Instead, we aim for the understandability of agents' behavior. To do so, we utilize two RL environments, a designed maze with different mutual exclusive paths and the inverted pendulum, with a large real-valued state space.

9.2 Methods for Analyzing Behavior

9.2.1 Behavior Space in Reinforcement Learning

The *behavior* of an RL agent encompasses its (re-)actions based on its environment and observed input states. The actions an agent takes for a specific state $s \in S$ is defined by a *policy* $\pi : S \to B$. The agents get a reward $r \in R$ for each state transition. The discussed methods apply to a wide range of RL agents. The only prerequisite is their ability to calculate a behavior for states that those agents themselves have not observed. More precisely, we define behavior as the set of actions for a set of input states. For an agent A, we denote its behavior as B_A, with $B_A = \pi_A(\mathbf{S})$. Here, $\mathbf{S}$ is a set containing n input state vectors, $\pi_A(\mathbf{S})$ is the policy function computing the actions of agent A for all states in $\mathbf{S}$. Consequently, the behavior space $\mathcal{B}$ is the set of all possible behaviors (or the behavior of all possible agents) for an RL task, that is, $B_A \in \mathcal{B}$.

9.2.2 Behavior Comparison and State Importance

For the comparison of two agents A and A', we can calculate the distance of their behaviors, which can then be denoted by $\mathrm{d}(B_A, B_{A'})$. Because the distance depends on the state space, we consider the distance of two behaviors concerning the same state set $\mathbf{S}$. The employed distance function can be chosen according to the data type of $B_A, B_{A'}$. That is, if they contain continuous values, we might use the Euclidean distance. If they are ordinal integers, we can choose the Manhattan distance instead, with $\mathrm{d}(B_A, B_{A'}) = \sum_i^n |\pi_A(s_i) - \pi_{A'}(s_i)|$. The comparison of actions for individual states can provide interesting insights into the specific behavior of an agent and further the importance of that state for the task. In particular, we analyze the effects of unobserved states (UOS), which are not present in the state set of a specific agent, and the influence of states with degrees of freedom (DFS), where several actions lead to the same or similar reward. In general, we consider a state as important (or problematic) if the correct action for this state is essential for getting a good reward (or challenging to learn, e.g., a majority of agents in a learning set fails to learn the correct action). For the impact of states on the reward, we utilize the *action reward rank*: All performed actions of the agents are compared for each state, and the best ranking agent who took this action is outlined. Hence, this action is related to the final best-performing agent in the set.

9.2.3 Reward Behavior Correlation

To understand the benefits of comparisons in the behavior space, the correlation between reward distance and behavior distance is attractive. Therefore, we investigate a set of m agents $\{A_1, ..., A_m\}$, their behaviors $\{B_{A_1}, ..., B_{A_m}\}$, and their accumulated rewards $\{R_{A_1}, ..., R_{A_m}\}$. We compute the reward behavior correlation (RBC) for all pairwise comparisons:

$$\mathrm{RBC}_{\mathrm{all}} = \mathrm{cor}\Big(\mathrm{d}\big(\{B_{A_1}, ..., B_{A_m}\}\big), \mathrm{d}\big(\{R_{A_1}, ..., R_{A_m}\}\big)\Big).$$

Here, $\mathrm{d}\big(\{B_{A_1}, ..., B_{A_m}\}\big)$ calculates all pairwise distances of the present agents using the behavioral distance $\mathrm{d}(B_{A_i}, B_{A_j})$. Correspondingly, $\mathrm{d}\big(\{R_{A_1}, ..., R_{A_m}\}\big)$ calculates all pairwise distances of the present agents using a distance of their accumulated rewards $\mathrm{d}(R_{A_i}, R_{A_j})$. The correlation $\mathrm{cor}(.,.)$ may be computed rank-based, if desired, or with standard linear correlation (Pearson correlation). Similarly to $\mathrm{RBC}_{\mathrm{all}}$, we can also compare each agent to the optimum agent A_{opt} (the agent with the largest reward) instead of performing all pairwise comparisons. We denote this as $\mathrm{RBC}_{\mathrm{opt}}$. A large RBC means that minor/substantial differences in reward coincide with slight/significant differences in behavior. Hence, a large RBC is a good indicator that the behavior space is easier to traverse for search algorithms and easier to learn for surrogate models.

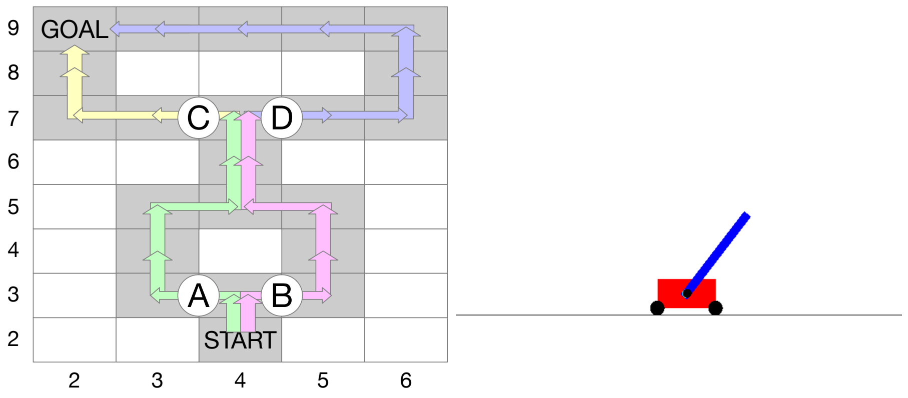
This property has a close connection to the fitness distance correlation (FDC) used in evolutionary computation to rate problem difficulty [Jones and Forrest, 1995]. There, differences in fitness are correlated with distances in the search space. $\mathrm{RBC}_{\mathrm{all}}$ considers all pairwise distances while $\mathrm{RBC}_{\mathrm{opt}}$ and FDC consider distances only between candidate solutions and the global optimum (or best-known solution [Kallel and Schoenauer, 1996]).

9.3 Experiments

9.3.1 Deterministic Maze

The deterministic maze was designed with *mazelab* [Zuo, 2018] as a comprehensible problem where correct actions are known and behavior is manually rateable. The environment, visualized in Figure 9.1(a), consists of a 10×7 matrix (shown in figures as 9x6, excluding external walls) with different encoding for accessible ways, walls, the agent, and goal. The target is to find the shortest path to the goal. The agent is allowed to take only deterministic actions for each observed agent position in each cardinal direction. Thus they can get stuck against a wall. Agents get a small negative reward for each movement, a larger negative reward if running against a wall or moving backward, and a positive reward for reaching the goal. The maximum step size of each agent is fixed to 30, whereas only 11 are needed to follow the shortest path. We manually designed the maze to feature DFS and UOS: The maze has four paths to the goal and 22 unique agent positions, but these are partly exclusive, and successful agents always have UOS. Moreover, the lower fork is a DFS, while the upper one is not. The intention was to construct a problem where agents with the same reward can have different behavior, cause of the forks, and different paths. Moreover, to analyze the effect of different exclusive paths and the UOS on the pairwise behavior comparison.

9.3.2 Continuous Inverted Pendulum



(a) *Deterministic Maze* **(b)** *Inverted Pendulum*

Figure 9.1: *Environments. For the maze environment, external walls are not displayed. Different ways: A and B are equal in reward, while D is slightly worse than C.*

The inverted pendulum is a time-dependent physics simulator with a continuous input space (Figure 9.1(b)). The target is to balance the pendulum on a car in the upright position for most time steps, starting at a random downwards position by moving the car. The environment is evaluated over 500 timesteps but discontinues if the base car moves out of designated limits. The action space was made deterministic for more comprehensible behavior comparisons. The pendulum environment has no exclusive paths, i.e., all states are observable, but agents will have an enormous number of UOS because of the real-valued input space. We also consider the environment to include multiple DFS, e.g., multiple correct behaviors are possible. The environment allows a large number of behaviors and different sized sets of observed states per agent.

9.3.3 Generating Reinforcement Learning Agents by Neuroevolution

The RL agents' policies are created and trained by utilizing neuroevolution to learn ANN policies in an evolutionary process. The underlying algorithm is the graph-based *cartesian genetic programming* `CGP` by A. Turner[1] [Khan et al., 2010; Turner and Miller, 2013]. For the maze problem, ANNs with 70 inputs and four outputs were evolved, where the softmax function computes the resulting action. The pendulum ANN has six inputs (5 + 1 bias) and a single output. For an output value > 0.5, the action is drive left; otherwise, drive right. The ANNs are evolved in terms of connection weights and structure, i.e., the number and placing of connections, nodes, and transfer functions. The maximum number of nodes and connections for each ANN was set to 100 (maze) and 1000 (pendulum). This leads to a vast amount of different ANN topologies. The inner workings of the ANNs are complex and very difficult to compare [Gaier et al., 2018; Stork et al., 2019b]. Thus, only the reward and the behavior of the agents using these ANNs are considered observable.
Table 9.1 summarizes the parameters and outcomes of the neuroevolution. The pendulum agents' rewards were aggregated over 30 different instances for reducing the impact of the random start positions; all states and actions from these instances are included in the agents' state sets. The agents of each environment were merged into one data set. Agents with equal state-input

[1]http://www.cgplibrary.co.uk - v2.4 - accessed: 2018-01-12

Table 9.1: *Parameters and results of the neuroevolution run for both environments*

	maze	pendulum		maze	pendulum
repeated runs	12	1	total agents	48e3	4020
evaluations per run	4020	4020 × 30	unique agents	43	3648
observed states	30	max 15e3	unique states	22	30e6

sets (i.e., those following precisely the same path) were filtered to acquire a feasibly sized data set. Due to the small number of input states for the maze environment, its amount of agents is significantly reduced. Conversely, the majority of trajectories in the pendulum experiment are unique. The cleaned-up data for each environment consist of all unique agents; the input states they observed, the corresponding actions, and their rewards. The agents were ranked, where equal performance leads to a shared rank. The maze problem has two best-ranked (rank 1) agents. For the following experiments, we arbitrarily chose one of these two as a reference agent (denoted as "best agent").

9.3.4 Experimental Setup for the Behavior Measures

Behavior Comparison: First, explorative data analysis is conducted to analyze the behaviors and visualize them in the environment. We analyze the behavior for individual input states, particularly for the maze problem, as we can manually identify wrong actions and understand their impact on the reward. Furthermore, we use a one-to-one comparison of agents with similar rewards to see the influence of UOS and DFS. For the pendulum problem, we analyze and reveal different behavior based on specific inputs and compare the influence of using different state sets as input for the behavior comparison. The denoted "best agent" for this problem is the best found.

State Importance: The maze environment has designed DFS and UOS, i.e., the forks with different importance and different exclusive paths to reach the goal. The goal of the importance analysis is to discover these states by comparing the behavior of all agents. We take a best-ranked agent as the reference for correct actions and calculate the percentage of different actions for each state by

one-to-many comparisons, weighted by the difference in rank for these agents, by $\mathrm{d}(B_A, B_{A'}) \times \mathrm{d}(rank_A, rank_{A'})/sum(rank_A, rank_{A'})$. Further, we calculate and visualize the action reward rank (Section 9.2.2) for each state.
Reward Behavior Correlation: The main challenge in computing the $\mathrm{RBC}_{\mathrm{all}}$ and $\mathrm{RBC}_{\mathrm{opt}}$ is selecting a suitable state set to compare the behavior. With the previous experiments' experience, we defined different options to select a suitable state set and analyze which of them leads to the best overall RBC:

– Input set A: Random states sampled from all known states of all agents.
– Input set B: All known states of an environment.
– Input set C: The observed states of the best agent.
– Input set D: The observed states of both compared agents.
– Input set E: The observed states of one compared agent.

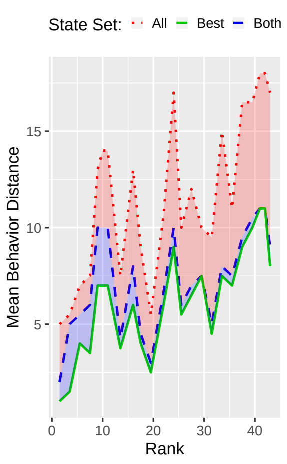
For the pendulum problem, we calculate the $\mathrm{RBC}_{\mathrm{all}}$ on an equidistant sampled (each 15th) subset of agents to significantly reduce the computation time.

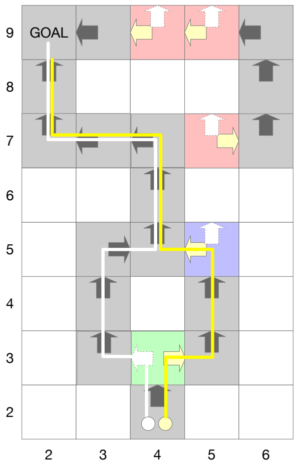
9.4 Results and Discussion

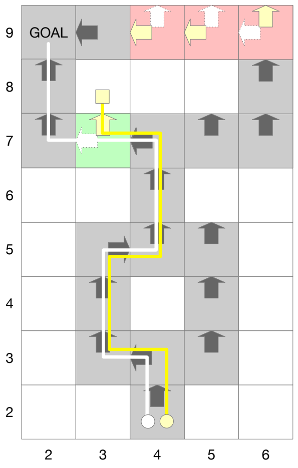
9.4.1 Behavior Comparison

The comparison of the best agent against all and selected inferior agents for the maze environment on different state sets (B, C, D) is illustrated in Figure 9.2. Interestingly, the best agent does not choose the best action in all states. It only chooses correctly for the states it observed by itself. The agent would run into walls if placed in certain positions (e.g., 5,5 or 4,9).

The behavior distance is amplified by different actions for states that were not observed by the compared agents, i.e., UOS lead to a larger distance, in particular visible in Figures 9.2(b) and 9.2(c), where red cells highlight the UOS. If the state input set of both agents are used instead of all, the influence of UOS is smaller, as at least one of the agent has observed these states (blue line/cells). However, it is still evident for the higher ranks.







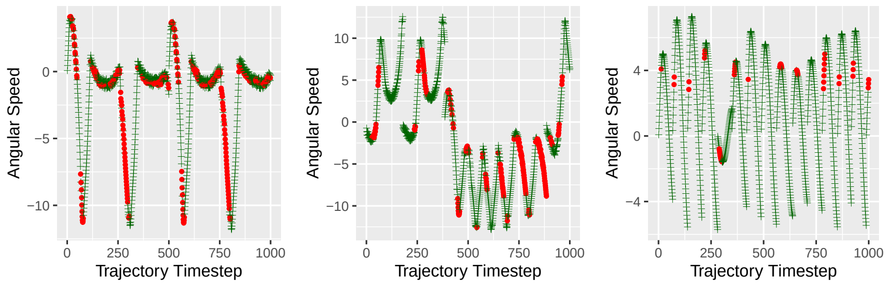
(a) *Behavior Distance, Aggregated Mean by Rank*

(b) *2x Rank 1 Comparison, d= All 5, Both 2, Best 1*

(c) *Best vs Rank 13, d= All 4, Both 1, Best 1*

Figure 9.2: *One-to-one behavior comparison of the best agent against all agents, computed with different state input sets (a) and two selected examples (b, c) Green: differences on input set C (BEST). Blue: differences on input set D (BOTH, includes C). Red: differences on input set B (ALL, includes C and D). Grey cells: agents act the same. a) Summary of behavior distance of best against all. Shaded areas illustrate the input set differences. b) Trajectories: best rank 1 (white) vs. another rank 1 (yellow) c) Trajectories: best (white) vs. Rank 13 (yellow)*

A remarkable observation is shown in Figure 9.2(c), for a comparison between the best agent and a medium-rank agent (rank 13). They have a behavior distance of only one if compared on their mutual state set, and four with UOS considered. Their behavior on their mutual state set is nearly identical, despite the significant difference in rank.



(a) *Trajectory from best, total difference is 33%* **(b)** *Trajectory rank 2000, total difference is 24%* **(c)** *Trajectory rank 3500, total difference is 4%*

Figure 9.3: *Behavior comparison for the pendulum. The behavior difference from best versus rank 1000 is shown (Green cross= same action, red dot= different action) for the* angular speed *value over the first 1000 states of state sets from best, rank 2000, and rank 3500. As visible, the behavior difference is influenced by the state sets. Particularly, the dissimilarity in (c) is smaller.*

The number of acquired states for the pendulum is enormous and unsuitable for complete comparisons as we visualized for the maze. However, we computed behavior differences of smaller state subsets and visualized them using a selected input, the *angular speed* of the pendulum, which is nearly zero if the pendulum is balanced in the upright position. Figure 9.3(a) shows the behavior of the best agent against the rank 1000 (of 3648) agent by calculating it on best, as well as on rank 2.000 (b) and rank 3500 (c) input sets. Figure 9.3(a) shows that for time-steps 250-300 and 750-800, the rank 1000 agent behaves consequently differently. These timesteps illustrate a situation of a falling pendulum shortly after it was balanced. While the best agent countersteers this movement, the rank 1000 agent accelerates it. Consequently, we were able to identify a situation where the lower-ranked agent fails to learn the correct actions. However, as the actions are based on all inputs and the angular speed is just one of them, finding these situations manually remains challenging. Figure 9.3(c) shows what happens if the behavior of the two agents is compared on the input set of a distant ranked agent. The state input set of the rank 3500 is considerably different: Each recorded trajectory is only some time steps long, presumably caused by the agent quickly driving the base car to the horizontal limit, which leads to termination. For such extreme situations, both compared agents (best vs. rank 1000) seem to behave similarly. Conversely, their difference in reward seems to be related to smaller differences in critical situations. We can summarize these observations to identify some properties of the behavior space:

I) Agents with the same reward/rank can have a considerable behavior distance, mainly if compared on state-input sets with UOS and DFS.

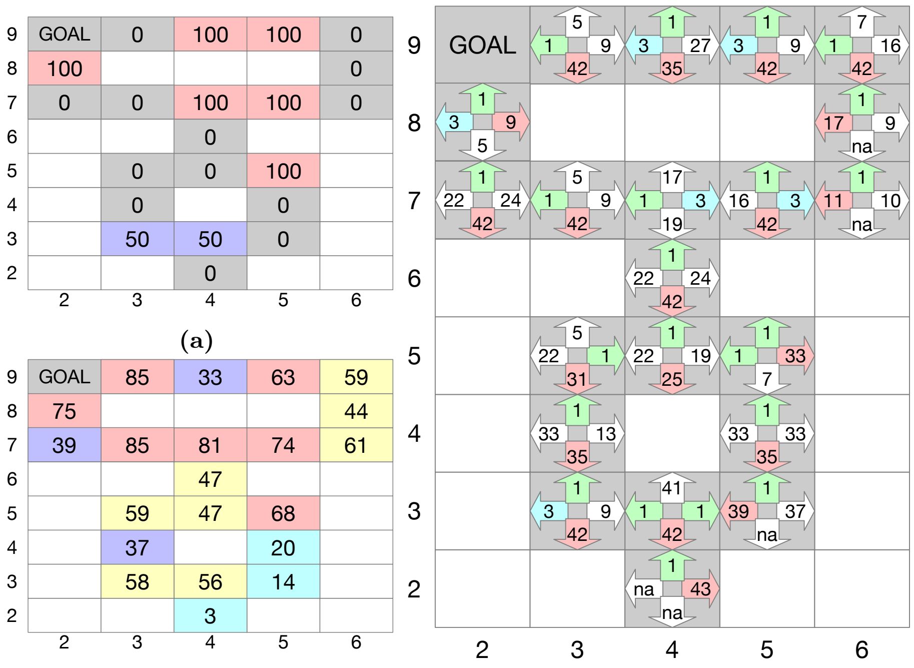
II) Small behavior differences (e.g., $d < 3$) can cause significant rank changes.

III) The input set has a significant impact on the behavioral distance comparison.

These observations reflect a central challenge of behavioral comparisons: We need to find essential states and a suitable state set for conducting behavior comparisons. We argue that comparing the behavior on input sets with UOS can help distinguish between agents of similar reward, but is presumably overestimating their behavior distance on task level and further influenced by significant variances due to random actions in UOS. Moreover, comparing agents on state sets of other agents, even without considering the influence of UOS, might not reveal proper behavior distances, as these states represent situations not suitable for telling apart good behavior.

9.4.2 State Importance

For the state importance, we illustrate the percentage of agents with behavior differing from the best agent for each state, weighted by their differences in rank. In case of the maze, Figure 9.4(a) shows this statistic only for agents reaching the goal, while Figure 9.4(b) concerns all agents. Here, highly valued states are considered to be more important, as most agents behave dissimilarly to the presumed “correct” action. For the comparison between the best agents, many states show no importance, i.e., similar behavior in this set.



(a)

	2	3	4	5	6
9	GOAL	0	100	100	0
8	100				0
7	0	0	100	100	0
6			0		
5		0	0	100	
4		0		0	
3		50	50	0	
2			0		

(b)

	2	3	4	5	6
9	GOAL	85	33	63	59
8	75				44
7	39	85	81	74	61
6			47		
5		59	47	68	
4		37		20	
3		58	56	14	
2			3		

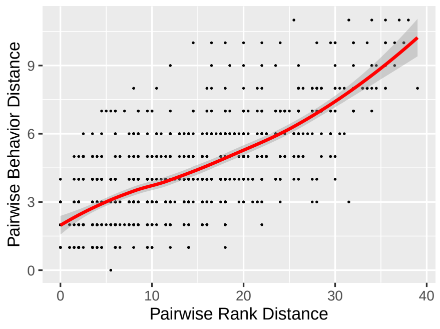
(c)

Figure 9.4: *(a) State importance calculated using either the best agents or (b) all agents. Higher values depict higher importance, colored by value quarters for easier comparison. (c) Action reward rank. Shows the best rank choosing each action for each state. Green=rank 1, blue=rank 3, red= worst action rank. The two rank 1 agents choose different actions in (4,3) and (5,5). (4,3) is DFS, and (5,5) an UOS for the best agent.*

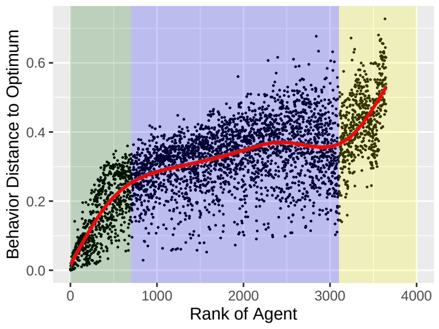
The maze was designed such that the importance of the DFS fork in (4,3), should be less than the no-DFS fork in (4,7). Our importance measure represents this, as (4,7) has a twice as high value in Figure 9.4(a) and (b). However, the importance measure also provides other states with a high importance value, particularly visible in the maze's upper part. This can be explained by the type of behavior comparison (all states) and the influence of UOS for each agent. Agents can have 'wrong' behavior for these states, even if they can solve the environment. This is observable in Figure 9.4(c), where for each state, the best agent choosing a specific action is shown. While for the state (4,3) and also for (4,7), we see a correct identification of different ways, (5,5), (4,9), and (5,9) give the wrong idea of correct actions, as the supposedly best-outlined action is surprisingly to run against a wall. This effect of UOS is amplified if all agents are considered. For example, the lowest-ranked agent runs directly against a wall. However, we compute and compare its behavior (intensified by its low rank) in all states. We assume that if we compare a broad set of agents, the UOS, with their presumably random actions, does not strongly affect the importance. Thus, the shown importance is presumably higher in the states of the upper part of the maze, as only a minority of agents reach this part of the maze. For the rest, we are comparing their behavior on UOS. Thus, our importance measures could also help identify states of an environment rarely reached by any agent in a set.

9.4.3 Reward Behavior Correlation

For the RBC analysis, the previous results have shown that it is essential to choose a suitable state set for each pairwise comparison. The results are displayed in Figure 9.5 and Table 9.2. Figures 9.5(a) and 9.5(b) shows the resulting $\mathrm{RBC}_{\mathrm{all}}$ and $\mathrm{RBC}_{\mathrm{opt}}$ values, respectively. For both, the overall correlation is notably positive. In 9.5(b) it differs between good agents (rank 1-800), medium agents (800-3100), and poor agents (3100-3600). The other input sets (A, B, C, and E) lead to an inferior $\mathrm{RBC}_{\mathrm{opt}}$ for both problems.



(a) *Pairwise RBC_{all} for maze, Correlation=0.72, Input set E*



(b) *Pairwise RBC_{opt} for pendulum, Correlation=0.70, Input set D*

Figure 9.5: *(a) RBC_{all} plot for the maze environment (b) RBC_{opt} plot for the pendulum problem. Computed on input set E and D, respectively. A decent correlation is visible.*

Moreover, set D and E lead to the highest RBC_{all}, with a very significant difference for the maze problem. We assume that the best correlation would be achieved if agents are compared in their mutual behavior space. The presumed cause for the higher RBC_{all} is the reduction of the influence of unobserved states in the comparison. In particular, the maze environment agents have less UOS where they likely act randomly if sets D or E are considered. Including UOS in a behavior comparison does not lead to a more detailed behavior distance, but one with higher overall variance, leading to an inferior RBC. This is visible for the pendulum, which for all sets, has a large amount of UOS due to the continuous state space, which leads to a smaller difference in the variants to compute the $\mathrm{RBC}_{\mathrm{all}}$. The overall positive $\mathrm{RBC}_{\mathrm{all}}$ outlines the potential of agent comparisons in the behavior space to improve the search for good ranking agents.

Table 9.2: *RBC_{all} of all agents for different input sets*

environment	A) random	B) all	C) reference	D) both	E) one
maze	0.27	0.29	0.28	0.62	**0.72**
pendulum	0.36	na	0.34	**0.45**	0.36

9.5 Conclusion and Outlook

In this chapter, we investigated the properties of the behavior space of RL agents and how this space can help to compare agents in learning sets to gain valuable insights. Regarding our research questions, we can conclude for Q-1, that even small changes in the behavior can have considerable effects on the reward. At the same time, agents achieving the same reward can show quite different behavior. We believe that focusing only on the reward of an agent might not be the optimal choice. Instead, the agents' behavior can give valuable insights into how agents achieve that reward. This can reveal agents with surprising behavior or help to improve the learning process. For instance, reward functions can be designed to enforce or suppress specific behavior.

The analysis of Q-2 has shown that accessing the variable importance is challenging and highly dependent on the underlying set of agents and the environment. These challenges are mainly caused by comparing an agent on states, which were not observed by it, or are even not observable by this agent due to environmental restrictions, e.g., mutually exclusive paths. For these cases, an agent's behavior can be random, even for the ones with the best reward. A comparison of behavior in these states might deliver misleading results. Only if multiple agents observed states could we access their real importance. This finding is further stressed when considering Q-3. The RBC is highest if we consider pairwise behavior comparisons on those states that both compared agents have observed. The reasonable positive RBC demonstrates that the behavior space is promising and searching in that space may be beneficial.

For future work, we aim to take a close look at how the understanding of behavioral spaces can be exploited, e.g., by new reward measures, direct search in the behavior-space, and specialized search operators:

Reward measures: Ideally, reward measures help to steer the search into desirable areas of the search space. Understanding which states are critical to receiving a good match between behavior and reward may help design better reward measures. The importance of developing proper reward measures for RL is stressed in a review by Doncieux and Mouret [2014].

Search in behavior space: The usage of agents' behavior distance as an additional search criterion seems very attractive. It can be used to preserve diversity in evolutionary search procedures [Doncieux and Mouret, 2010]. Further, the search for a specific behavior may be of interest, independent, or in addition to reward-driven search, e.g., by modeling the reward to behavior space with surrogate models. An example application would be *inverse reinforcement learning* [Ng et al., 2000]. The search in behavior space allows using entirely different agent topologies or even comparing agents trained by different algorithms.

Search operators: Finally, a good understanding of the latent, behavioral space may help to define better search operators. For instance, search operators could be designed to search directly in the behavior space rather than the policy or topology space.

10
A Framework for Behavioral Optimization in Reinforcement Learning

Chapter 10 was published as:

Stork J, Zaefferer M, Eisler N, Tichelmann P, Bartz-Beielstein T, & Eiben AE (2021). Behavior-based neuroevolutionary training in reinforcement learning. *(to appear) In 2021 Genetic and Evolutionary Computation Conference Companion*. ACM, New York. DOI: 10.1145/3449726.3463171

In addition to their undisputed success in solving classical optimization problems, neuroevolutionary and population-based algorithms have become an alternative to standard reinforcement learning methods. However, evolutionary methods often lack the sampling efficiency of standard value-based methods that leverage gathered state and value experience. If reinforcement learning for real-world problems with significant resource costs is considered, sampling efficiency is essential. The enhancement of evolutionary algorithms with experience exploiting methods is thus desired and promises valuable insights. This work presents a hybrid algorithm that combines topology-changing neuroevolutionary optimization with value-based reinforcement learning. We illustrate how policies' behavior can be used to create distance and loss functions, which benefit from stored experiences and calculated state values. They allow us to model behavior and perform a directed search in the behavior space by gradient-free evolutionary algorithms and surrogate-based optimization. For this purpose, we consolidate different methods to generate and optimize agent policies, creating a diverse population. We exemplify the performance of our algorithm on standard benchmarks and a purpose-built real-world problem. Our results indicate that combining methods can enhance the sampling efficiency and learning speed for evolutionary approaches.

10.1 Introduction

In the last years, neuroevolution and population-based methods have shown to be a valuable and scalable alternative to classic approaches in reinforcement learning, as they have shown promising performance on several problems [Salimans et al., 2017; Jaderberg et al., 2017; Jung et al., 2020; Khadka and Tumer, 2018]. In particular, they are computationally efficient if the problem itself is fast to evaluate and a large number of parallel evaluations are possible. Evolutionary algorithms applied to RL often rely on an episode-to-episode fitness evaluation, considering an RL episode's final cumulative reward for selection and updating. Further, they do not take advantage of the details of individual behavioral interactions and do not leverage from the gathered experiences of state information. Thus, they often require a significant amount of fitness evaluations to evolve well-performing agents. In particular, if artificial neural networks with changing topologies are considered, which employ a large search space [Stork et al., 2019b]. The resulting low sampling efficiency is a challenge in real-world problems due to their high costs, as each action can have a considerable duration.

This paper's primary goal is to combine a topology-changing neuroevolutionary algorithm with a behavior-based search to improve sampling efficiency.

Standard value-based and actor-critic policy gradient methods exhaust behavioral information for their updates and are remarkably successful in many different domains [Sutton and Barto, 2018]. The combination of evolutionary methods with value-based methods has recently become an active area of research. Recent methods include hybrid approaches, which collect experiences by an evolutionary part, then apply value-based learning, such as policy-gradients, to selected population members [Khadka and Tumer, 2018; Khadka et al., 2019]. The use of experience-based mutation operators, e.g., which perform gradient updates, is also promising to improve evolutionary methods [Franke et al., 2019].

Another approach for improving the sampling efficiency is to replace the actual problem function with a surrogate, as employed in surrogate model-based optimization. The implementation of surrogates in neuroevolutionary algorithms for reinforcement learning has been of increased interest [Gaier et al., 2018; Stork et al., 2019b]. The difficulty in modeling agents is the definition of an adequate distance between their policies. In the case of changing topologies, the definition of genotypic differences (i.e., differences of the encoding) is algorithm-dependent and not necessarily helpful [Gaier et al., 2018; Stork et al., 2019a]. One method is to utilize behavioral distances, in this context defined as the actions of an agent taken in pre-defined states. Behavioral embeddings are also used to maintain diversity in a population, which is essential to avoid overfitting certain behaviors [Parker-Holder et al., 2020].

In this paper, we seize these ideas and implement a hybrid framework for *behavior-based neuroevolutionary training* (BNET), which combines the evolution of a population of agents with topology-changing neuroevolution, employed by *cartesian genetic programming* [Miller and Thomson, 2000; Turner and Miller, 2015]. Our contributions are as follows:

(1) We introduce the new algorithm where the population's agents are optimized in parallel by fitness-based search, behavior-based search, and surrogate-based search. As a critical challenge in RL is exploration, our implementation is focused on maintaining a diverse set of agents, each contributing to the fitness search and sharing experiences. Different approaches to leverage from a shared experience set exist [Schmitt et al., 2020]; we employ a selection method to create an archive of experiences from high-performing yet diverse policies.

(2) We define a set of new advantage-weighted behavior loss functions to conduct the behavior-based search, leveraging on principles from standard actor-critic policy algorithms and behavior-based distances [Sutton and Barto, 2018]. In combination, they are part of a gradient-free learning process for the agent policies. For SMBO, we employ the *surrogate model-based optimization for neuroevolution* algorithm, first introduced in [Stork et al., 2019b].

(3) We define a method for robust candidate selection to increase learning stability. Our fitness-based search focuses on keeping a stable elitist solution, referred to as champion, which is of central importance considering real-world environments. As these environments frequently provide stochastic rewards, e.g., caused by random starting conditions, we employ a robust selection method, which compares mean estimates of their actual performance and reduces the probability of choosing an inferior candidate.

(4) We implement a prototype of the framework and test it on common synthetic benchmark problems against standard value-based RL methods to prove our concept. We also analyze the performance of each search method in BNET. Finally, we introduce a new adaptable real-world problem featuring a robot-controlled maze environment as a benchmark for RL algorithms.

The chapter is organized as follows: In Section 10.2, we introduce the basics for the algorithm and describe each of the BNET components in Section 10.3. In Section 10.4, we describe our experiments, introduce the real-world benchmarks, and further discuss the experiments' results. We conclude the chapter in Section 10.5 and provide an outlook.

10.2 Methods

10.2.1 Reinforcement Learning and Value-based Methods

In RL, we train agents' policies $\pi_n, n = 1, ..., N$ to solve instances of an environment: for each environment step t the agent observes an environment state s_t and the policy decides, which action a_t is conducted. The agent receives a reward $r(s_t, a_t, s_{t+1})$ for each observed state during its learning episode. This leads to a *trajectory* with T steps. The cumulative reward $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$ at the end of an episode, discounted by factor γ, is typically utilized as a target function of the policy optimization process. We refer to R_t of a full episode as *fitness* of a policy.

Advantage-based actor-critic methods [Sutton and Barto, 2018] have a policy actor and estimate the *advantage* of an action a_t in state s_t by a critic. The critic $V_\varphi(s_t)$, with φ being its parameters, approximates the estimated *value* of a state s_t, given by the value function $V(s) = \mathbb{E}_\pi\{R_t | s_t = s\}$. For a full-episode learner, i.e., considering the complete per-state reward information R_t at the end of an RL episode, a typical approach is to compute the advantage by a Monte-Carlo (MC) estimate,

$$A_\varphi(s_t, a_t) = R_t(s_t, a_t) - V_\varphi(s_t), \tag{10.1}$$

where $V_\varphi(s_t)$ is approximated by the critic. Further, the actor is updated with help of a gradient function:

$$\nabla_\theta J(\theta) = \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) \; A_\varphi(s_t, a_t) \tag{10.2}$$

Here θ are the ANN parameters, and π_θ is the policy associated with these parameters. We refer to the *experience* of an agent, considering the (s_t, a_t, r_{t+1}) triplets an agent observes during (multiple) interaction episodes in a RL environment.

10.2.2 Neuroevolution by Cartesian Genetic Programming

CGP [Miller and Thomson, 2000] is a genetic programming method relying on grid-based encodings to realize graph representations. If applied to ANNs [Turner and Miller, 2015], it allows to create topologies with different transfer functions and free node-to-node connections, i.e., the generated topologies do not follow the typical layered structure. CGP, in its basic version, does not allow the direct use of back-propagation or gradients for ANN optimization, mainly due to the topology-changing neuroevolution process. Certain CGP implementations allow the utilization of gradient information [Izzo et al., 2017]. However, these learn topologies and weights sequentially, not simultaneously. The genotype-behavior mapping is complicated as distances on the genotype level are barely related to their distance in behavior space [Stork et al., 2019a]. We optimize the CGP-ANNs with a gradient-free $(\mu + \lambda)$-EA with rank-based fitness selection [Eiben and Smith, 2015b] for the optimization of NE candidates, utilizing behavior-based loss functions, as described in Section 10.2.3.

10.2.3 Optimization in the Behavior Space

Each policy computes the probability $p(a_t|s_t)$ of taking an action a_t for an input state s_t. Formally, we denote *behavior* as the set of probabilities corresponding to K states $\mathbf{S} = \{s_1, ..., s_K\}$. Further, $\mathcal{B}$ is denoted as the *behavior space*, i.e., the set of all possible behaviors, with $\pi_n \in \mathcal{B}$ and $n = 1, ..., N$. In the behavior space, two agents can be directly compared on the state set $\mathbf{S}$ by calculating their mean *behavior distance* [Stork et al., 2019a,b], denoted by

$$\mathrm{d}_b(\pi, \pi', \mathbf{S}) = \frac{1}{T} \sum_{t=1}^{T} |\pi(\mathbf{s}_t) - \pi'(\mathbf{s}_t)| \tag{10.3}$$

As the Manhattan distance applies to both ordinal integers and continuous values, it is preferred over the Euclidean norm. For the case of experience-based policy optimization, we defined the *advantage-weighted behavior distance*:

$$\mathrm{d}_{wb}(\pi, \pi', \mathbf{S}, \mathbf{A}) = \left(\frac{\frac{1}{T^*} \sum_{t=1}^{T^*} |\pi(s_t) - \pi'(s_t)| \times A_\varphi(s_t, a_t)}{\frac{1}{T^*} \sum_{t=1}^{T^*} |A_\varphi(s_t, a_t)|} \right) \tag{10.4}$$

Here, $\mathbf{S}$ are sampled states from environment interactions with a trajectory of T^* time-steps, while $\mathbf{A}$ are the respective actions taken during this trajectory. Further, A_φ is the precomputed approximated advantage for each of the stored actions, see Section 10.2.1. As a second method, we utilize the *advantage-weighted cross-entropy.* The cross-entropy is defined by:

$$H(\pi, \pi', s_t) = -\sum_{i=1}^{I} \left[\pi_i(s_t) \ln\left(\pi'_i(s_t)\right)\right] \tag{10.5}$$

$\pi_i(s_t)$ is the i-th element of the probability output of the policy π given the input state s_t. The cross-entropy distance (d_c) is further weighted by the advantage estimation $A^+_\varphi(a_t, s_t)$ and computed over a set of stored states $\mathbf{S}$ and actions $\mathbf{A}$:

$$d_c(\pi, \pi', \mathbf{S}, \mathbf{A}) = \frac{1}{T^*} \sum_{t=1}^{T^*} \left[H(\pi, \pi', s_t) \times A^+_\varphi(s_t, a_t)\right] \tag{10.6}$$

Instead of the complete estimated advantage, we only consider the set of positive advantages:

$$A^+_\varphi(s_t, a_t) = \begin{cases} A_\varphi(a_t, s_t), & \text{if } A_\varphi(s_t, a_t) \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{10.7}$$

The positive advantages drive the search towards estimated beneficial behavior, while negative advantages will not affect the weighted cross-entropy distance d_c. We define our loss function for NE-EA during the behavior-based optimization as the sum of distances (d_{wb} or d_c) between the target policy π and several stored reference behaviors π_m with $m = 1, ..., M$.

Furthermore, their sampled trajectories with states $\mathbf{S}_m$ and actions $\mathbf{A}_m$ are:

$$L(\pi) = \sum_{m=1}^{M} \big[\mathrm{d}(\pi, \pi_m, \mathbf{S}_m, \mathbf{A}_m)\big]. \tag{10.8}$$

For the loss functions, the probability distribution of the stored behaviors is priorly adapted to maximize the performed action's probability. The optimization in the behavior space $\mathcal{B}$ is similar to imitation learning, i.e., fitting a network to replicate a stored behavior. However, instead of replicating a single reference behavior, we optimize the target policy to minimize the distance to an advantage-weighted set of multiple reference behaviors from different policies. The EA for the optimization is outlined in Algorithm 10.2.1.

Algorithm 10.2.1: Neuroevolution EA

```
1  INPUT: Memory of $M$ stored reference policies $\pi^*_m$, states and actions
       $\mathbf{S}^*_m, \mathbf{A}^*_m$;
2  optional: pre-defined candidates $\pi$
3  preset: mutation rate, NE candidate parameters
4  begin
5      initialize new polices as candidates
6      evaluate initial candidates with loss function $L(\pi)$
7      select $\mu$ parents from initial candidates
8      while not termination-condition do
9          mutate parents to get $\lambda$ offspring
10         evaluate offspring with loss function $L(\pi)$
11         select $\mu$ next iteration's parents with minimum loss from parents and
             offspring
12         optional: update mutation rate
13     end
14 end
15 OUTPUT: best found policies
```

10.2.4 Optimization by Behavior Surrogates

The definition of the behavior distance between policies also allows us to create approximation models, so-called surrogates, which predict a policy's fitness during optimization. The surrogates allow searching in the behavior space without any additional environment interactions, which are substituted by the surrogate. SMBO is frequently applied for costly processes with high resource-demand for each function evaluation [Forrester et al., 2008]. Our SMBO is based on the SMB-NE algorithm [Stork et al., 2019b], which utilizes a Kriging [Forrester et al., 2008] regression model. The model measures the similarity of samples by a kernel, utilizing distance and correlation matrices of observations. If applied to real-valued samples, the exponential kernel $\mathrm{k}(x, x') = \exp(-\theta ||x - x'||_2)$ is a typical choice. The essential kernel parameter θ influences how fast the correlation decays to zero if the Euclidean distance between two samples $||x - x'||_2$ increases. In our work, we follow the idea of kernel-based models for combinatorial search spaces [Moraglio and Kattan, 2011a; Zaefferer et al., 2014b]. We replace the Euclidean distance $||x - x'||_2$ by the *behavior distance*, resulting into the kernel:

$$\mathrm{k}(\pi, \pi', \mathbf{S}) = \exp(-\theta\, \mathrm{d}(\pi(\mathbf{S}), \pi'(\mathbf{S}))) \tag{10.9}$$

Here, one challenge is the appropriate definition of the state set $\mathbf{S}$, as it has a considerable influence on the distance. We rely on a selection approach presented in [Stork et al., 2020], where both policies' stored states are combined for each pairwise distance calculation. If a new target network's fitness without stored states needs to be predicted, the reference policies' stored states are applied as reference input. The Kriging model parameters are fitted by maximum likelihood estimation and optimized with DIRECT-L [Gablonsky and Kelley, 2001], further utilizing the *nugget effect* [Van Beers and Kleijnen, 2003]. Kriging combines relatively accurate mean predictions with the ability to provide uncertainty estimates of each prediction. The combination of mean and uncertainty are used to compute infill functions, which predict the desirability of a solution. A frequently applied infill-criterion is *expected improvement* [Mockus, 1974; Jones et al., 1998], which integrates the uncertainty estimate to explore new solutions, which may be farther away from observed solutions. In [Rehbach et al., 2020], the benefits and downsides of using EI for different optimization problems are discussed. For high-dimensional cases, it is recommended to use the predicted mean instead of EI since the increase in dimensionality leads to an inherent increase in uncertainty (see also: [Wessing and Preuss, 2017]). In the case of neuroevolution, the behavior space is high-dimensional. Thus, we use the predicted mean of the Kriging surrogate as an infill function. The R package `CEGO` [Zaefferer, 2017; Stork et al., 2019b; Zaefferer et al., 2014b] is used to train the Kriging surrogates, while the NE-EA is utilized to optimize the target network.

10.3 Behavior-based Neuroevolutionary Training

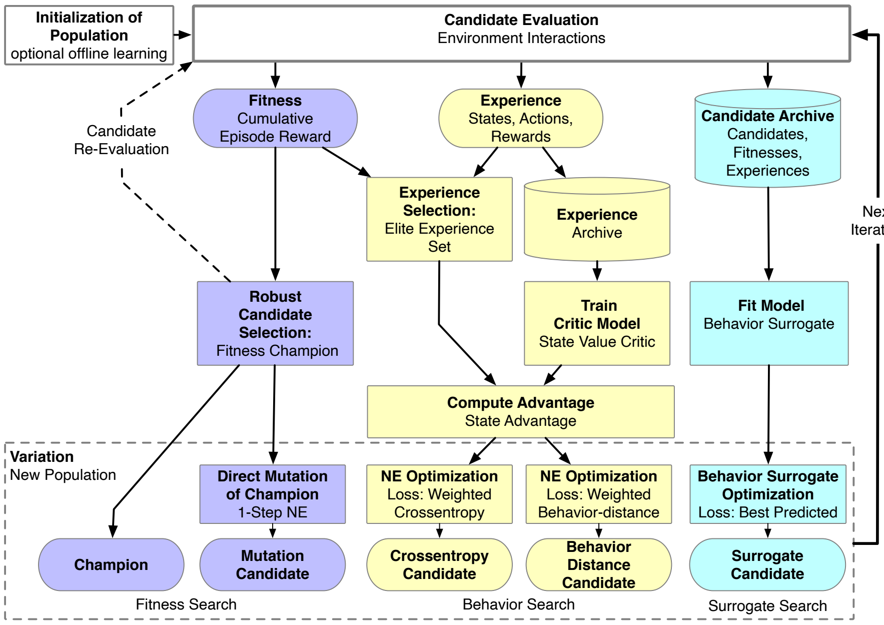


Figure 10.1: *BNET algorithm cycle. Quadratic boxes are methods or algorithms, and rounded edges illustrate observed data. The candidate generation methods are based on extracted data from the environment interactions during the fitness evaluation: The fitness-based search selects a champion with the best overall fitness (violet). The behavioral search utilizes a selected set of experiences, an advantage-critic model, and the defined behavior-distance and cross-entropy loss-functions to optimize networks and use them as new candidates (yellow). Further, a surrogate model is fitted to all candidates' archives and generates a candidate with SMB-NE (light-blue).*

In this section, we introduce our new algorithm for population-based neuroevolution for the training of RL policies. The population consists of CGP-ANNs, which are utilized as RL policies π. We refer to each network instance as a candidate. BNET is similar to a population-based evolutionary algorithm with elitism. Nevertheless, new candidates are not merely created employing a variation of previous candidates. Instead, new candidates are also generated by behavior learning and by exploiting behavior-based surrogates. The algorithm outlined in Figure 10.1 is essentially divided into three parts, which employ different ways to search for new candidates: fitness-based, behavior-based, and surrogate-based. In summary, the fitness-based search selects a champion with the best overall fitness by robust selection and creates direct mutations of this champion. The behavior-based search utilizes a selected set of experiences, a value-based critic model, and the loss function from Eq. 10.8 to generate new candidates. Further, a surrogate model is fitted to an archive of all candidates and searched for new candidates.

10.3.1 Initialization and Evaluation

The first population is either initialized by NE or by optimizing a population to fit previously evaluated policies' behavior. We refer to the second method as *offline* initialization, as it does not require any new (online) environment interactions. If we initialize NE policies offline, the behavior search is applied, where a stored experience set is required as a reference.

The candidates are then evaluated in the RL environment with two goals: First, evaluate the current population's policies to acquire their fitness; Second, to gather new environment experiences from these policies. The policies can be evaluated either fully deterministic, stochastic, or stochastic with exploration. The deterministic policy evaluation chooses the action with the highest probability, whereas the stochastic evaluation samples actions based on the probabilities. Deterministic policies have equal behavior if evaluated repeatedly, but the achieved fitness still may differ, e.g., if the environment itself is stochastic. Deterministic policies are most suitable for exploitation or testing.

As stochastic policies depend on probabilities, they allow for a certain level of exploration. Additional exploration can be enforced by taking random actions or adding a random factor to any policy behavior.

10.3.2 Fitness Search and Robust Selection

The fitness search, visualized as the leftmost, violet path in Figure 10.1, focuses on selecting a *fitness champion*, i.e., the candidate of the population with the best overall fitness. Moreover, a mutation candidate is generated by applying a small direct mutation to the champion's policy network and used as a second new candidate, the *mutated champion*. However, RL environments, such as control tasks, typically have different starting conditions or even stochastic state transitions, i.e., an action in a specific state may transition to different states in the next time step. Thus, the measured fitness will be noisy, and even a deterministic policy produces different results when evaluated repeatedly.

Therefore, we employ a robust fitness selection, including the re-evaluation of candidates and comparing their fitness distributions. The robust selection shall ensure that the champion is a reasonable estimate of the best policy discovered so far, even in the presence of noisy fitness. In the first iteration, the fitness champion, denoted as π^*, is selected as the candidate with the highest sampled fitness so far and re-evaluated in the second iteration. Its fitness is then set to the mean of all evaluations. Starting with the second and all future iterations, we use the concept of challengers: A challenger is a candidate π that has a one-time evaluated fitness better than the mean champion fitness, i.e., if $f(\pi) > \frac{1}{n^*}\sum_1^{n^*} f(\pi^*)$, where n^* is the number of samples of champions performance $f(\pi^*)$. If a new population contains at least one challenger, this challenger is re-evaluated to get a more robust fitness estimate and the mean fitness values are utilized as a measure for selection. A challenger is accepted as champion, if $\frac{1}{n}\sum_1^{n} f(\pi) > \frac{1}{n^*}\sum_1^{n^*} f(\pi^*)$, where n is a parameter of the selection, either set to match n^* or a desired number of repeats r, by $n = \max(n^*, r)$. The value of r should be chosen according to the task and estimated noise of the environment. If a champion was re-evaluated less than r, it is also re-evaluated.

In the case of multiple challengers, they are ordered by their fitness and sequentially evaluated against the champion. If a new champion is selected during the consecutive comparisons, the next challenger is only considered if it is still superior in fitness. The selected champion is then re-evaluated in the next iteration. The constant re-evaluation of the champion, if not changed, leads to a stable estimate of the actual fitness value.

10.3.3 Behavior Search

The behavioral optimization of the neuroevolutionary policies builds upon principles from standard RL algorithms, such as *policy-gradients* and *actor critic*. The challenge in neuroevolution is the absence of any gradient information between NE candidates if a simultaneous topology optimization is performed. A neuroevolutionary behavior optimization thus requires metrics that allow the comparison of policies further to establish a search direction in the behavior space. We employ the behavior optimization with the *advantage-weighted behavior distance* from Eq. 10.4 or the *advantage-weighted cross-entropy* Eq. 10.6 and implement it in our loss function Eq. 10.8. This loss application requires a defined set of experiences, an advantage function, and an optimization algorithm. As a reference set, we collect and store a fixed-sized set of *elite experiences*. The set consists of experiences and the performance of single evaluations from different candidates or repeated evaluations of the same candidate. In the first iterations, the set grows until the maximum size of the set is reached. After the robust selection, the experience is replaced with those of higher fitness episodes in each consecutive iteration. However, the number of replacements in each iteration is limited. The limitation prevents that the candidate set gets dominated by the experiences of single candidates. A diverse set is thought to help avoid overfitting and increase the chance to learn better behavior. A diverse experience set is the first difference to classic imitation learning, where a single policy's behavior is adapted. The second difference is given by the advantage weighting of each action in the distances by Eq. 10.4 and 10.6. The required advantage function is predicted by a critic model, which is learned to the complete set of all discovered experiences. Also, the advantage-weighted cross-entropy considers only the actions with positive advantage, i.e., the policy is trained to replicate this behavior. Both loss metrics are employed in the NE-EA to generate new candidates.

10.3.4 Surrogate Search

Corresponding to Section 10.2.4, the surrogate search is based on a Kriging model fitted to an archive of tested candidates with their connected mean fitness and experiences. The distance kernel in Eq. 10.9 is applied for modeling the relations between policy behaviors and their fitnesses, where the state set $\mathbf{S}$ of each comparison consists of the stored experience archive of the associated candidate pair. The fitted surrogate is then employed to predict the fitness of new candidates and utilized as loss-function $L(\pi) = -\hat{f}(\pi)$ during a NE-EA optimization process. The fitness values are adapted to ensure a minimization problem (i.e., negated in the typical reward maximization case).

10.4 Experiments

The BNET framework is flexible, as the algorithm modules for generating candidates can be combined in several ways. For example, it can also be employed as a pure direct neuroevolution approach by refraining from using the behavior-based or surrogate-based search, or in contrast, as a pure behavior search algorithm. Thus, our experiments are two-fold: first, we tested different versions of the algorithm against a set of open-AI baselines algorithms on the problems CartPole-V0 and MountainCar-V0. For this experiment, the focus was to estimate overall performance, how beneficial the different proposed modules are, and how they affect the search quality. In a second experiment, we tested our algorithm against the same baselines on a designed real-world problem.

As comparison baselines, we employ *advantage actor critic* (A2C) [Mnih et al., 2016] and *proximal policy optimization* (PPO) [Schulman et al., 2017] from the *stable baselines* package [Hill et al., 2018]. Both do not require full episodes to learn (i.e., the algorithm is trained after x time steps, not necessarily full episodes), which might give it performance advantages over BNET. Our performance measure, particularly concerning the real-world environment, is the number of required time steps until a (stable) solution is found. For all experiments, we implement a prototype version of the BNET framework using R 3.6.3, an R-interface to the CGP-ANN Library by Turner [Stork et al., 2019b] and *reticulate 1.14*, *Keras 2.3*, OpenAI *gym 0.18.0*, and *tensorflow 1.15.4* [Brockman et al., 2016; Abadi et al., 2015; Chollet et al., 2015]. All simulated experiments were conducted on an HPC-Cluster. More than 50,000h of total computation time was spent during development and experimentation. The BNET prototype was not systematically tuned for optimal parameter settings due to the high computational effort. The used parameter setup is based on preliminary tests and CGP-ANN or SMB-NE related publications [Stork et al., 2019b, 2020]. The baseline algorithms parameter were also improved (from the stable baseline default settings) based on preliminary results for each problem at hand, e.g., the reward discount parameter gamma and the learning rate. One aspect of the BNET setup was kept equal for all tested problems: The CGP-ANNs have a maximum of 200 active nodes with arity ten and a function set including tanh, sigmoid, gaussian, softmax, step, and rectified linear units. The direct, behavior, and surrogate search's mutation rates were 1%, 5% and 5%, respectively. The NE-EA uses a (20+2) population with 1000 iterations for the behavior search and (8+2), 500 for the surrogate. The critic network is a fully-connected feed-forward ANN with two layers and 128/64 nodes, trained for 1000 steps in each iteration. The prototype code and all experimental results are available in an online repository: `https://github.com/jstork/BNET-GECCO21`.

10.4.1 Open AI Gym Benchmark Setup

For comparing the performance of BNET against different baselines, we chose two basic Open AI Gym standard benchmarks. They were explicitly chosen because of their different characteristics. Moreover, both should be solvable in less than 20,000 steps. They present a baseline for a real-world scenario, where only a few steps are possible due to the high resource cost.

Cartpole-v1 is a standard benchmark, where the goal is to balance a pole placed on a cart. The environment has four observable variables (x position, x velocity, pole angle, angular velocity) and two discrete actions (drive left or right). The target is to keep the pole upright in a slight angle range for an average of 195 steps over 100 consecutive episodes. If the pole fails to balance (i.e., the angle reaches a threshold) or the cart drives out of a certain x-range, an episode is stopped. For each step, the agent receives a reward of +1.

In **MountainCar-v0**, a car is located between two mountains and has to drive to the top of one of them. As the direct acceleration is not high enough, it has to build up momentum by alternatively driving up and down the mountains. The environment has two observable variables (x-position, velocity) and three actions (accelerate left/right, do nothing). The target is to drive to a goal position on the proper mountain in less than average 110 timesteps. Each step is rewarded by -1, and the environment is stopped if either the step limit (200) or the goal is reached. The environment requires considerable exploration to find a solution to reach the goal point. If the exploration is unsuccessful, it remains with a -200 reward in each episode and gains no valuable experience. This flat reward landscape renders the environment challenging to solve in a small number of steps. For both environments, the starting state of the pole or car is randomly set in a small range, leading to different initial scenarios for each episode. Therefore, each setup was repeated at least 20 times with random initial seeds. The run was stopped if a found policy reached the required average target, which was evaluated in an extra function to save unnecessary computation time. The fitness or experiences of these stopping criteria evaluations were not utilized in any other form (e.g., for the algorithm itself).

The first benchmarks include setup variants of BNET, where selected candidates were generated in each iteration. The elitist was always kept and repeated (for the robust evaluation). The setups are: *Base* (all proposed modules are active), *BDist* (behavior distance), *Cross* (cross-entropy), *Surr* (SMBO) and *Mut* (champion mutation). For each variant, the population consists of the champion and a single candidate per active module (e.g., BDist has two candidates per iteration), and the maximum number of episodes was fixed to 1000, except for the *Mut* variant, which served as an additional internal baseline and was run until the environments were solved. In MountainCar-v0, we always kept the mutation candidate in the population. Further, we added additional random exploration (30%) to its policy. Random exploration was also added to the environment evaluations of the initial candidates. All remaining policies are always evaluated deterministically. Each run starts with five initial, random candidates, while the elitist experience set contains a maximum of ten archived episode results.

10.4.2 OpenAI Benchmark Results

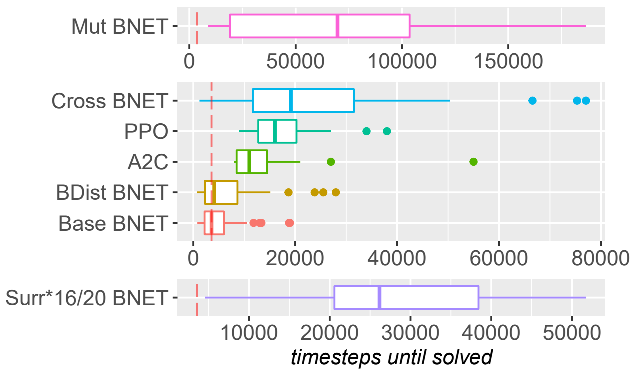


Figure 10.2: *Results of the CartPole-v0 environment. Please mind the different scales for the surrogate and mutation variant. The surrogate variant was only able to finish in 16 out of 20 runs. The best results are achieved by generating all candidates (Base median=3576, red dashed line) or only the behavior distance candidate (BDist with median=4111).*

(CartPole-v0) Figure 10.2 illustrates our results from the CartPole-V0 environment. For the plot, each algorithm was repeated 20 times, except for the *Base* and *BDist* variants, which were repeated 50 times (in order to make more minor differences visible). The surrogate-only variant of BNET only succeeded in 16 out of 20 runs to find a solution in 1000 iterations. Overall, the *BDist* variant only generating the behavioral distance optimized candidate (and keeping the robust champion), and the *Base* version using all proposed search methods were the most successful. As expected, the use of only direct mutation performed slower than all other variants. The overall result was surprising for us, as we expected that one variant beside *Base* would be the best performing, as it includes a relatively large sampling overhead by the larger population. However, the algorithm seems to leverage from a diverse set of candidates, and each search variant seems to contribute to the overall algorithm's performance. To test this assumption, we tracked the candidate type with the best fitness (mean over 100 iterations) in each iteration of the BNET *Base* setup. Figure 10.3 shows the results. The underlying data is from the 50 repeats of the BNET *Base* runs, with 432 iterations.

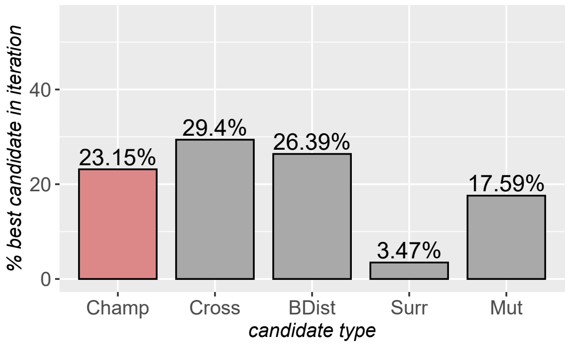


Figure 10.3: *Frequency proportion of each best performing candidate in each iteration of Base BNET for CartPole-v0.*

The plot shows two insights: first, in 77% of the iterations, one of the generated candidates was superior to the stored champion; this implies a reasonable learning rate; second, all candidates of BNET contribute to the performance, where only the surrogate selected candidate performs significantly worse. The surrogate candidate's inferiority could be due to poor parametrization or due to the low number of evaluations, which might not be sufficient to create a proper surrogate model for the complex search space. Interestingly, the direct mutation also generated the best candidates and thus added significantly to the overall performance.

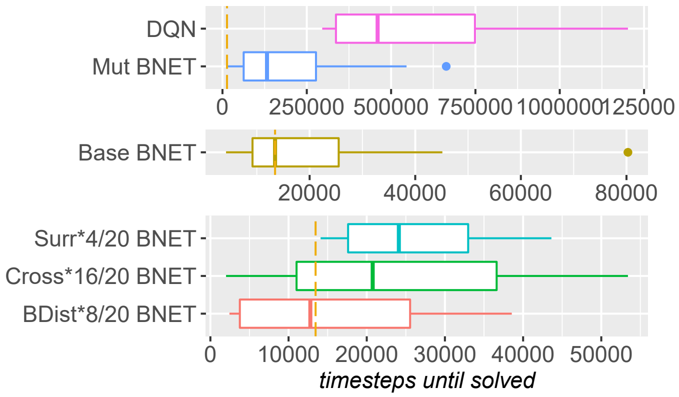


Figure 10.4: *MountainCar-v0 results. Please mind the different scales. The Base variant was able to solve the environment in all cases (median=13457, gold dashed line). The results of the unfinished runs are not comparable. In their case, the attached number of finished runs is meaningful.*

(MountainCar-v0) Due to the challenging nature of this problem, we were not able to find working setups for our baseline algorithms A2C and PPO. All tested parametrization showed no learning effect and got stuck at a reward level of -200, even considering significantly large timestep budgets. We thus tested an additional algorithm, *deep Q networks* (DQN) [Mnih et al., 2013] in available setups (double-DQN, dueling-DQN, prioritized experience replay) and were able to solve the environment using prioritized experience replay [Schaul et al., 2015] and high exploration constants. However, DQN still took a vast number of steps to solve the environment and performs even inferior to the BNET mutation variant. Figure 10.4 displays the MountainCar-v0 results. As visible, the BNET Base variant is dominating this benchmark and remains the only variant that solves the environment in the 100k step limit. Still, the other algorithms' result is quite interesting, as they tend to either solve the environment in a small number of steps or seem to get completely stuck.

As Figure 10.5 illustrates, the percentage of successful candidates in the Base variant per iteration is 47%, much less compared to CartPole-v0, with a clear lead of cross-entropy and mutation. The BDist optimization clearly falls

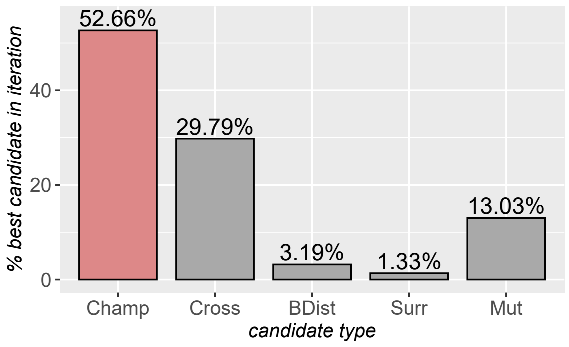


Figure 10.5: *Frequency proportion of each best performing candidate in each iteration of Base BNET for MountainCar-v0.*

behind for this environment, visible in both Figures 10.4 and 10.5. We assume this issue is related to the problematic value estimation due to the flat reward landscape of MountainCar-v0. Again, the surrogate search does not perform as desired, which is also caused by the very flat fitness landscape at the beginning, which does not include much valuable information.

10.4.3 Real-World Robot Maze Setup

Our robot maze problem was explicitly designed to represent a costly real-world demonstrator to test RL algorithms on different setups. We chose a classic maze problem to track and observe an agent's progress and performance efficiently.

The maze consists of a lego brick base plate with 250 x 250 mm, 4x2 black brick walls, and 4x2 white tiles floor, covered by an acrylic glass cover, where a camera is mounted on top. The camera is used to track the position of the red marble in the maze. The system is mounted on a universal robot UR10e 6-axis robotic arm, allowing it to move the maze in all directions. The setup is displayed in Figure 10.6. The target is to move the marble to the upper left position from the starting point. A central challenge in designing real-world problems is learning without manual user interaction (i.e., resetting a robot position). Our demonstrator allows the automatic resetting by flipping the

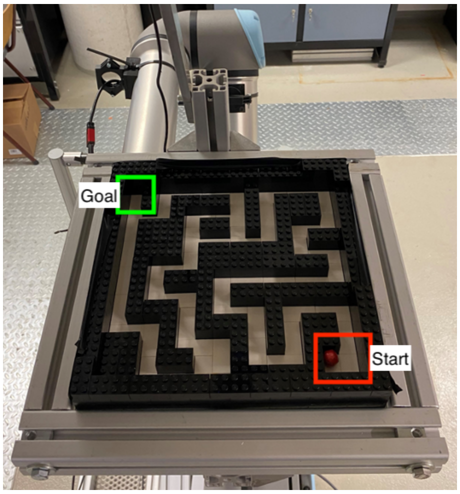


Figure 10.6: *Robot maze test environment mounted on a universal robots UR-10 collaborative robot.*

complete maze and navigating the marble on the glass window to the start. This reset allows episode-to-episode learning and further remote control of the environment without any presence in the lab. The demonstrator is adaptable, i.e., the maze can be redesigned, and the action and observation space can be adapted to discrete or continuous values.

For this work, we restricted the action space to the discrete four cardinal directions and defined designated robot movements, which tilt the maze by an angle of 24,6 and then move back to its base position. Each action takes about 10 seconds and lets the marble roll in a straight line. For the illustrated maze setup, only 23 correct actions are required to reach the target area. The observation space is set to a discrete matrix of 15x15, equal to the maze size, where the marble's current position is set to one, else zero. The reward function forces exploration of the maze by rewarding the agent with 0.1 if he drives the marble to a prior unseen position. If run against a wall, it is penalized by -0.75 and by -0.25 if moved to an already discovered position. Reaching the goal is worth +10. The setup is reset if the goal, a number of 75 steps, or a cumulative reward of -12.5 is reached. The robot maze environment should be fast to learn, as, for most positions, only a single action is correct. However, it requires a perfect mixture of exploration and exploitation to find and learn the correct actions sequentially, as the probability of getting stuck is high, and only a small number of steps is possible in each episode. Moreover, the environment fitness is noisy, as sometimes the marble rolls to a difficult position (i.e., edges of a brick), or the camera position detections are incorrect. Due to the natural time constraints (each run took 8-24h) and several trials to set up the environment, we could not run an extensive experimental design for the benchmark. Thus, we restricted the final results to the most promising ones for this setup: the behavior distance and cross-entropy versions. Each variant was run five times. Regarding the algorithm setup, we set the fitness value in BNET to the number of correctly performed steps. The behavior and cross-entropy distances were weighted by the direct reward instead of a critic and advantage estimation. The purpose-built reward function already delivers correct action value information, and no additional value critic is required. As a baseline, we utilize PPO that showed very stable results in preliminary runs. We set gamma=0.5 due to the apparent correlation between correct actions and rewards and 50 steps per actorbatch for frequent learning.

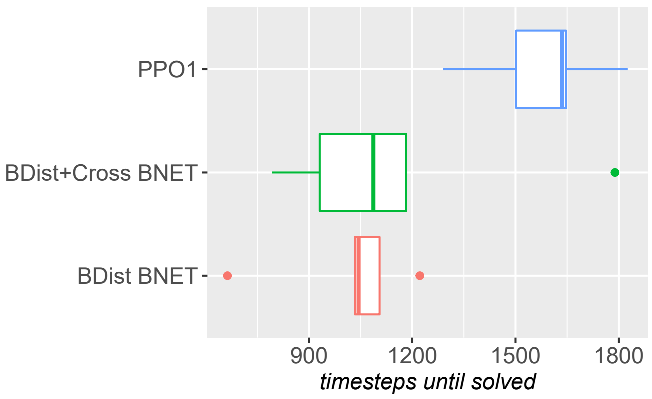


Figure 10.7: *Maze results. Steps until the correct way is learned; The target is consequently reached afterwards. All tested algorithms learn fast. Means (top-down) = 1580, 1156, 1013.*

Figure 10.7 illustrates the results. All pre-selected algorithms were quite fast in solving the problem, with the *BDist* and *BDist+Cross* versions performing best. In Figure 10.8 we visualize the learning progress of the *BDist+Cross* runs. The algorithm advances quite fast and even learns to take more reward-giving actions (>23), assumingly by taking extra detours in the maze. This behavior is similar for PPO and caused by the definition of the rewarding of previous unvisited positions.

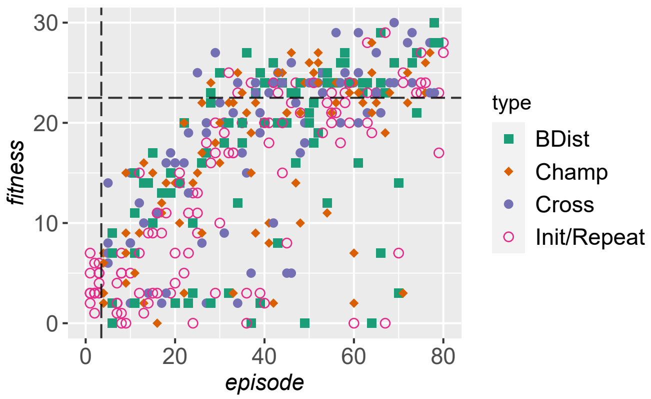


Figure 10.8: *Learning progress of all runs of BDist+Cross BNET on robotmaze. Considered solved at fitness>=23 (horizontal line). BNET shows fast and stable learning process.*

However, the underlying NE-EA with the behavior-based losses demonstrates to be capable of fitting the CGP-ANN policies excellently to the collected experience, even given the sizeable neuroevolutionary search space, and without the use of gradients.

10.5 Conclusion and Outlook

In this work, we investigated methods to combine neuroevolutionary search with standard value-based RL methods. The target was to leverage gathered experiences and create a sample-efficient RL algorithm applicable to real-world problems. We evolve CGP-ANNs as our agents' policies and search for the best network topologies and optimal weights simultaneously. We defined a hybrid, population-based algorithm, called BNET, which utilizes different methods to generate new candidates: direct NE-based mutation, behavior optimization using gathered experience, and finally, behavior-surrogate-based learning. We discovered that the behavior-based search significantly supports the performance. Combining all methods in a population with shared experience and fitness pools leads to excellent sampling efficiency and extraordinary explorative abilities. Moreover, even elementary direct neuroevolutionary mutation steps can contribute significantly to the overall algorithm's performance. As our real-world experiment demonstrates, the defined behavior-loss functions seem well suited to optimize complex networks with changing topologies if the actions' value is estimated correctly. Furthermore, the robust selection with an adaptive re-evaluation of candidates significantly improves our learning progress' stability, as shown in the experiments with a real-world setup. They prove the ability to learn fast and adapt the CGP-ANN policies by solely relying on a gradient-free evolutionary algorithm for optimization. In future work, we want to tackle several open issues:

Framework: We presented an implementation prototype of BNET. The current version is relatively slow due to a single-thread implementation in R. The underlying ideas need to be transferred to faster and computationally more efficient implementations.

Analysis / Tuning: The algorithm structure and parameters need to be profoundly analyzed, understood, and optimized.

Offline initialization: Real-world problems can benefit considerably from experience from prior runs or human demonstrations. We want to implement and test offline-initialization methods.

Environments: We focused on discrete action spaces. Extending benchmarks to continuous action spaces would be interesting.

Modified Real-World Experiment: Our real-world experiment can be adapted to create more challenging RL problems.

11 Concluding Remarks

This thesis gave insights into the matter of behavioral optimization. The foci were set on complex candidate structures, evolutionary and surrogate model-based algorithms, paradigms for comparing solutions, establishing directed search, and implementing behavioral optimization algorithms designed to tackle challenging real-world tasks. This chapter concludes the contributions of this thesis, aggregated into four parts, each dedicated to one of the central requirements stated in the introduction:
(I) overview of optimization algorithms for solving complex problems,
(II) extending SMBO beyond continuous spaces,
(III) genotypic and phenotypic metrics for comparing complex structures, and
(IV) the design and analysis of behavioral optimization in reinforcement learning.

(I) Chapters 2 and 3 concentrated on EC and SMBO in the context of global optimization algorithms for solving problems with complex characteristics. Based on an extensive literature review, a new taxonomy for global optimization algorithms was introduced, specified by the characteristic search elements of each algorithm class. It was determined that the considered algorithms have close connections in their search strategies and share similar components, further allowing the extraction and combination of these components to new hybrid algorithm designs. EAs and SMBO were identified to be most suited to tackle complex problem characteristics: EAs have an excellent generalization ability, and SMBO is frequently employed to increase the sampling efficiency in case of expensive problems. Artificial intelligence applications were highlighted as one of the most promising fields for the application of these algorithms.

An applied study compared evolutionary and surrogate-based approaches to optimize ANN controllers for an elevator group control problem. According to the assumption of their high potential, the results indicated that the algorithms could provide excellent performance for this application. In particular, SMBO improved the sampling efficiency significantly by up to a factor of five, however, at high computation costs for the algorithm.

(II) Chapter 4 analyzed steps for enhancing the applicability of SMBO to combinatorial search spaces as the basis for applying custom distances. Based on considerations from the literature, the requirements for advancing continuous Kriging models were investigated. Besides the improvement of Kriging for this application, a central focus was to evaluate suitable distance measures. In contrast to the standard vector norms (L0, L1, L2) in continuous spaces, a large variety of commonly task-dependent, combinatorial distances exist. Thus, an empirical study featuring 14 different measures and several standard test functions was conducted to analyze their performance. Firstly, this study demonstrated the capabilities of a hybrid SMBO algorithm composed of evolutionarily optimized Kriging surrogates. Secondly, the analysis indicated that the choice of a distance measure has a significant impact on the performance. In this context, it was impossible to identify a generally best distance; on the contrary, almost each test function required a different choice of distance measure.

Ultimately, a dynamic distance selection approach based on maximum likelihood estimation seemed most promising. This insight further illustrates the complexity of comparing solutions in combinatorial spaces, where a genotypic distance requires to be inherited from the problem definition, while a general, task-independent distance is difficult to identify.

(III) Chapters 5, 6, and 7 formalized and analyzed distance measures based on the phenotype or behavior for modeling and SMBO. Three studies featuring candidate solutions with complex structures (genetic programming tree-structures, fixed-topology, and topology-changing neural networks) compared the performance of genotypic and phenotypic distance measures. A part of the research considered the selection of input data to generate behaviors for modeling Kriging with distance-based kernels. It could be shown that behavioral distances, based on high dimensional data sets with several hundreds of observations, could yield sound Kriging models. Also, the behaviors generated by these high dimensional data sets could be well reproduced with only a few components.

The benchmark results of the studies supported the benefits of employing phenotypic distances in SMBO. They performed at least as well as genotypic measures for graph-based trees and fixed-topology neural networks, while applied to neuroevolutionary algorithms, they illustrated clear superiority. All employed phenotypic-based SMBO algorithms could significantly improve the sampling efficiency by up to a factor of ten compared to a model-free approach. The compared genotypic distances often do not correlate well to a candidate's fitness and in addition to that, they are often computationally expensive or even infeasible for modeling (e.g., a total edit distance of two complex graphs). Contrary, the phenotype and its interactions in the environment define a candidate's behavior, which is quickly transformed into fitness. The experiment illustrated that the phenotypic distance measure is insensitive to the ANN size or topology changes.

Ultimately, a phenotypic approach to modeling is independent of a genotype: entirely different genotypes, such as a linear model, symbolic expression, and neural network, can be compared solely by their behavior.

(IV) Chapters 8, 9, and 10 were dedicated to implementing, testing, and analyzing behavior-based optimization for reinforcement learning process. For this final part, topology-changing ANNs generated by CGP-ANN were used as candidates. The genotypes were no longer considered for modeling; however, they still build the basis for any candidate variation. A clear emphasis was put on the implications of employing models and operators utilizing behavior generated by querying the candidate ANN with selected data and the mechanism and implication of this data selection. The leading research aspects were:
a) analyzing the selection of data sets for generating ANN behaviors,
b) in-depth analysis of the correlation between behavior and fitness, and
c) the performance of behavior-based SMBO compared to model-free approaches.

A first empirical study compared fixed data sets (experimental designs, stored data) and dynamic data sets (extracted task-data, changing over consecutive iterations) as input for the behavior distance. It did not provide evidence for significant differences, indicating a robustness of the behavioral distance towards this choice. Still, the best results were achieved with a dynamic task-generated data set, including all known observations of the compared candidates. The dimensionality of the input data has a considerable influence on the computation time, i.e., high-dimensional data sets quadratically increase computation costs.

Behavioral optimization has a set of challenging properties due to its task and input dependence. An in-depth analysis determined that even small changes in the behavior considerably affect fitness, while at the same time, similar fitness can emerge from different behaviors. However, an empirical study exposed that a sufficient behavioral distance correlation can be achieved by selecting an adequate data set for generating the behaviors. It was discovered that behavior-based SMBO significantly reduces the number of required evaluations by up to 80% compared to model-free neuroevolution. Moreover, it was exposed that behavior delivers information that is not covered by fitness alone. Thus, an ideal optimization algorithm should consider both fitness and behavior as the basis for generating candidate solutions.

The final study also considered the direct behavior-based search for candidates using gradient-free neuroevolution. In this case, the challenge was to first identify beneficial behaviors with high fitness. Thus, based on standard methods from reinforcement learning, each decision in the input data was weighted by its advantage to create weighted comparisons, which favor valuable behavior. With this, an even higher level of behavior adaption was realized, illustrating a high performance on standard benchmarks and a real-world application.

Applicability and Proposed Future Work

Table 11.1 summarizes methods introduced in this thesis. Behavioral modeling and optimization demonstrated impressive results for enhancing evolutionary search. Nevertheless, the current implementation is computationally expensive and sensitive to high dimensional data, limiting their applicability. Distance-based Kriging models are well known for their excellent interpolation ability; however, this comes at the cost of expensive modeling. The modeling effort scales significantly with the number of data samples, rendering it frequently computationally infeasible for single node computations. A significant improvement could be achieved by parallelizing Kriging and SMBO in different stages (i.e., matrix computations, parallel evaluations, and model optimization) [Rehbach et al., 2018]. Overall, the computation time renders the outlined solution most helpful if the underlying environments or problems themselves are expensive, i.e., real-world optimization, reinforcement learning, and evolutionary robotics.

Phenotypic or behavioral optimization is best applied for problems with active entities, i.e., embedded in space and time. Using the definition of the 3-fold (*genotype-phenotype-fitness)* and 4-fold (*genotype-phenotype-behavior-fitness*) transitions chain [Eiben and Smith, 2015b], it can be shown that many problems allow the direct fitness evaluation of phenotypes. In this case, direct fitness computations are often computationally more efficient than approximating them in a surrogate-based process. However, expensive problems, such as the hyperparameter tuning of complex data models, are an exception. In this context, a single fact highlights the applicability of behavioral optimization for learning data models: it is built upon metrics that are entirely independent of the structure of the model (genotype). Only the environmental reactions

Table 11.1: *Overview of introduced methods.*

Method	Description	Ch.
SMB-C	Kriging-based surrogate model-based optimization applying custom distance measures.	4
PHD	Phenotypic distance measure. The distance is based on the (static) response of entities evaluated for a task input.	5
SMB-NE	Surrogate model-based optimization for enhancing topology-changing neuroevolutionary algorithms. Applies SMB-C with PHD-based kernels.	7
BD	Behavioral distance measure. Similar to the PHD, but focused on comparing reactions to time-dependent inputs (e.g., consecutive environment states in reinforcement learning).	8
BNET	Behavior-based neuroevolutionary training of neural networks. A hybrid algorithm for reinforcement learning, based on BD, SMB-NE, and value-based RL methods.	10

(behaviors) are of importance. A vast potential arises from evaluating and optimizing diverse data-model structures in a single process.

Behavioral optimization is not limited to enhancing search efficiency; it was also successfully applied to enhance solution diversity [Hagg, 2021]. Applied in methods such as quality diversity or novelty search, it allows producing new surprising solutions, enhancing the creativity in engineering and arts.

Final Words

In conclusion, the thesis is a fundamental step towards exposing the potential of behavioral optimization. The included studies demonstrate the benefits of adding behavior-based search methods to EC and SMBO. The methods for searching the behavioral space provide examples of how to overcome problems of genotypic variation in optimization, such as the complex, non-linear transitions between the genotypes and their evaluated fitness and the concomitant challenging optimization. Behavioral optimization allows controlled adaption of individuals, robustly steering them in the direction of high-performing individuals and ultimately improving the sampling efficiency in optimizing complex tasks.

Bibliography

Abadi M, et al. (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. URL `https://www.tensorflow.org/`

Abdul-Razaq T, Potts C, Wassenhove LV (1990) A survey of algorithms for the single machine total weighted tardiness scheduling problem. Discrete Applied Mathematics 26(2–3):235–253

Aggarwal CC, Hinneburg A, Keim DA (2001) On the surprising behavior of distance metrics in high dimensional space. In: Database Theory — ICDT 2001: 8th International Conference, London, UK, LNCS, pp 420–434

Angeline PJ (1995) Adaptive and self-adaptive evolutionary computations. In: Computational intelligence: A Dynamic Systems Perspective, IEEE Press, pp 152–163

Archetti F, Schoen F (1984) A survey on the global optimization problem: general theory and computational approaches. Annals of Operations Research 1(2):87–110

Arnold DV, Beyer HG (2003) A comparison of evolution strategies with other direct search methods in the presence of noise. Computational Optimization and Applications 24(1):135–159

Arnold DV, Hansen N (2012) A (1+1)-CMA-ES for constrained optimisation. In: Proceedings of the 14th annual conference on Genetic and evolutionary computation, ACM, pp 297–304

Arora J, Elwakeil O, Chahande A, Hsieh C (1995) Global optimization methods for engineering applications: a review. Structural optimization 9(3-4):137–159

Audet C (2014) A survey on direct search methods for blackbox optimization and their applications. In: Mathematics Without Boundaries, Springer, pp 31–56

Augusto DA, Barbosa HJ (2000) Symbolic regression via genetic programming. In: Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks, IEEE, pp 173–178

Bäck T (1996) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press

Bäck T, Fogel DB, Michalewicz Z (1997) Handbook of evolutionary computation. IOP Publishing Ltd.

Baluja S (1994) Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Tech. rep., Carnegie-Mellon University Pittsburgh Department Of Computer Science

Bartz-Beielstein T (2010) SPOT: An R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. arXiv preprint arXiv:10064645

Bartz-Beielstein T, Zaefferer M (2017) Model-based methods for continuous and discrete global optimization. Applied Soft Computing 55:154 – 167, DOI 10.1016/j.asoc.2017.01.039

Bartz-Beielstein T, Lasarczyk C, Preuß M (2005a) Sequential parameter optimization. In: McKay B, et al. (eds) Congress on Evolutionary Computation (CEC'05), Proceedings, IEEE, pp 773–780

Bartz-Beielstein T, Preuss M, Markon S (2005b) Validation and optimization of an elevator simulation model with modern search heuristics. Metaheuristics: Progress as Real Problem Solvers pp 109–128

Basheer IA, Hajmeer M (2000) Artificial neural networks: fundamentals, computing, design, and application. Journal of microbiological methods 43(1):3–31

Beasley JE (1990) OR-Library: distributing test problems by electronic mail. Journal of the Operational Research Society 41(11):1069–1072

Beume N, Naujoks B, Emmerich M (2007) Sms-emoa: Multiobjective selection based on dominated hypervolume. European Journal of Operational Research 181(3):1653–1669

Beyer HG (2013) The theory of evolution strategies. Springer Science & Business Media

Beyer HG, Schwefel HP (2002) Evolution Strategies: A Comprehensive Introduction. Natural Computing 1(1):3–52

Bezerra LC, López-Ibánez M, Stützle T (2014) Automatic design of evolutionary algorithms for multi-objective combinatorial optimization. In: International Conference on Parallel Problem Solving from Nature, Springer, pp 508–517

Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing 11(6):4135–4151

Bongard JC (2013) Evolutionary robotics. Communications of the ACM 56(8):74–83

Booker AJ, Dennis Jr J, Frank PD, Serafini DB, Torczon V, Trosset MW (1999) A rigorous framework for optimization of expensive functions by surrogates. Structural optimization 17(1):1–13

Bosman PA, Thierens D (2000) Continuous iterated density estimation evolutionary algorithms within the idea framework

Bossek J, Doerr C, Kerschke P (2020) Initial design strategies and their effects on sequential model-based optimization: An exploratory case study based on BBOB. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp 778–786, DOI 10.1145/3377930.3390155

Bouhlel MA, Hwang JT, Bartoli N, Lafage R, Morlier J, Martins JR (2019) A python surrogate modeling framework with derivatives. Advances in Engineering Software 135:1–13

Boussaïd I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. Information Sciences 237:82–117

Breiman L (2001) Random forests. Machine Learning 45(1):5–32

Breiman L, Friedman J, Stone CJ, Olshen RA (1984) Classification and regression trees. CRC press

Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) Openai gym. arXiv preprint arXiv:160601540

Burkard RE (1984) Quadratic assignment problems. European Journal of Operational Research 15(3):283 – 289, DOI http://dx.doi.org/10.1016/0377-2217(84)90093-6

Burkard RE, Karisch SE, Rendl F (1997) QAPLIB – a quadratic assignment problem library. Journal of Global Optimization 10(4):391–403, DOI 10.1023/A:1008293323270

Burke E, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyper-heuristics: An emerging direction in modern search technology. In: Handbook of metaheuristics, Springer, pp 457–474

Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward JR (2010) A classification of hyper-heuristic approaches. In: Handbook of metaheuristics, Springer, pp 449–468

Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Özcan E, Qu R (2013) Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society 64(12):1695–1724

Campos V, Laguna M, Martí R (2005) Context-independent scatter and tabu search for permutation problems. INFORMS Journal on Computing 17(1):111–122

Caprara A (1997) Sorting by reversals is difficult. In: Proceedings of the First Annual International Conference on Computational Molecular Biology, ACM, New York, NY, USA, RECOMB '97, pp 75–83, DOI 10.1145/267521.267531

Carson Y, Maria A (1997) Simulation optimization: methods and applications. In: Proceedings of the 29th conference on Winter simulation, IEEE Computer Society, pp 118–126

Chelouah R, Siarry P (2000) Tabu search applied to global optimization. European journal of operational research 123(2):256–270

Chollet F, et al. (2015) Keras. `https://keras.io`

Coello CAC (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Computer methods in applied mechanics and engineering 191(11):1245–1287

Collobert R, Weston J (2008) A unified architecture for natural language processing: Deep neural networks with multitask learning. In: Proceedings of the 25th international conference on Machine learning, ACM, pp 160–167

Conover WJ, Iman RL (1979) On multiple-comparisons procedures. Tech. Rep. LA-7677-MS, Los Alamos Sci. Lab.

Corana A, Marchesi M, Martini C, Ridella S (1987) Minimizing multimodal functions of continuous variables with the “simulated annealing” algorithm. ACM Transactions on Mathematical Software (TOMS) 13(3):262–280

Cowling P, Kendall G, Soubeiga E (2000) A hyperheuristic approach to scheduling a sales summit. In: International Conference on the Practice and Theory of Automated Timetabling, Springer, pp 176–190

Cowling P, Kendall G, Soubeiga E (2002) Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In: Workshops on Applications of Evolutionary Computation, Springer, pp 1–10

Cox DD, John S (1997) Sdo: A statistical method for global optimization. Multidisciplinary design optimization: state of the art pp 315–329

Črepinšek M, Liu SH, Mernik M (2013) Exploration and exploitation in evolutionary algorithms: a survey. ACM Computing Surveys (CSUR) 45(3):35

Crombecq K, Gorissen D, Deschrijver D, Dhaene T (2011) A novel hybrid sequential design strategy for global surrogate modeling of computer experiments. SIAM Journal on Scientific Computing 33(4):1948–1974

Daniels SJ, Rahat AAM, Everson RM, Tabor GR, Fieldsend JE (2018) A suite of computationally expensive shape optimisation problems using computational fluid dynamics. In: International Conference on Parallel Problem Solving from Nature, pp 296–307

Dawkins R (1982) The Extended Phenotype. Oxford University Press Oxford

De Grave K, Ramon J, De Raedt L (2008) Active learning for high throughput screening. In: International Conference on Discovery Science, pp 185–196

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation 6(2):182–197

Doerr B, Doerr C, Yang J (2020) Optimal parameter choices via precise black-box analysis. Theoretical Computer Science 801:1–34

Doncieux S, Mouret JB (2010) Behavioral diversity measures for evolutionary robotics. In: IEEE Congress on Evolutionary Computation, pp 1–8

Doncieux S, Mouret JB (2014) Beyond black-box optimization: a review of selective pressures for evolutionary robotics. Evolutionary Intelligence 7(2):71–93, DOI 10.1007/s12065-014-0110-x

Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE computational intelligence magazine 1(4):28–39

Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research 12(7):2121–2159

Eiben AE, Smith J (2015a) From evolutionary computation to the evolution of things. Nature 521(7553):476–482

Eiben AE, Smith JE (2015b) Introduction to Evolutionary Computing, 2nd edn. Springer

Eiben ÁE, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. IEEE Transactions on evolutionary computation 3(2):124–141

Emmerich MT, Giannakoglou KC, Naujoks B (2006) Single-and multiobjective evolutionary optimization assisted by gaussian random field metamodels. IEEE Transactions on Evolutionary Computation 10(4):421–439

Faber R, Jockenhövel T, Tsatsaronis G (2005) Dynamic optimization with simulated annealing. Computers & chemical engineering 29(2):273–290

Filipiak P, Lipinski P (2014) Infeasibility driven evolutionary algorithm with feed-forward prediction strategy for dynamic constrained optimization problems. In: Applications of Evolutionary Computation, Springer, pp 817–828

Flasch O (2015) A modular genetic programming system. PhD thesis, TU Dortmund, DOI http://dx.doi.org/10.17877/DE290R-7807

Flasch O, Bartz-Beielstein T, Davtyan A, Koch P, Konen W, Oyetoyan TD, Tamutan M (2010a) Comparing CI methods for prediction models in environmental engineering. In: Proc. of CEC

Flasch O, Mersmann O, Bartz-Beielstein T (2010b) RGP: An open source genetic programming system for the R environment. In: Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation, pp 2071–2072, DOI 10.1145/1830761.1830867

Flasch O, Mersmann O, Bartz-Beielstein T, Stork J, Zaefferer M (2014) rgp: R genetic programming framework. URL `https://CRAN.R-project.org/package=rgp`, r package version 0.4-1

Fleming PJ, Purshouse RC (2002) Evolutionary algorithms in control systems engineering: a survey. Control engineering practice 10(11):1223–1241

Fletcher R (1976) Conjugate gradient methods for indefinite systems. In: Numerical analysis, Springer, pp 73–89

Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution. John Wiley

Fomin FV, Kaski P (2013) Exact exponential algorithms. Commun ACM 56(3):80–88, DOI 10.1145/2428556.2428575, URL `http://doi.acm.org/10.1145/2428556.2428575`

Fonseca CM, Fleming PJ, et al. (1993) Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: Proceedings of the 5th International Conference on Genetic Algorithms, vol 93, pp 416–423

Forrester A, Sobester A, Keane A (2008) Engineering Design via Surrogate Modelling. John Wiley & Sons

Forrester AI, Keane AJ (2009) Recent advances in surrogate-based optimization. Progress in Aerospace Sciences 45(1):50–79

Franke JK, Köhler G, Awad N, Hutter F (2019) Neural architecture evolution in deep reinforcement learning for continuous control. arXiv preprint arXiv:191012824

Friese M, Bartz-Beielstein T, Emmerich MTM (2016) Building ensembles of surrogates by optimal convex combination. In: Mernik M, Papa G (eds) Bioinspired Optimization Methods and their Applications, pp 131–144

Gablonsky JM, Kelley CT (2001) A locally-biased form of the direct algorithm. Journal of Global Optimization 21(1):27–37

Gaier A, Asteroth A, Mouret JB (2018) Data-efficient neuroevolution with kernel-based surrogate models. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp 85–92

Gallagher M, Frean MR, Downs T (1999) Real-valued evolutionary optimization using a flexible probability density estimator. In: GECCO, vol 99, pp 840–846

Gämperle R, Müller SD, Koumoutsakos P (2002) A parameter study for differential evolution. Advances in intelligent systems, fuzzy systems, evolutionary computation 10:293–298

Glover F (1989) Tabu search—part I. ORSA Journal on computing 1(3):190–206

Goel T, Haftka RT, Shyy W, Queipo NV (2007) Ensemble of surrogates. Structural and Multidisciplinary Optimization 33(3):199–216

Goffe WL, Ferrier GD, Rogers J (1994) Global optimization of statistical functions with simulated annealing. Journal of Econometrics 60(1):65–99

Guo H (2003) A bayesian approach for automatic algorithm selection. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI03), Workshop on AI and Autonomic Computing, Acapulco, Mexico, pp 1–5

Haftka RT, Villanueva D, Chaudhuri A (2016) Parallel surrogate-assisted global optimization with expensive functions–a survey. Structural and Multidisciplinary Optimization 54(1):3–13

Hagg A (2021) Discovering the preference hypervolume, an interactive model for real world computational co-creativity. PhD thesis, Leiden Institute of Advanced Computer Science (LIACS), Faculty of Science, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands

Hagg A, Zaefferer M, Stork J, Gaier A (2019) Prediction of neural network performance by phenotypic modeling. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion - GECCO'19, ACM, Prague, Czech Republic, GECCO '19, pp 1576–1582, DOI 10.1145/3319619.3326815

Hansen N (2006) The CMA evolution strategy: a comparing review. In: Towards a new evolutionary computation, Springer, pp 75–102

Hansen N, Müller SD, Koumoutsakos P (2003) Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary computation 11(1):1–18

Hansen N, Auger A, Finck S, Ros R (2010a) Real-parameter black-box optimization benchmarking 2010: Experimental setup. Tech. rep., INRIA

Hansen N, Auger A, Ros R, Finck S, Pošík P (2010b) Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In: Proceedings of the 12th annual conference companion on Genetic and evolutionary computation, ACM, pp 1689–1696

Hansen P, Mladenovic N (2003) Variable neighbourhood search. Handbook of Metaheuristics, Dordrecht, Kluwer Academic Publishers

Hansen P, Mladenović N, Pérez JAM (2010c) Variable neighbourhood search: methods and applications. Annals of Operations Research 175(1):367–407

Harik G, et al. (1999) Linkage learning via probabilistic modeling in the ecga. IlliGAL report 99010

Hauschild M, Pelikan M (2011) An introduction and survey of estimation of distribution algorithms. Swarm and Evolutionary Computation 1(3):111–128

Haykin S (2004) Neural Networks: A comprehensive foundation. Prentice Hall, Upper Saddle River, New Jersey

Henderson D, Jacobson SH, Johnson AW (2003) The theory and practice of simulated annealing. In: Handbook of metaheuristics, Springer, pp 287–319

Hildebrandt T, Branke J (2015) On using surrogates with genetic programming. Evolutionary computation 23(3):343–367

Hill A, Raffin A, Ernestus M, Gleave A, Kanervisto A, Traore R, Dhariwal P, Hesse C, Klimov O, Nichol A, Plappert M, Radford A, Schulman J, Sidor S, Wu Y (2018) Stable baselines. `https://github.com/hill-a/stable-baselines`

Hinton G, Deng L, Yu D, Dahl GE, Mohamed Ar, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN, et al. (2012) Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine 29(6):82–97

Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural computation 18(7):1527–1554

Hirschberg DS (1975) A linear space algorithm for computing maximal common subsequences. Communications of the ACM 18(6):341–343

Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press

Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural networks 2(5):359–366

Hu N (1992) Tabu search method with random moves for globally optimal design. International Journal for Numerical Methods in Engineering 35(5):1055–1070

Hutter F (2009) Automated configuration of algorithms for solving hard computational problems. PhD thesis, University of British Columbia

Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. LION 5:507–523

Izzo D, Biscani F, Mereta A (2017) Differentiable genetic programming. In: European conference on genetic programming, Springer, pp 35–51

Jaderberg M, Dalibard V, Osindero S, Czarnecki WM, Donahue J, Razavi A, Vinyals O, Green T, Dunning I, Simonyan K, et al. (2017) Population based training of neural networks. arXiv preprint arXiv:171109846

Jin Y (2005) A comprehensive survey of fitness approximation in evolutionary computation. Soft Computing-A Fusion of Foundations, Methodologies and Applications 9(1):3–12

Jin Y (2011) Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm and Evolutionary Computation 1(2):61–70

Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments-a survey. IEEE Transactions on evolutionary computation 9(3):303–317

Jin Y, Wang H, Chugh T, Guo D, Miettinen K (2019) Data-driven evolutionary optimization: An overview and case studies. IEEE Transactions on Evolutionary Computation 23(3):442–458, DOI 10.1109/TEVC.2018.2869001

Jones DR (2001) A taxonomy of global optimization methods based on response surfaces. Journal of global optimization 21(4):345–383

Jones DR, Schonlau M, Welch WJ (1998) Efficient global optimization of expensive black-box functions. Journal of Global Optimization 13(4):455–492

Jones T, Forrest S (1995) Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann, Pittsburgh, PA, USA, pp 184–192

Jung W, Park G, Sung Y (2020) Population-guided parallel policy search for reinforcement learning. arXiv preprint arXiv:200102907

Kallel L, Schoenauer M (1996) Fitness distance correlation for variable length representations. Tech. Rep. 363, CMAP, Ecole Polytechnique

Kattan A, Ong YS (2015) Surrogate genetic programming: A semantic aware evolutionary search. Information Sciences 296:345–359

Kendall MG, Gibbons JD (1990) Rank Correlation Methods. Charles Griffin Book Series, Oxford University Press, London

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Neural Networks, 1995. Proceedings., IEEE International Conference on, IEEE, vol 4, pp 1942–1948

Kerschke P, Trautmann H (2019) Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. Evolutionary computation 27(1):99–127

Khadka S, Tumer K (2018) Evolution-guided policy gradient in reinforcement learning. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp 1196–1208

Khadka S, Majumdar S, Nassar T, Dwiel Z, Tumer E, Miret S, Liu Y, Tumer K (2019) Collaborative evolutionary reinforcement learning. In: International Conference on Machine Learning, PMLR, pp 3341–3350

Khan MM, Khan GM, Miller JF (2010) Evolution of neural networks using cartesian genetic programming. In: IEEE Congress on Evolutionary Computation, pp 1–8, DOI 10.1109/CEC.2010.5586547

Khan N, Goldberg DE, Pelikan M (2002) Multi-objective bayesian optimization algorithm. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, GECCO'02, pp 684–684

Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. science 220(4598):671–680

Kolda TG, Lewis RM, Torczon V (2003) Optimization by direct search: New perspectives on some classical and modern methods. SIAM review 45(3):385–482

Koppejan R, Whiteson S (2011) Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. Evolutionary Intelligence 4(4):219–241, DOI 10.1007/s12065-011-0066-z

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT press

Koziel S, Leifsson L (2013) Surrogate-based modeling and optimization. Springer

Kruskal WH, Wallis WA (1952) Use of ranks in one-criterion variance analysis. Journal of the American Statistical Association 47:583–621

Kushner HJ (1964) A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. Journal of Basic Engineering 86(1):97–106

Larrañaga P, Lozano JA (2001) Estimation of distribution algorithms: A new tool for evolutionary computation, vol 2. Springer Science & Business Media

Larranaga P, Etxeberria R, Lozano J, Pena J, Pe J, et al. (1999) Optimization by learning and simulation of bayesian and gaussian networks. Tech. rep., University of the Basque Country

Lawler EL, Wood DE (1966) Branch-and-bound methods: A survey. Operations research 14(4):699–719

Lee C (1958) Some properties of nonbinary error-correcting codes. Information Theory, IRE Transactions on 4(2):77–82, DOI 10.1109/TIT.1958.1057446

Leon A (1966) A classified bibliography on optimization. Recent Advances in Optimization Techniques, John Wiley & Sons, Inc, New York and London pp 599–649

Lewis RM, Torczon V, Trosset MW (2000) Direct search methods: then and now. Journal of computational and Applied Mathematics 124(1):191–207

Li R, Emmerich MTM, Eggermont J, Bovenkamp EGP, Bäck T, Dijkstra J, Reiber J (2008) Metamodel-assisted mixed integer evolution strategies and their application to intravascular ultrasound image analysis. In: Congress on Evolutionary Computation, IEEE, pp 2764–2771, DOI 10.1109/CEC.2008.4631169

Liaw A, Wiener M, et al. (2002) Classification and regression by randomforest. R news 2(3):18–22

Lim D, Jin Y, Ong YS, Sendhoff B (2010) Generalizing surrogate-assisted evolutionary computation. IEEE Transactions on Evolutionary Computation 14(3):329–355

Lindauer M, Hoos HH, Hutter F, Schaub T (2015) Autofolio: An automatically configured algorithm selector. Journal of Artificial Intelligence Research 53:745–778

Liu DC, Nocedal J (1989) On the limited memory bfgs method for large scale optimization. Mathematical programming 45(1-3):503–528

Lizotte DJ (2008) Practical bayesian optimization. PhD thesis, University of Alberta

Lobo F, Lima CF, Michalewicz Z (2007) Parameter setting in evolutionary algorithms, Studies in Computational Intelligence, vol 54. Springer Science & Business Media

Locatelli M (2002) Simulated annealing algorithms for continuous global optimization. In: Handbook of global optimization, Springer, pp 179–229

López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3:43–58

Loshchilov I, Schoenauer M, Sebag M (2012) Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In: Proceedings of the 14th annual conference on Genetic and evolutionary computation, ACM, pp 321–328

Marler RT, Arora JS (2004) Survey of multi-objective optimization methods for engineering. Structural and multidisciplinary optimization 26(6):369–395

Marsden AL, Wang M, Dennis Jr JE, Moin P (2004) Optimal aeroacoustic shape design using the surrogate management framework. Optimization and Engineering 5(2):235–262

Martin MA, Tauritz DR (2013) Evolving black-box search algorithms employing genetic programming. In: Proceedings of the 15th annual conference companion on Genetic and evolutionary computation, pp 1497–1504

McKay B, Willis MJ, Barton GW (1995) Using a tree structured genetic algorithm to perform symbolic regression. In: Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414), IET, pp 487–492

McKay MD, Beckman RJ, Conover WJ (1979) Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21(2):239–245

Mebane Jr WR, Sekhon JS (2011) Genetic optimization using derivatives: the rgenoud package for r. Journal of Statistical Software 42(11):1–26

Mercer RE, Sampson J (1978) Adaptive search using a reproductive meta-plan. Kybernetes 7(3):215–228

Mersmann O, Bischl B, Trautmann H, Preuss M, Weihs C, Rudolph G (2011) Exploratory landscape analysis. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, Association for Computing Machinery, New York, NY, USA, GECCO '11, pp 829–836, DOI 10.1145/2001576.2001690

Meyerson E, Lehman J, Miikkulainen R (2016) Learning behavior characterizations for novelty search. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, Association for Computing Machinery, New York, NY, USA, GECCO '16, pp 149–156, DOI 10.1145/2908812.2908929

Miller JF, Thomson P (2000) Cartesian genetic programming. In: European Conference on Genetic Programming, Springer, pp 121–132

Mladenović N, Dražić M, Kovačevic-Vujčić V, Čangalović M (2008) General variable neighborhood search for the continuous optimization. European Journal of Operational Research 191(3):753–770

Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602

Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In: International conference on machine learning, PMLR, pp 1928–1937

Mockus J (1974) On bayesian methods for seeking the extremum. In: Proceedings of the IFIP Technical Conference, Springer-Verlag, pp 400–404

Mockus J (1994) Application of bayesian approach to numerical methods of global and stochastic optimization. Journal of Global Optimization 4(4):347–365

Mockus J (2012) Bayesian approach to global optimization: theory and applications, vol 37. Springer Science & Business Media

Molina D, Lozano M, García-Martínez C, Herrera F (2010) Memetic algorithms for continuous optimisation based on local search chains. Evolutionary Computation 18(1):27–63

Montgomery DC (2017) Design and analysis of experiments, 9th edn. John Wiley & Sons, Inc.

Moraglio A, Kattan A (2011a) Geometric generalisation of surrogate model based optimisation to combinatorial spaces. In: Proceedings of the 11th European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, Berlin, Heidelberg, Germany, EvoCOP'11, pp 142–154

Moraglio A, Kattan A (2011b) Geometric surrogate model based optimisation for genetic programming: Initial experiments. Tech. rep., University of Birmingham

Moraglio A, Poli R (2005) Geometric landscape of homologous crossover for syntactic trees. In: 2005 IEEE Congress on Evolutionary Computation, IEEE, Edinburgh, UK, vol 1, pp 427–434, DOI 10.1109/cec.2005.1554715

Moraglio A, Krawiec K, Johnson CG (2012) Geometric semantic genetic programming. In: International Conference on Parallel Problem Solving from Nature, Springer, pp 21–31

Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech concurrent computation program 826:1989

Mouret JB, Clune J (2015) Illuminating search spaces by mapping elites. arXiv e-prints ArXiv:1504.04909v1

Mouret JB, Doncieux S (2012) Encouraging behavioral diversity in evolutionary robotics: An empirical study. In: Evolutionary Computation, pp 91–133

Mullen K, Ardia D, Gil DL, Windover D, Cline J (2011) DEoptim: An R package for global optimization by differential evolution. Journal of Statistical Software 40(6):1–26

Mullen KM (2014) Continuous global optimization in R. Journal of Statistical Software 60(6):1–45

Müller J (2016) MISO: mixed-integer surrogate optimization framework. Optimization and Engineering 17(1):177–203

Müller J, Shoemaker CA (2014) Influence of ensemble surrogate models and sampling strategy on the solution quality of algorithms for computationally expensive black-box global optimization problems. Journal of Global Optimization 60(2):123–144

Naujoks B, Beume N, Emmerich M (2005) Multi-objective optimisation using s-metric selection: Application to three-dimensional solution spaces. In: Evolutionary Computation, 2005. The 2005 IEEE Congress on, IEEE, vol 2, pp 1282–1289

Nelder JA, Mead R (1965) A simplex method for function minimization. The Computer Journal 7(4):308–313

Neumaier A (2004) Complete search in continuous global optimization and constraint satisfaction. Acta numerica 13:271–369

Ng AY, Russell SJ, et al. (2000) Algorithms for inverse reinforcement learning. In: Icml, vol 1, pp 663–670

Nguyen S, Zhang M, Johnston M, Tan KC (2014) Selection schemes in surrogate-assisted genetic programming for job shop scheduling. In: Simulated Evolution and Learning, 10th International Conference, SEAL, Springer Science + Business Media, pp 656–667, DOI 10.1007/978-3-319-13563-2_55

Nguyen S, Zhang M, Tan KC (2016) Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. IEEE Transactions on Cybernetics pp 1–15, DOI 10.1109/tcyb.2016.2562674

Nguyen S, Mei Y, Zhang M (2017) Genetic programming for production scheduling: a survey with a unified framework. Complex & Intelligent Systems 3(1):41–66, DOI 10.1007/s40747-017-0036-x

Ollion C, Doncieux S (2011) Why and how to measure exploration in behavioral space. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, Association for Computing Machinery, New York, NY, USA, GECCO '11, pp 267–274, DOI 10.1145/2001576.2001613

Ong YS, Nair PB, Keane AJ (2003) Evolutionary optimization of computationally expensive problems via surrogate modeling. AIAA Journal 41(4):687–696

Ong YS, Nair P, Keane A, Wong K (2005) Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. In: Knowledge Incorporation in Evolutionary Computation, Springer, pp 307–331

Papa G, Doerr C (2020) Dynamic control parameter choices in evolutionary computation: Gecco 2020 tutorial. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, pp 927–956

Parisotto E, Mohamed A, Singh R, Li L, Zhou D, Kohli P (2016) Neuro-symbolic program synthesis. ArXiv e-prints 1611.01855, `1611.01855`

Parker-Holder J, Pacchiano A, Choromanski K, Roberts S (2020) Effective diversity in population-based reinforcement learning. arXiv preprint arXiv:200200632

Pawlik M, Augsten N (2015) Efficient computation of the tree edit distance. ACM Transactions on Database Systems 40(1):1–40, DOI 10.1145/2699485

Pawlik M, Augsten N (2016a) APTED release 0.1.1. GitHub, `https://github.com/DatabaseGroup/apted`, last accessed: 2017-06-01

Pawlik M, Augsten N (2016b) Tree edit distance: Robust and memory-efficient. Information Systems 56:157–173, DOI 10.1016/j.is.2015.08.004

Pearl J (1985) Heuristics. intelligent search strategies for computer problem solving. The Addison-Wesley Series in Artificial Intelligence, Reading, Mass: Addison-Wesley, 1985, Reprinted version

Pittenger AO (2012) An introduction to quantum computing algorithms, vol 19. Springer Science & Business Media

Pohlert T (2018) PMCMRplus: calculate pairwise multiple comparisons of mean rank sums extended - R package, version 1.4.1

Pugh JK, Soros LB, Stanley KO (2016) Searching for quality diversity when diversity is unaligned with quality. In: International Conference on Parallel Problem Solving from Nature, Springer, pp 880–889

Queipo NV, Haftka RT, Shyy W, Goel T, Vaidyanathan R, Tucker PK (2005) Surrogate-based analysis and optimization. Progress in aerospace sciences 41(1):1–28

R Core Team (2018) R: A language and environment for statistical computing

Rasmussen CE (2004) Gaussian processes in machine learning. In: Advanced Lectures on Machine Learning, Springer

Rechenberg I (1973) Evolutionsstrategie–Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog

Rechenberg I (1994) Evolutionsstrategie ’94. Frommann-Holzboog

Reeves CR (1995) A genetic algorithm for flowshop sequencing. Computers & operations research 22(1):5–13

Reeves CR (1999) Landscapes, operators and heuristic search. Annals of Operations Research 86:473–490

Regis RG, Shoemaker CA (2013) Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. Engineering Optimization 45(5):529–555

Rehbach F, Zaefferer M, Stork J, Bartz-Beielstein T (2018) Comparison of parallel surrogate-assisted optimization approaches. In: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, pp 1348–1355

Rehbach F, Zaefferer M, Naujoks B, Bartz-Beielstein T (2020) Expected improvement versus predicted value in surrogate-based optimization. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp 868–876

Reinelt G (1991) TSPLIB—A traveling salesman problem library. ORSA journal on computing 3(4):376–384

Richter SN, Tauritz DR (2018) The automated design of probabilistic selection methods for evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp 1545–1552

van Rijn S, Wang H, van Stein B, Bäck T (2017) Algorithm configuration data mining for CMA evolution strategies. In: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA, GECCO '17, pp 737–744, DOI 10.1145/3071178.3071205

Rozenberg G, Bäck T, Kok JN (eds) (2012) Handbook of natural computing, vol 1. Springer

Ruder S (2016) An overview of gradient descent optimization algorithms. arXiv preprint arXiv:160904747

Sacks J, Welch WJ, Mitchell TJ, Wynn HP (1989) Design and analysis of computer experiments. Statistical Science 4(4):409–435

Salimans T, Ho J, Chen X, Sidor S, Sutskever I (2017) Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:170303864

Santana R (2017) Gray-box optimization and factorized distribution algorithms: where two worlds collide. arXiv preprint arXiv:170703093

Schaffer JD, Whitley D, Eshelman LJ (1992) Combinations of genetic algorithms and neural networks: A survey of the state of the art. In: COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks, IEEE, pp 1–37

Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized experience replay. arXiv preprint arXiv:151105952

Schiavinotto T, Stützle T (2007) A review of metrics on permutations for search landscape analysis. Computers & operations research 34(10):3143–3153

Schmitt S, Hessel M, Simonyan K (2020) Off-policy actor-critic with shared experience replay. In: International Conference on Machine Learning, PMLR, pp 8545–8554

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:170706347

Schuman CD, Potok TE, Patton RM, Birdwell JD, Dean ME, Rose GS, Plank JS (2017) A survey of neuromorphic computing and neural networks in hardware. arXiv preprint arXiv:170506963

Schwefel HP (1977) Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie, 1st edn. Interdisciplinary Systems Research, Birkhäuser

Schwefel HPP (1993) Evolution and optimum seeking: the sixth generation. John Wiley & Sons, Inc.

Scrucca L (2013) GA: a package for genetic algorithms in R. Journal of Statistical Software 53(4):1–37

Sekhon JS, Mebane WR (1998) Genetic optimization using derivatives. Political Analysis 7:187–210

Sevaux M, Sörensen K (2005) Permutation distance measures for memetic algorithms with population management. In: Metaheuristics International Conference, pp 832–838

Shanno DF (1970) Conditioning of quasi-newton methods for function minimization. Mathematics of computation 24(111):647–656

Shi Y, Eberhart R (1998) A modified particle swarm optimizer. In: Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, IEEE, pp 69–73

Shir OM, Bäck T (2005) Niching in evolution strategies. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation, ACM, pp 915–916

Siarry P, Berthiau G (1997) Fitting of tabu search to optimize functions of continuous variables. International journal for numerical methods in engineering 40(13):2449–2457

Siarry P, Berthiau G, Durdin F, Haussy J (1997) Enhanced simulated annealing for globally minimizing functions of many-continuous variables. ACM Transactions on Mathematical Software (TOMS) 23(2):209–228

Smit S, Eiben A (2011) Multi-problem parameter tuning using bonesa. In: Artificial Evolution, pp 222–233

Snoek J, Larochelle H, Adams RP (2012) Practical bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems, pp 2951–2959

Søndergaard J (2003) Optimization using surrogate models-by the space mapping technique. PhD thesis, Technical University of Denmark

Stanley KO (2006) Exploiting regularity without development. In: Proceedings of the AAAI Fall Symposium on Developmental Systems, AAAI Press, pp 49–56

Stanley KO, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evolutionary computation 10(2):99–127

Stanley KO, Clune J, Lehman J, Miikkulainen R (2019) Designing neural networks through neuroevolution. Nature Machine Intelligence 1(1):24–35

Stork J, Zaefferer M, Fischbach A, Bartz-Beielstein T (2017) Surrogate-assisted learning of neural networks. In: Proceedings 27. GMA Workshop Computational Intelligence, KIT Scientific Publishing, pp 195–210

Stork J, Zaefferer M, Bartz-Beielstein T (2018) Distance-based kernels for surrogate model-based neuroevolution. ArXiv e-prints

Stork J, Zaefferer M, Bartz-Beielstein T (2019a) Improving neuroevolution efficiency by surrogate model-based optimization with phenotypic distance kernels. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, pp 504–519

Stork J, Zaefferer M, Bartz-Beielstein T, Eiben AE (2019b) Surrogate models for enhancing the efficiency of neuroevolution in reinforcement learning. In: Proceedings of the Genetic and Evolutionary Computation Conference - GECCO'19, ACM, Prague, Czech Republic, pp 934–942, DOI 10.1145/3321707.3321829

Stork J, Zaefferer M, Bartz-Beielstein T, Eiben A (2020) Understanding the behavior of reinforcement learning agents. In: International Conference on Bioinspired Methods and Their Applications, Springer, pp 148–160

Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11(4):341–359

Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press

Swersky K, Snoek J, Adams RP (2013) Multi-task bayesian optimization. In: Advances in neural information processing systems, pp 2004–2012

Taillard E (1990) Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research 47(1):65–74

Talbi EG (2002) A taxonomy of hybrid metaheuristics. Journal of heuristics 8(5):541–564

Talbi EG (2009) Metaheuristics: from design to implementation, vol 74. John Wiley & Sons

Törn A, Zilinskas A (1989) Global Optimization. Springer

Turner AJ, Miller JF (2013) Cartesian genetic programming encoded artificial neural networks: a comparison using three benchmarks. In: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, pp 1005–1012

Turner AJ, Miller JF (2015) Introducing a cross platform open source cartesian genetic programming library. Genetic Programming and Evolvable Machines 16(1):83–91

Van Beers WC, Kleijnen JP (2003) Kriging for interpolation in random simulation. Journal of the Operational Research Society 54(3):255–262

Van Groenigen J, Stein A (1998) Constrained optimization of spatial sampling using continuous simulated annealing. Journal of Environmental Quality 27(5):1078–1086

Venables WN, Ripley BD (2002) Modern Applied Statistics with S, 4th edn. Springer

Vermetten D, van Rijn S, Bäck T, Doerr C (2019) Online selection of CMA-ES variants. In: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA, GECCO '19, pp 951–959, DOI 10.1145/3321707.3321803

Voutchkov I, Keane A, Bhaskar A, Olsen TM (2005) 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, DOI 10.1016/j.cma.2005.02.003

Wang H, Jin Y, Jansen JO (2016) Data-driven surrogate-assisted multiobjective evolutionary optimization of a trauma system. IEEE Transactions on Evolutionary Computation 20(6):939–952

Wessing S, Preuss M (2017) The true destination of ego is multi-local optimization. In: 2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI), IEEE, pp 1–6

Whitley LD, Chicano F, Goldman BW (2016) Gray box optimization for Mk landscapes (NK landscapes and MAX-kSAT). Evolutionary computation 24(3):491–519

Woeginger GJ (2003) Exact algorithms for np-hard problems: A survey. In: Combinatorial optimization—eureka, you shrink!, Springer, pp 185–207

Won KS, Ray T (2004) Performance of kriging and cokriging based surrogate models within the unified framework for surrogate assisted optimization. In: Evolutionary Computation, 2004. CEC2004. Congress on, IEEE, vol 2, pp 1577–1585

Xiang Y, Gubian S, Suomela B, Hoeng J (2013) Generalized simulated annealing for global optimization: The GenSA package. R Journal 5(1)

Zaefferer M (2017) Combinatorial efficient global optimization in R - CEGO v2.2.0. online, URL `https://cran.r-project.org/package=CEGO`, accessed: 2018-01-10

Zaefferer M (2018) Surrogate models for discrete optimization problems. phdthesis, Technische Universität Dortmund

Zaefferer M, Stork J, Bartz-Beielstein T (2014a) Distance measures for permutations in combinatorial efficient global optimization. In: International Conference on Parallel Problem Solving from Nature, Springer, pp 373–383

Zaefferer M, Stork J, Friese M, Fischbach A, Naujoks B, Bartz-Beielstein T (2014b) Efficient global optimization for combinatorial problems. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp 871–878

Zaefferer M, Stork J, Flasch O, Bartz-Beielstein T (2018) Linear combination of distance measures for surrogate models in genetic programming. In: Parallel Problem Solving from Nature – PPSN XV: 15th International Conference, Springer, Coimbra, Portugal, vol 11102, pp 220–231, DOI 10.1007/978-3-319-99259-4_18

Zeng Z, Tung AKH, Wang J, Feng J, Zhou L (2009) Comparing stars: On approximating graph edit distance. Proc VLDB Endow 2(1):25–36

Zlochin M, Birattari M, Meuleau N, Dorigo M (2004) Model-based search for combinatorial optimization: A critical survey. Annals of Operations Research 131(1-4):373–395

Zuo X (2018) mazelab: A customizable framework to create maze and gridworld environments. `https://github.com/zuoxingdong/mazelab`

Way of the Fittest: Optimization by Behavioral Evolution

Jörg Willi Stork

Vrije Universiteit Amsterdam

Evolutionary computation (EC) methods belong to the state-of-the-art for solving optimization problems with complex characteristics, such as no available analytical descriptions or no differentiability. One outstanding EC paradigm in artificial intelligence applications is neuroevolution, a method to construct artificial neural networks (ANN) and optimize their weights, parameters, and topologies. Neuroevolutionary optimization can be challenging due to complex and costly environments in which they are applied, multi-modal fitness landscapes, and the number of degrees of freedom necessary to generate ANNs that can perform in these environments.

Surrogate model-based optimization (SMBO) can improve search efficiency by approximating complex fitness landscapes and predicting high-performing candidate solutions. Surrogate models often rely on distance metrics that determine the correlation of a candidate solution's fitness to that of similar individuals. However, the employment of distances for complex-structured candidates, such as ANNs, is difficult due to their non-linear relationships between their encoding (genotype) and realization (phenotype). A possible solution is to compare candidates based on their observable behavior, i.e., an individual's decisions, actions, and movements in an environment. Task-dependent behavior has a strong connection to fitness, which renders optimization in this space of behaviors promising. The central concept of this thesis is to research, analyze and develop methods that exploit the strong connection between an individual's behavior, environment, and fitness. The included publications are, in their entirety, a fundamental step towards establishing behavioral optimization.

Initially, an extensive introduction to EC and SMBO in the greater context of global optimization is given to allow an in-depth understanding of their concepts, particularities, and various available implementations. A new taxonomy

is presented, and it is determined that global optimization algorithms have close connections in their search strategies and share similar components, which allows combining them into new hybrid algorithm designs. In large parts of this thesis, data-drive Kriging models were employed for the task of SMBO as a flexible and accurate predictor model within the SMBO paradigm. For this purpose, Kriging was extended to allow the use of custom kernels and non-continuous distances, i.e., combinatorial, genotypic, or behavioral measures. Empirical studies confirmed the impact of distance measures: task-dependent distances, which significantly correlate to the underlying problem definition, provide the most accurate performance predictions. On this basis, different genotypic and phenotypic distance metrics were introduced and tested in empirical studies featuring tree-based genetic programming models, fixed-topology ANNs, as well as graph-based, topology-changing ANNs in neuroevolution. In all studies, a phenotypic distance, comparing the individuals' outputs instead of their encodings, provided the most accurate predictions and illustrated superior performance embedded in SMBO processes compared to model-free EC methods. Finally, the developed methods were extended to time-dependent reinforcement learning with a new task-dependent and precise notion of an individual's behavior. An in-depth analysis of ways to compare, control, and steer these behaviors was performed to design efficient behavioral optimization algorithms. Such an algorithm was exemplified in final studies, where behavioral optimization was included in a diverse set of neuroevolutionary optimization and ANN training algorithms. With these methods, the sampling efficiency of neuroevolutionary algorithms improved significantly.

In conclusion, this thesis illustrates the benefits of adding behavior-based search methods to EC and SMBO. The methods for searching the behavior space outlined in this work provide examples of how to overcome problems of genotypic variation in optimization, such as the complex, non-linear transitions between the genotypes and their evaluated fitness and the concomitantly challenging optimization. Behavioral optimization allows controlled adaption of individuals, robustly steering them in the direction of high-performing individuals and ultimately improving the sampling efficiency in optimizing complex tasks.

De weg van de sterkste: Optimalisatie door gedragsevolutie

Jörg Willi Stork

Vrije Universiteit

Evolutionaire berekeningsmethoden (EC) behoren tot de state-of-the-art voor het oplossen van optimalisatieproblemen met complexe karakteristieken, zoals het niet voorhanden zijn van beschikbare analytische beschrijvingen of differentieerbaarheid. Een opmerkelijk EC paradigma in toepassingen van kunstmatige intelligentie is neuro-evolutie, een methode om kunstmatige neurale netwerken (ANN) te construeren en hun gewichten, parameters, en topologieën te optimaliseren. Neuroevolutionaire optimalisatie kan een uitdaging zijn door de complexe en kostbare omgevingen waarin ze worden toegepast, multi-modale fitness landschappen, en het aantal vrijheidsgraden dat nodig is om ANNs te genereren die in deze omgevingen kunnen presteren.

Surrogaatmodelgebaseerde optimalisatie (SMBO) kan de zoekefficiëntie verbeteren door complexe fitness landschappen te benaderen en goed presterende kandidaatoplossingen te voorspellen. Surrogaatmodellen steunen vaak op afstandsmetrieken die de correlatie bepalen tussen de kwaliteit van een kandidaatoplossing en die van vergelijkbare individuen. Het gebruik van afstanden voor complex gestructureerde kandidaten, zoals ANNs, is echter moeilijk omwille van hun niet-lineaire relaties tussen hun codering (genotype) en realisatie (fenotype). Een mogelijke oplossing is om kandidaten te vergelijken op basis van hun waarneembaar gedrag, d.w.z. de beslissingen, acties en bewegingen van een individu in een omgeving. Taak-afhankelijk gedrag heeft een sterk verband met kwaliteit, wat optimalisatie in deze ruimte van gedrag veelbelovend maakt. Het centrale concept van dit proefschrift is het onderzoeken, analyseren en ontwikkelen van methoden die gebruik maken van het sterke verband tussen het gedrag van een individu, zijn omgeving en kwaliteit. De opgenomen publicaties zijn, in hun geheel, een fundamentele stap in de richting van gedragsoptimalisatie.

Eerst wordt een uitgebreide inleiding gegeven tot EC en SMBO binnen de ruimere context van globale optimalisatie, om een diepgaand begrip mogelijk te maken van hun concepten, bijzonderheden en diverse beschikbare implementaties. Er wordt een nieuwe taxonomie voorgesteld en er wordt vastgesteld dat globale optimalisatiealgoritmen nauwe verbanden hebben in hun zoekstrategieën en gelijkaardige componenten delen, wat toelaat ze te combineren in nieuwe hybride algoritmen. In grote delen van dit proefschrift werden data gedreven Kriging-modellen gebruikt als een flexibel en nauwkeurig voorspellingsmodel binnen het SMBO-paradigma. Voor dit doel werd Kriging uitgebreid om het gebruik van aangepaste kernels en niet-continue afstanden, d.w.z. combinatorische, genotypische, of gedragsmetrieken, mogelijk te maken. Empirische studies bevestigden de impact van afstandsmetrieken: taakafhankelijke afstanden, die significant correleren met de onderliggende probleemstelling, leveren de meest accurate prestatievoorspellingen op. Op deze basis werden verschillende genotypische en fenotypische afstandsmaatstaven geïntroduceerd en getest in empirische studies met boom-gebaseerde genetische programmeermodellen, ANNs met vaste topologie, alsook grafen-gebaseerde, topologie-veranderende ANNs in neuro-evolutie. In alle studies leverde een fenotypische afstand, die de outputs van de individuen vergelijkt in plaats van hun coderingen, de meest accurate voorspellingen op en illustreerde superieure prestaties als zij worden ingebed in SMBO processen in vergelijking met modelvrije EC methoden. Tenslotte werden de ontwikkelde methodes uitgebreid naar tijdsafhankelijk reinforcement learning met een nieuwe taakafhankelijke en precieze notie van het gedrag van een individu. Een diepgaande analyse van manieren om dit gedrag te vergelijken, te controleren en te sturen werd uitgevoerd om efficiënte gedragsoptimaliseringsalgoritmen te ontwerpen. Een dergelijk algoritme werd geïllustreerd in laatste studies, waar gedragsoptimalisatie werd opgenomen in een diverse set van neuro-evolutionaire optimalisatie en ANN training algoritmes. Met deze methoden werd de bemonsteringsefficiëntie van neuro-evolutionaire algoritmen aanzienlijk verbeterd.

Concluderend illustreert deze dissertatie de voordelen van het toevoegen van op gedrag gebaseerde zoekmethoden aan EC en SMBO. De methoden voor het zoeken in de gedragsruimte die in dit werk zijn geschetst, geven voorbeelden van hoe problemen van genotypische variatie in optimalisatie kunnen worden overwonnen, zoals de complexe, niet-lineaire overgangen tussen de genotypes en hun geëvalueerde kwaliteit en de daarmee gepaard gaande uitdagende optimalisatie. Gedragsoptimalisatie laat een gecontroleerde aanpassing van individuen toe, waardoor ze robuust in de richting van goed presterende individuen gestuurd worden en uiteindelijk de bemonsteringsefficiëntie bij het optimaliseren van complexe taken verbetert.